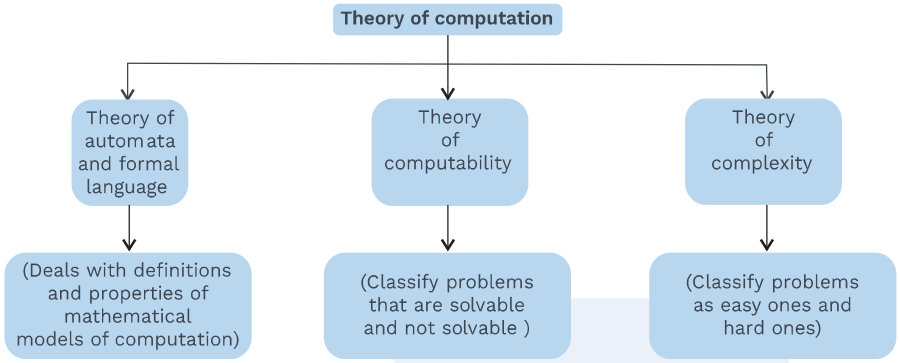
1 Introduction of Theory of Computation

1.1 INTRODUCTION OF THEORY OF COMPUTATION:

Theory of computation: The study of mathematical representation of computing machines & their capabilities.



Why study theory of computation?

It provides mathematical models of a computer:

What can/can't a computer do?

The efficiency of a computer

Finite automata: Useful models for many important kinds of hardware and software.

Pushdown automata: More capable than finite machines but less capable than liner bound automata turning machines.

Turing machines: Mathematical model of computation that defines an abstract machine, which is most powerful as it has unlimited memory.

1.2 INTRODUCTION

Symbol:

A symbol is a member of an alphabet.

OR

Symbol is an abstract entity that we shall not define formally, just as a point and lines which are not defined in geometry.

Letters and digits are examples of frequently used symbols.

Alphabet:

A finite non-empty set of symbols is known as alphabet. It is represented by $\sum$.

Example: $\sum_{binary} = \{0,$

$\sum_{decimal} = \{0, 1, 2, ...$

String (Word):
A string is finite sequence of alphabets. It is denoted by w.

Example: 1001 is a string from binary alphabet $\Sigma = \{0, 1\}$.

xyz is a string from alphabet $\Sigma = \{x, y, z\}$

Length of string:
Number of alphabets in the string.

Example: w = xyzwu
|w| = 5

Empty string:

- This is a string with no symbol at all.
- It is denoted by $\in$ (Epsilon), λ.
- Length of empty string = 0

Substring:
u is a substring of w if $\exists$ xuy = w, where u, x, y, w$\in \Sigma^*$

Example: $\underset{x}{a}\underset{u}{bcd}\underset{y}{e}$

$\underset{x}{\in}\underset{u}{mno}\underset{y}{pq}$

Note:

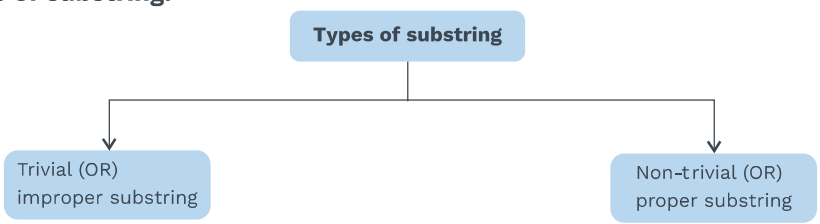
a) Every string is a substring to itself.

b) '$\in$' i.e. empty string is substring to every string.

Rack Your Brain

How many substring are there for word "COMPUTATION"?

Type of substring:



1) Trivial (OR) improper substring:
If 'w' is any string, then '$\in$' and 'w' itself is called trivial substring.

2) Non-Trivial (OR) proper substring:

If 'w' is any string over $\sum$, then any substrings of w other than '$\in$' and 'w' is called non-trivial substring.

Example: w = GATE

Trivial string = $\in$

Non-trivial string = G, A, T, E, GA, AT, TE, GAT, ATE

Note:

If 'w' is any string with a distinct symbol i.e. |w| = n, then

a) Number of substring = $\sum n+1 = \frac{n(n+1)}{2}+1$

b) Number of trivial substring = 2

c) Number of non-trivial substring = $\sum n-1 = \frac{n(n+1)}{2}-1$

Operation on string:

Prefix:

'u' is a prefix of 'w' if $\exists x$ such that ux = w where $u, x, w \in \sum^*$

Example: abcde

u

w

prefixes = $\in$, a, ab, abc, abcd, abcde

Proper prefix: A prefix of a string other than the string itself is known as a proper prefix.

In other words 'u' is a proper prefix of 'w' if u≠w.

Suffix:

'v' is a suffix of 'w' if $\exists x$ such that xv = w where $v, x, w \in \sum^*$

Example: abcde

v

w

Suffixes = $\in$, e, de, cde, bcde, abcde

Proper Suffix: A suffix of a string other than the string itself is known as a proper suffix.

In other words, 'v' is a proper suffix of 'w' if v≠w.

Rack Your Brain

If Lenth of string i.e. |w| = n, then how many number of prefixes and suffixes are possible?

Concatenation of strings:

The concatenation of two strings w and v is the string obtained by appending the symbols of v to the right end of w, that is, if

$w = a_1a_2.....a_n$

and $v = b_1b_2.....b_m$,

then the concatenation of w and v, denoted by wv, is

$wv = a_1a_2.....a_nb_1b_2.....b_m$

Let x any y be strings, then xy denotes the concatenation of x and y.

Example: Let x = 1101 and y = 011, then

xy = 1101011

and yx = 0111101

Note:

$\in .w = w. \in = w$ holds

Reverse of strings:

The reverse of a string is obtained by writing the symbols in a reverse order. If w is a string where $w = a_1a_2.....a_n$, then its reverse w^R is

$w^R = a_n.....a_2a_1$

Example: If w = abcd, then w^R = dcba

Rack Your Brain

"$W=W^R$ iff w is palindrome"

Is it true?

Power of an alphabet:

If Σ is an alphabet, then Σ^k be the set of strings of length k, each of whose symbol is in Σ.

Example: Σ = {a,b,c}

$\Sigma^0 = \{\in\}$

$\Sigma^1 = \{a,b,c\}$

$\Sigma^2 = \{aa, ab, ac, ba, bb, bc, ca, cb, cc\}$

$\Sigma^3 = \{aaa, aab, aba, aac, abb, ...ccc\}$

Σ^* = Set of all strings over an alphabet Σ

$$\Sigma^* = \bigcup_{i \geq 0} \Sigma^i = \{w \mid |w| \geq 0\}$$

$= \Sigma^0 \cup \Sigma^1 \cup \Sigma^2 \cup \ldots\ldots\ldots\ldots\ldots\ldots$ (Kleene closure)

$$\Sigma^+ = \bigcup_{i \geq 1} \Sigma^i = \{w \mid |w| \geq 1\}$$

$\Sigma^+ = \Sigma^1 \cup \Sigma^2 \cup \Sigma^3 \cup \ldots\ldots\ldots\ldots\ldots\ldots$ (Positive closure)

$$\Sigma^* = \Sigma^+ \cup \{\epsilon\}$$

Rack Your Brain

What will be the power set of Σ^*?

Basic properties of Σ^+ and Σ^*:

1) $\Sigma^+ \subset \Sigma^*$
2) $\Sigma^* = \Sigma^+ \cup \{\epsilon\}$
3) $\Sigma^+ = \Sigma^* - \{\epsilon\}$
4) $\Sigma^* \cap \Sigma^+ = \Sigma^+$
5) $\Sigma^* \cup \Sigma^+ = \Sigma^*$
6) $\Sigma^* \Sigma^+ = \Sigma^+ \Sigma^* = \Sigma^+$

1.3 GRAMMAR:

- Grammar is a set of rules to produce valid strings in a formal language.

OR

The collection of production rules is used for the formation of strings.

- A grammar G is defined as a quadruple:

  G = (V, T, S, P)

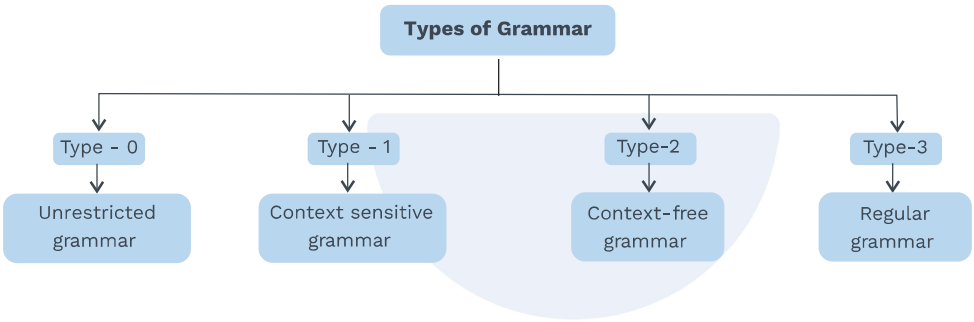
Where V is a finite set of objects called variables.
T is a finite set of objects called terminal symbols.
S $\in$ V is called the start variable.
P is a finite set of productions.

- It will be assumed that the sets V and T are non-empty and disjoint.

Example: 1) $S \rightarrow AB$ $A \rightarrow a$ $B \rightarrow b$

2) $S \rightarrow AB$ $A \rightarrow a$ $B \rightarrow b|c$

Types of grammar:



1.3.2 Chomsky hierarchy:

Chomsky's hierarchy represents the classes of formal grammar.

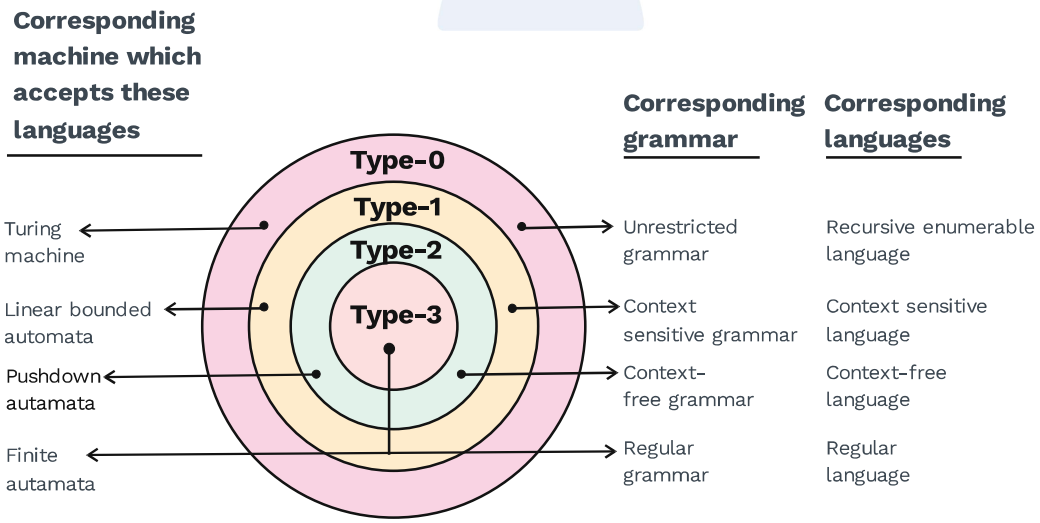


Fig. 1.1 Diagramatic Representation of Chomsky Hierarchy

1.4 LANGUAGES:

A set of strings, all of which are chosen from some Σ^*, where Σ is a particular alphabet is called a language.

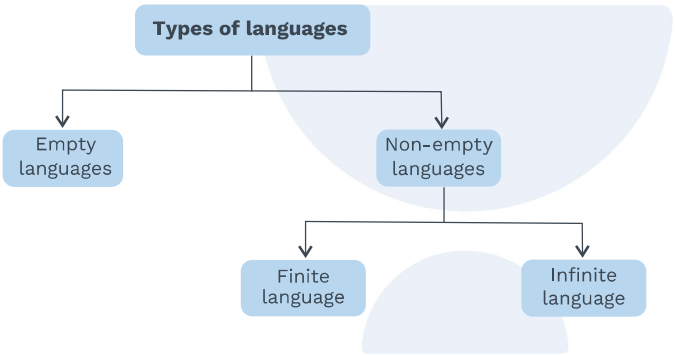
OR

Σ is any alphabet, then the set of all strings from the alphabet 'Σ' is called language.

Example: The language of all strings consisting of n number of 0's followed by n number of 1's for some $n \geq 0$.

$$L = \{\in, 01, 0011, 000111,.....\}$$

Types of languages:



Note:

Σ^* is a universal language.

Empty languages:

- Language which contains nothing i.e. $L = \phi$ does not even contain $\in$.
- Also called null language.
- Represented by ϕ.
- Cardinality of smallest language i.e. $L = \phi$ is $|L|$

Non-empty languages:

- Language which contains atleast one string is called non-empty Language.

Example:

$L = \{\in\}, |L| =$ cardinality of language

$L = \{a\}, |L| =$ cardinality of language

$L = \{a^n \mid n \geq 0\} = \{\in, a, aa.....\}$

Rack Your Brain

What will be the kleen star of an empty language?

Finite language:
The language which contains a finite number of strings where the length of each string in a language is finite.

Example:
$L = \{w \mid |w| \leq 2\}$ over $\Sigma = \{a, b\}$

$= \{\in, a, b, aa, ab, ba, bb\}$

$|L|$ = Length of language = 7

Infinite language:
The language which contains an infinite number of strings and cardinality of language is not finite.

Example:

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

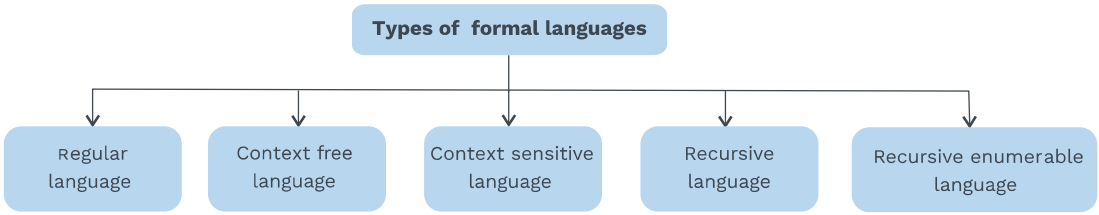
$= \{ab, aabb, aaabbb, ...\}$

Rack Your Brain

Will the complement of finite language always be infinite?

Formal language:
It is a collection of strings where the format of a string is important but meaning of a string is not important.

Types of formal language:



Regular language:

- A language is called a regular language if some finite automaton recognizes it.
- The empty language ϕ is a regular language.
- For each $a \in \Sigma$, the singleton language {a} is a regular language.
- If A is a regular language, then A* (kleen-closure) is a regular language.
- If A and B are regular languages, then $A \cup B$ and $A \bullet B$ (concatenation) are regular languages.

Example: $L = \{a^n \mid n \geq 0\}$ is a regular language.

Deterministic context-free language:

- Deterministic context-free languages (DCFL) are a proper subset of context-free languages.
- They are accepted by a deterministic pushdown automata.
- Deterministic context-free languages (DCFL) are always unambiguous. So, they accepts an unambiguous grammar.

Example: $L = \{a^n b^n c^m \mid m \geq 0 \text{ and } n \geq 0\}$

Context-free language:

- Context-free language is generated by context-free grammar (CFG).
- Most arithmetic expressions are generated by context-free grammar.

Example: $L = \{ww^R \mid w \in \{a.b\}^*\}$, w^R is a reverse of w.

Context sensitive language:

- Context sensitive language is defined by context sensitive grammar.

Example: $L = \{a^n b^n c^n \mid n \geq 1\}$

Recursive language:

- Recursive language is a subset of recursive enumerable languages.
- Recursive language can be decided by the turing machine.
- Turing machine will enter into the final state for the strings which are parts of language and enter into rejecting state for strings not in part of language.

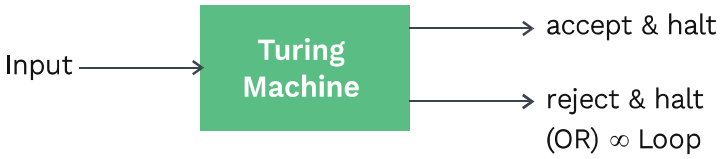
Example: $L = \{ww \mid w \in (a,b)^*\}$

Recursive enumerable language:

Recursive Enumerable Language is a formal language for which there exists a Turing Machine which will-

a) Halt and accept when any string present in the language.

b) But may either halt and reject or loop forever when the string not present in the language.



Example: $L = \{a^p \mid p \text{ is a prime number}\}$

1.5 AUTOMATA:

- Singular form of automata is known as automaton.
- Automata theory deals with the definitions and properties of mathematical models of computation.
- Automata theory is the study of abstract computing devices or machines.
- One model, called the finite automaton, is used in text processing, compilers, and hardware design. Another model, called the context-free grammar, is used in programming languages and artificial intelligence

Characteristics of such machines:

Includes:

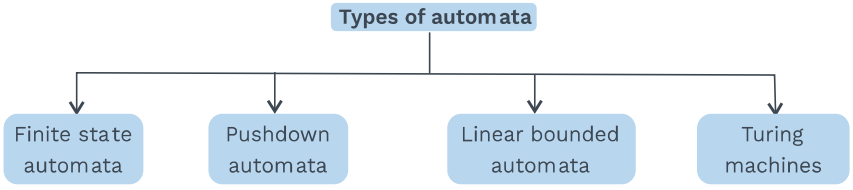
Inputs: It is assumed to be sequences of symbols selected from a finite set of input. Let $\{x_1, x_2, x_3, \ldots\ldots\ldots x_m\}$ is input set where m is the number of inputs.

Outputs: It is sequences of symbols selected from a finite set Z, where $Z = \{y_1, y_2, y_3, \ldots\ldots\ldots\ldots y_n\}$; n is number of outputs.

States: It is a finite set Q, whose definition depends on what type of automata is defined.

Types of automata:

There are 4 major families of automata:



Finite state automata (FA):

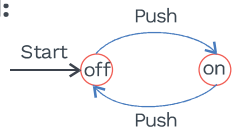
- Simplest machines.
- Finite automata are good models for computers with an extremely limited amount of memory.
- Abstract machines having 5 tuples.
- Formal specification of machines:

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

Where
- Q : Finite set of states
- Σ : Set of input symbols
- q_0 : Initial State
- F : Set of final states
- δ : Transition function

Real world examples:

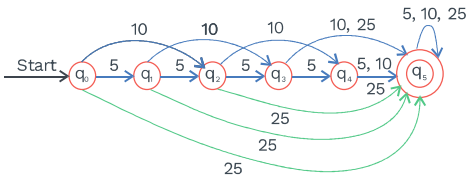
Example 1:



Odd number of pushes = ON State
Even number of pushes = OFF State

Fig. 1.2 A Finite Automaton: An On/Off Electric Switch

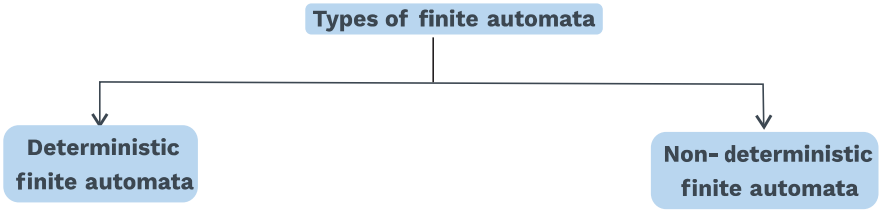
Example 2: Design a computer that controls a "toll gate". Assume that we have only 3 coins 5, 10 and 25 cents i.e. $\Sigma = \{5, 10, 25\}$. Make an assumption that any excess change amount is not refunded back. The driver, upon his arrival at the toll slots into the machine a number of coins. It is the decision of the machine if the way in is to be provided to the driver or not. The machine permits the driver to pave in if the amount is at least 25 cents.



When a car arrives at the toll gate, the gate is closed. The gate opens as soon as the driver paid 25 cents.

Fig. 1.3 A Finite Automata: Controlling a Toll Gate

Types of finite automata:



Pushdown automata (PDA):

- PDA is a finite automata with a stack.

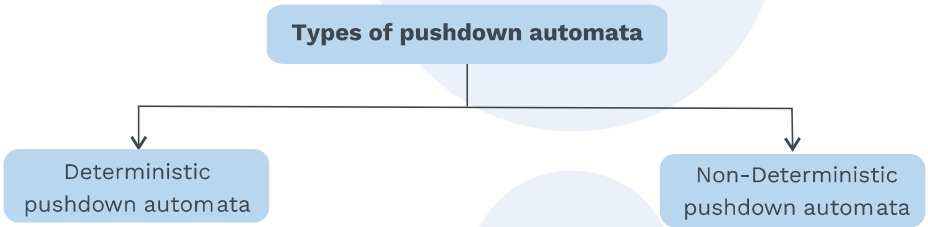
FA+1Stack=PDA

- It is used to recognize context-free languages.
- PDA is defined as:

$(Q, \Sigma, \delta, q_0, Z_0, F, \tau)$

Where
- Q : Finite set of states
- Σ : Input symbol
- δ : Transition function
- q_0 : Initial State
- Z_0 : Bottom of the stack
- F : Set of final states
- τ : Stack alphabets

Types of pushdown automata:

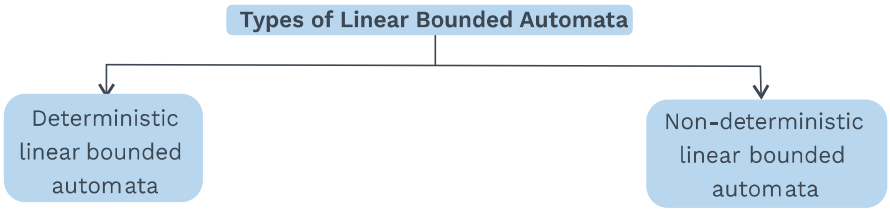


Linear bounded automata (LBA):

- A restrictive approach to the turing machine is depicted by LBA.
- A linear bounded automaton is designed with an unbounded tape. The extent up to which the tape needs to be made use of is purposely an instant of the input. Practically, the unbounded tape is restricted by the length of the input string. The read/write head is utilized to read or retrieve from the tape.
- They are classified as CSL acceptors.
- Linear bounded automata is defined as: $(Q, T, \Sigma, q_0, M_L, M_R, \delta, F)$

Where
- Q : Finite set of states
- T : Tape alphabet
- Σ : Input alphabet
- q_0 : Initial State
- M_L : Left Bound of tape
- M_R : Right Bound of tape
- δ : Transition function
- F : A finite set of final states

- **Types of linear bounded automata:**



Turing machine (TM):

- A Turing Machine is a mathematical model which consists of an infinite length tape.

FA+2Stack=PDA + 1 Stack = TM

- It consists of a head which reads input tape.
- Turing machine is hypothetical machine which can simulate any computer algorithm, however complicated it is.
- Turing machine is as powerful as a computer.

Turing machine is defined as:

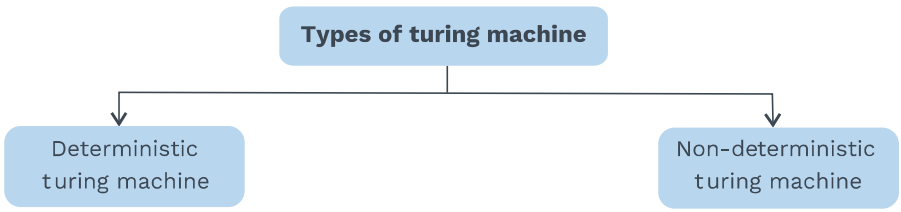
7 tuple ($Q, \Sigma, T, B, \delta, q_0, F$)

Where
- Q : Finite set of states
- T : Tape alphabet
- Σ : Input alphabet
- B : Blank symbol
- δ : Transition function
- q_0 : Initial State
- F : A finite set of final states

Rack Your Brain

Can we make any modification in standard Turing Machine and increase its power?

Types of turing machine:



Expressive power of an automata:
Number of languages accepted by that automata
i.e. E (TM) > E(LBA) > E(PDA) > E(FA)
4 3 2 1

Where E represents expressive power
TM represents turing machines
LBA represents linear bounded automata
PDA represents pushdown automata
FA represents finite automata

Expressive power of DFA and NFA:

Expressive power of DFA = Expressive power of NFA

i.e. DFA and NFA define the same class of languages.

Expressive power of DPDA and NPDA:

Expressive power of DPDA < Expressive power of NPDA

i.e. DPDA defines a class of language which is a proper subset of the class of languages defined by NPDA.

Expressive power of DLBA and NLBA:

Expressive power of DLBA $\overset{?}{=}$ Expressive power of NLBA ↓ Can't say (Not known)

i.e. It is not known whether deterministic linear bounded automata (DLBA) is more powerful or NLBA (Non-deterministic linear bounded automata).

Expressive power of DTM and NTM:

Expressive power of DTM = Expressive power of NTM

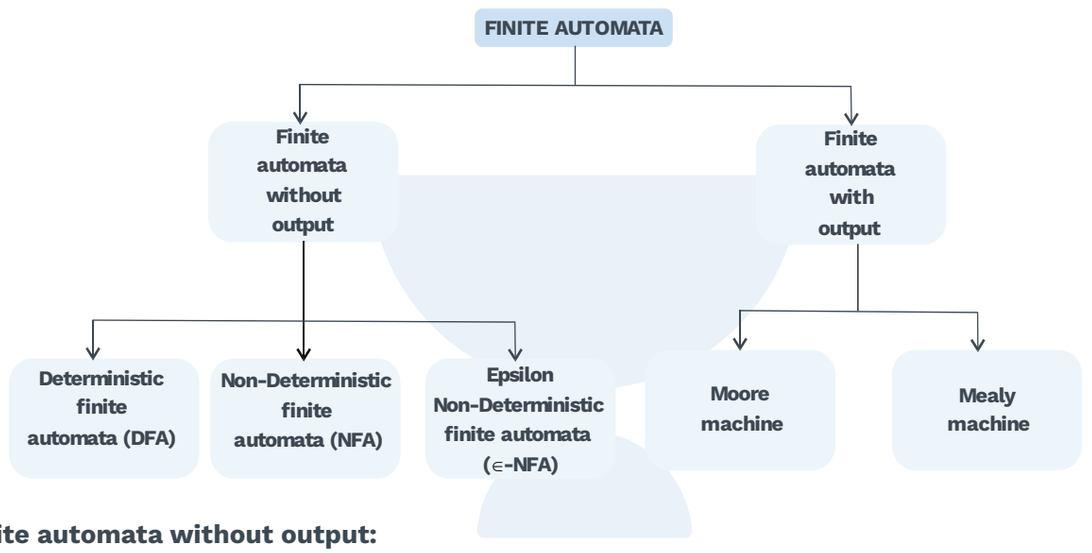
i.e. Deterministic turing machine (DTM) and non-deterministic turing machine (NTM) are equivalent in power.

2 Finite Automata

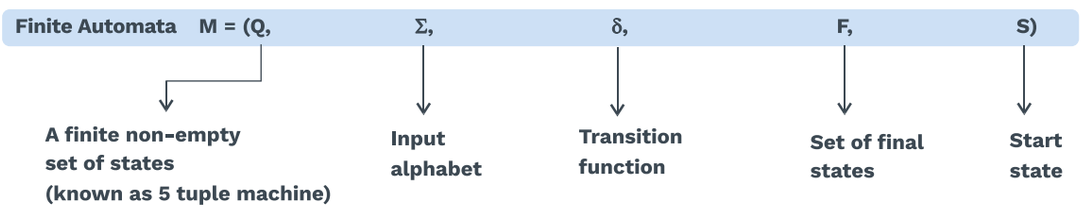
2.1 FINITE AUTOMATA

Definitions

It is a mathematical model which contains a finite number of states and transitions among states in response to inputs.



Finite automata without output:



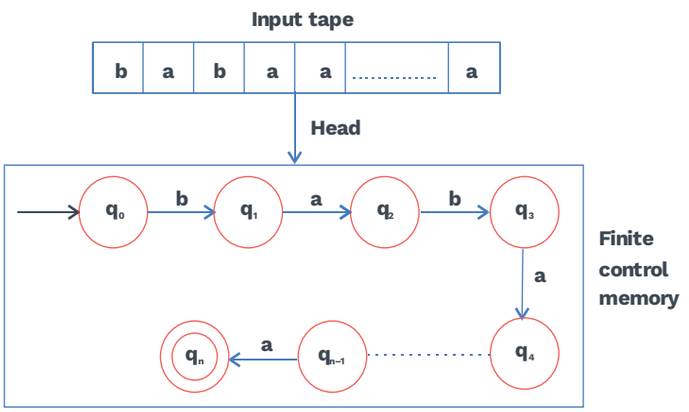


Fig. 2.1 Diagram of Finite Automata

Input tape: It consists of many cells where each cell can have only one symbol. Input tape contains a single string.

Head: It reads one symbol at a time from an input string.

Finite control memory: Memory which is having the finite number of states.

2.2 FINITE ACCEPTOR (FINITE AUTOMATA WITHOUT OUTPUT)

Deterministic finite automata (DFA)

- A DFA is set of 5 tuple and defined as:

$$M = \{Q, \Sigma, \delta, q_0, F\}$$

Where
- Q : Finite set of states
- Σ : Input alphabet
- δ : Transition function
- q_0 : Initial State
- F : A finite set of final states

Here
- δ : Transition function

$$\boxed{Q \times \Sigma \rightarrow Q}$$

↓

Next State

$\{A, B, C\} \times \{a, b\} \rightarrow$ Only deterministic output will come.

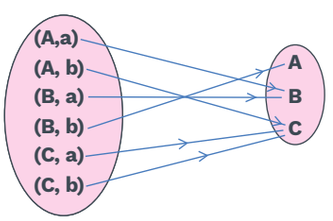
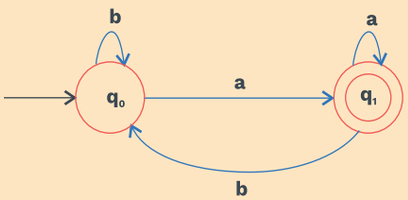


Fig. 2.2

- DFA is a finite Automata in which, from every state on every input symbol, exactly one transition should exist.
- DFA cannot use empty string transition.
- In DFA, if any string terminates in a state which is different from accepting state, then DFA rejects that string.
- All DFA are NFA.

SOLVED EXAMPLES

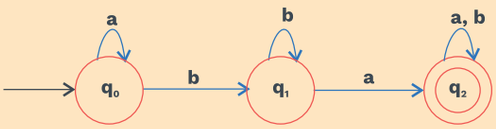
Q1



Language accepted by above Finite Automata?

Sol: L = {a, aa, aaa, aaaa, ba, bbbba, bababaaaa,.....} = Set of all strings ends with 'a' over $\sum$ = {a, b}

Q2



Language accepted by above Finite Automata?

Sol: L = {ba, bab, baa, aba, abaa, abab, ...} = Set of all strings containing "ba" as a substring over $\sum$ = {a, b}.

Note:

- **When a string is accepted by DFA?**

  Upon Scanning the string if we reach final state from initial state, then a string is accepted by DFA

- **When a language is accepted by DFA?**

  A language is accepted by DFA

i) When all strings present in the language are accepted by DFA

AND

ii) When all strings not present in the language are rejected by DFA.

Rack Your Brain

How many number of final state require to accept & in minimal finite automata?

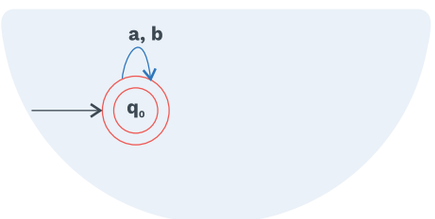
Minimal DFA: When all the states in a DFA are distinguishable, i.e. no two states in a final DFA are equivalent. The DFA is known as minimal DFA.

SOLVED EXAMPLES

Q1 **Construct minimal DFA that accepts all strings of 'a's and b's including ∈.**

Sol: L = {∈, a, b, aa, bb, ab, ba, aaa, ...}

DFA:



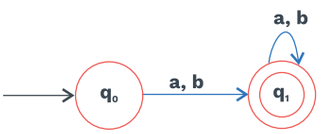
Transition table:

δ	a	b
→ q_0	q_0	q_0

Q2 **Construct minimal DFA that accepts all strings of 'a's and b's excluding ∈.**

Sol: L = {a, b, ab, ba, aa, bb, ...}

DFA:



Transition table:

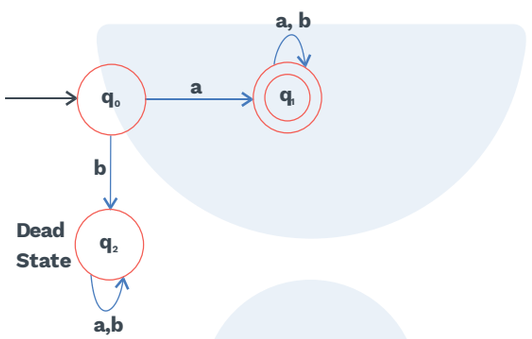
	a	b
→ q_0	q_1	q_1
q_1	q_1	q_1

Q3 **Construct minimal DFA that accepts all strings of 'a's & b's where every string "starts with a".**

Sol:

$$L = \{a, aa, aaa, aaaa, \ldots\ldots\ldots\ldots, ab, aab, aba, abb, \vdots\}$$

DFA:



- If any string starts with 'a', then it will be accepted by the DFA.

Let's consider an example.
String w = aab
$\delta(q_0, a) = q_1$
$\delta(q_1, a) = q_1$
$\delta(q_1, b) = q_1$ (accepted by DFA)

- But if the string starts with 'b', it directly goes to the dead state and get rejected by the DFA.

Let's consider an example.
String w = ba
$\delta(q_0, b) = q_2$
$\delta(q_2, a) = q_2$ (Dead state)
Thus, string ba is rejected (not accepted by DFA).

Dead state: It is a rejecting state i.e a non final state from which there is no path to the final state.
Once we reach to dead state, there is no way to reach the final state.

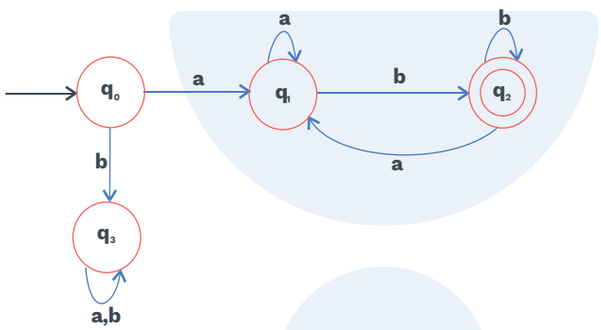
Transition table:

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

Q4 **Construct a minimal DFA that accepts all the strings of a's and b's where "each string starting with a and ending with b".**

Sol: L = {ab, aab, abb, aaab, abab, aabb, ...}

DFA:



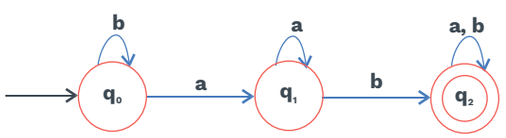
Transition table:

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

Q5 **Construct a minimal DFA that accepts all the strings of a's and b's where "each string contains <u>ab</u> as substring".**

Sol: L = {ab, aba, aab, ...}

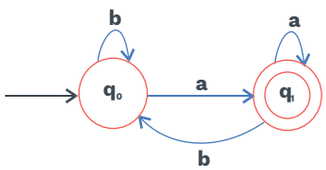
DFA:



Q6 **Construct a minimal DFA that accepts all the strings of a's and b's where "each string ends with a".**

Sol: L = {a, aa, ba, aaa, aba, ...}

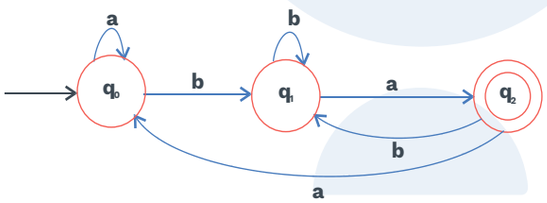
DFA:



Q7 **Construct a minimal DFA that accepts all the strings over $\sum$ = {a, b} where "each string ends with ba".**

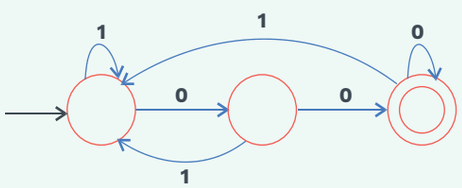
Sol: L = {ba, aba, bba, aaba, ...}

DFA:



Previous Years' Question

(GATE 2009)



The above DFA accepts the set of all strings over {0, 1} that

1) Begin either with 0 or 1 **2)** End with 0

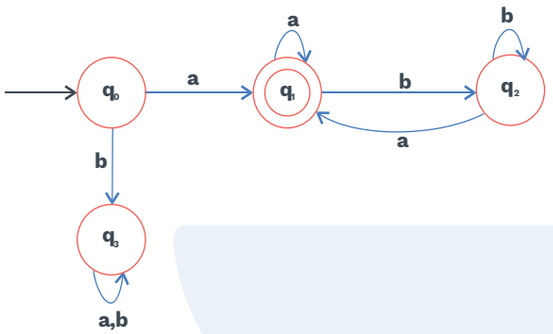
3) End with 00 **4)** Contain the substring 00

Sol: 3)

Q8 **Construct a minimal DFA that accepts all strings over $\sum$ = {a, b} where "each string starting and ending with a".**

Sol: L = {a, aa, aba, aaa, ...}

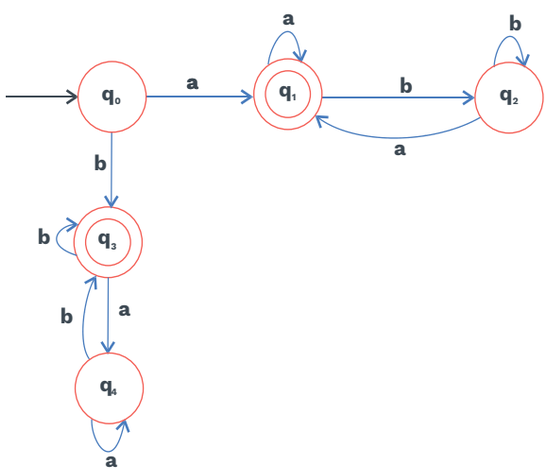
DFA:



Q9 **Construct a minimal DFA that accepts all the strings over $\sum$ = {a, b} where "each string starting and ending with same symbol".**

Sol: L = {a, b, aaa, bbb, aba, bab, ...}

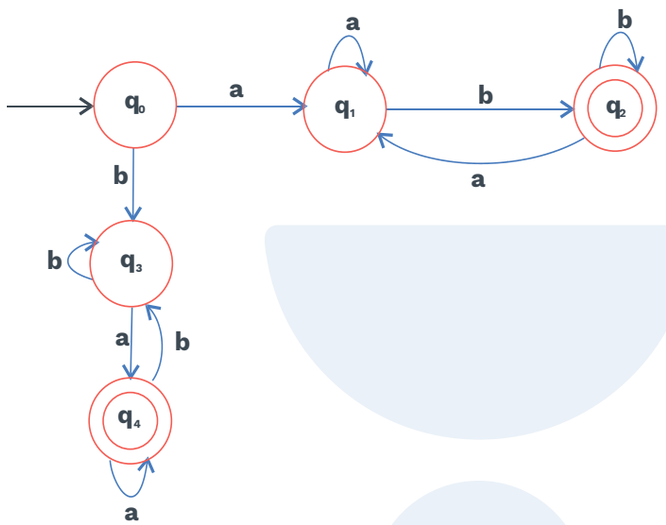
DFA:



Q10 **Construct a minimal DFA that accepts all the strings over ∑ = {a, b} where "each string starting and ending with different symbol".**

Sol: L = {ab, ba, aab, abb, baa, ...}

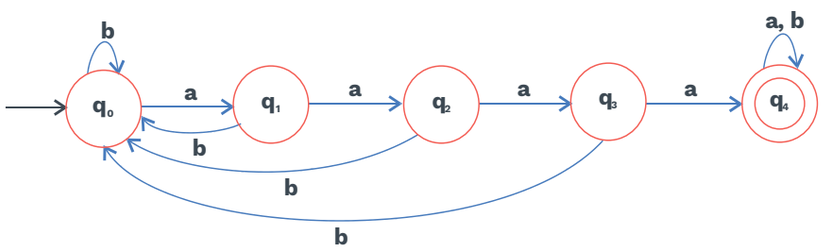
DFA:



Q11 **Construct a minimal DFA that accepts all the strings over ∑ = {a, b} where "each string contains aaaa as substrings".**

Sol: L = {aaaa, baaaa, aaaab, aaaaa, ...}

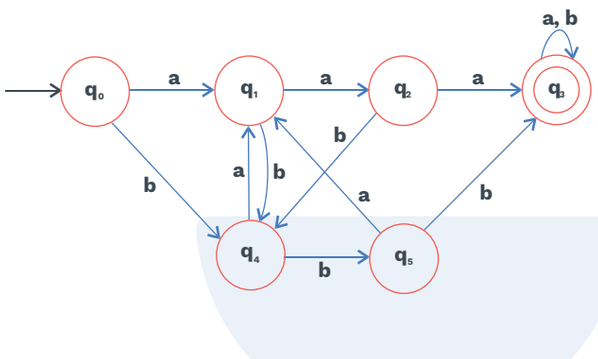
DFA:



Q12 **Construct a minimal DFA that accepts all the strings over $\sum$ = {a, b} where "each string contains aaa (or) bbb as substrings".**

Sol: L = {aaa, baaa, abbb, bbba, ...}

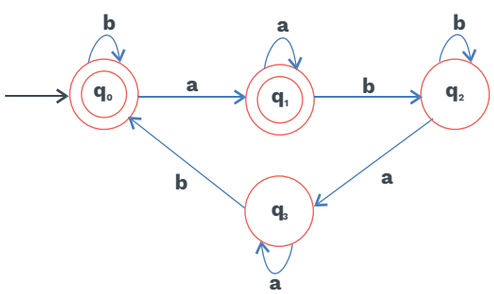
DFA:



Q13 **Construct a minimal DFA that accepts all the strings over $\sum$ = {a, b} where "number of occurrence of substring "ab" is even".**

Sol: L = $\{\in, abab, ababab, ...aa, bb, aaa, ...\}$

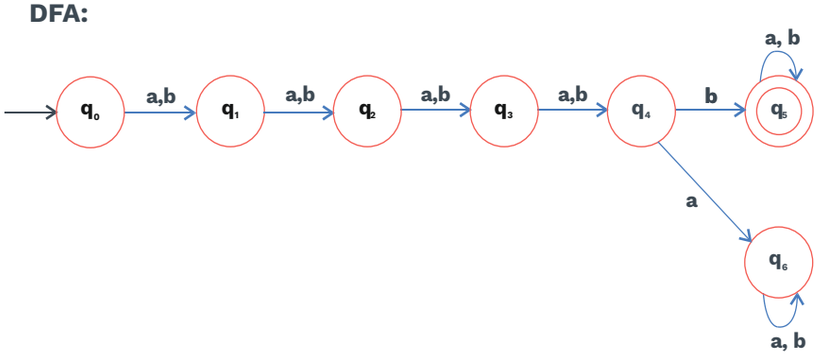
DFA:



Q14 **Construct a minimum DFA that accepts all the strings over $\sum$ = {a, b} where 5th symbol from left is 'b'.**

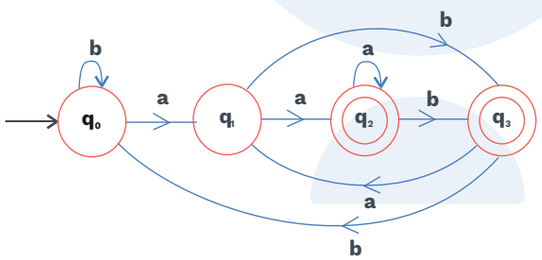
Sol: L = {aaaab, baaab, ...}

DFA:



Q15 **Construct a minimum DFA that accepts all the strings over ∑ = {a, b} where 2nd symbol from RHS is 'a'.**

Sol:



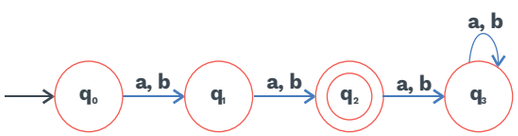
Q16 **Construct a minimum DFA that accepts all the strings over ∑ = {a, b} such that**

i) Length of string is exactly 2 i.e. |w| = 2.

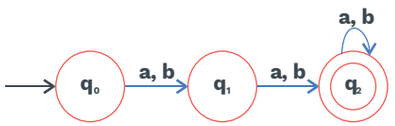
ii) Length of string is atleast 2 i.e. |w| ≥ 2

iii) Length of string is atmost 2 i.e. |w| ≤ 2

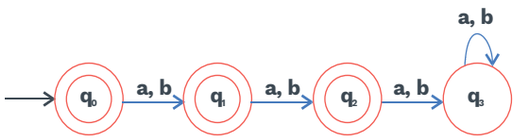
Sol: **i)** Length of the string is exactly 2 i.e. |w| = 2



ii) Length of the string is atleast 2 i.e. $|w| \geq 2$



iii) Length of the string is at most 2, i.e. $|w| \leq 2$



Note:

- If length of strings is exactly n i.e. $|w| = n$ over $\Sigma = \{a, b\}$, then number of states to construct minimum DFA = (n + 2) states.
- If length of strings is atleast n i.e. $|w| \geq n$ over $\Sigma = \{a, b\}$, then number of states to construct minimum DFA = (n + 1) states.
- If length of strings is atmost n, i.e. $|w| \leq n$ over $\Sigma = \{a, b\}$, then number of states to construct minimum DFA = (n + 2) states.

Rack Your Brain

If L is a non-empty language such that any win L has length atleast n, then any DFA accepting L must have atleast n +1 states. this statement true (or) false?

Q17 **Construct a minimal DFA which accepts all the strings over $\Sigma = \{a,b\}$ such that**

i) Length of each string is even

ii) Length of each string is odd

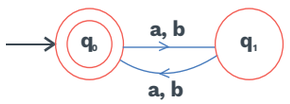
iii) Length of each string is divisible by 3

Sol: **i)** All even length strings over $\Sigma = \{a, b\}$

$|w| \bmod 2 = 0$

$L = \{\in, aa, bb, ab, \ldots\ldots\ldots\ldots\}$

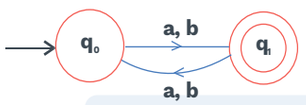
DFA:



ii) All odd length strings over $\sum = \{a, b\}$

Odd length strings meSol: |w| mod 2 = 1

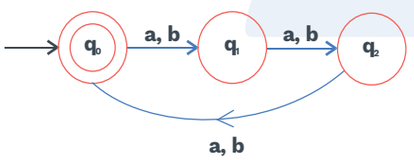
DFA:



iii) All strings whose length is divisible by 3 over $\sum = \{a, b\}$

|w| mod 3 = 0

$$L = \left\{ \begin{array}{ll} \in, & aaa, aaaaaa, ... \\ & aab \quad \vdots \\ & \vdots \quad \vdots \\ & bbb \quad \vdots \end{array} \right\}$$



Note:

If |w| mod n = m, then in minimum DFA, number of states = n and final state is q_m [th] states.

Q18 **Construct a minimal DFA that accepts all the strings over $\sum$ = {a, b} such that**

i) **$n_a(w) = 2$ i.e. number of a's in string = 2**

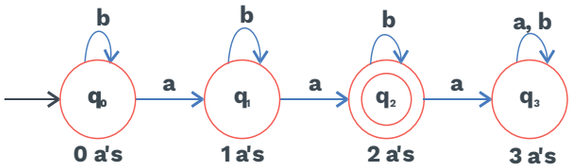
ii) **$n_a(w) \geq 2$ i.e. number of a's in string is atleast 2**

iii) **$n_a(w) \leq 2$ i.e. number of a's in string is atmost 2**

Sol:

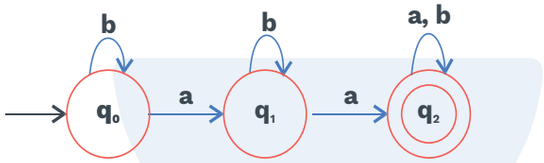
i) L = {aa, baa, aba, aab, bbaa,.....}

DFA:



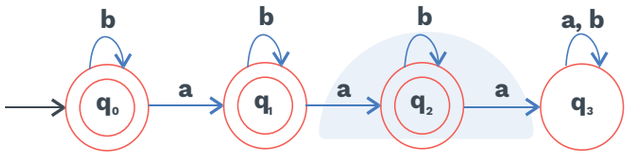
ii) L = {aa, aba, aaa, aaab,.....}

DFA:



iii) L = {∈, a, ab, aa, aba,.....}

DFA:



Note:

The minimal deterministic finite automata that accepts all the strings of a's and b's where each string contains:

- Exactly n number of a's has 'n+2' states.
- Atleast n number of a's has 'n+1' states.
- Atmost n number of a's has 'n+2' states.

Grey Matter Alert!

Number of states in minimal DFA which accepts all the strings over $\Sigma = \{a, b\}$ such that:

a) Number of a's is 'atleast m' and number of b's is 'atleast n' = (m + 1)(n + 1)

b) Number of a's is 'exactly m' and number of b's is 'exactly n' = (m + 1) (n + 1) + 1 (because of trap state, we require one more state).

c) Number of a's is 'atmost m' and number of b's is 'atmost n' = (m + 1)(n+1) + 1

Previous Years' Question

The minimum possible number of states of a deterministic finite automaton that accepts the regular language

$L = \{w_1 a w_2 \mid w_1, w_2 \in \{a, b\}^*, \mid w_1 \mid = 2, \mid w_2 \mid \geq 3\}$ is _____

Sol: Range: 8 to 8 **(GATE 2017 (Set-2))**

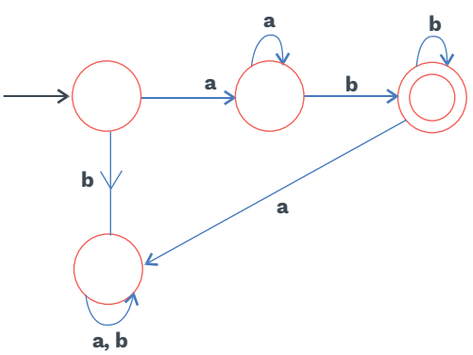
SOLVED EXAMPLES

Q19 **Construct a minimal DFA which accepts $L = \{a^n b^m \mid n, m \geq 1\}$**

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

$= \{ab, aab, abb, aabb, \ldots\ldots\ldots\ldots\}$

DFA:

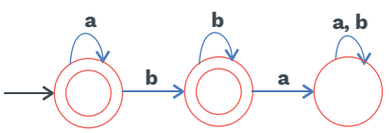


Q20 **Construct a minimal DFA which accepts L = {$a^n b^m$ | n, m ≥ 0}**

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

$= \{\in, a, b, ab, aa, bb, ...\}$

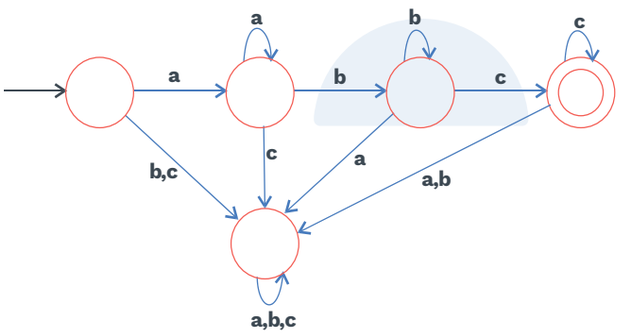
DFA:



Q21 **Construct a minimal DFA which accepts L = {$a^n b^m c^l$ | n, m, l ≥ 1}**

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

$= \{abc, aabc, abbc, abcc, ...\}$



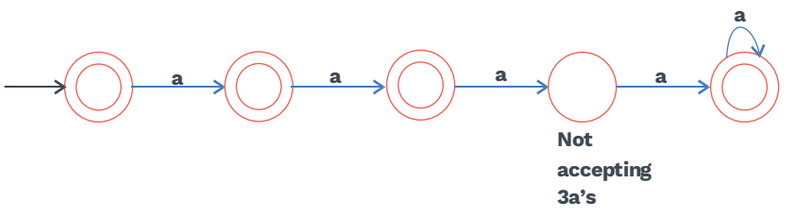
Q22 **Construct a minimal DFA over {a} :**

i) L = {a^n | n ≥ 0, n ≠ 3}

ii) L = {a^n | n ≥ 0, n ≠ 2, n ≠ 4}

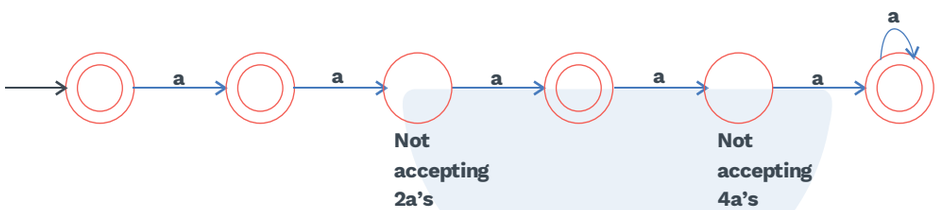
Sol: **i)** $L = \{a^n \mid n \geq 0, n \neq 3\}$

$= \{\in, a, aa, aaaa, ...\}$



ii) $L = \{a^n \mid n \geq 0, n \neq 2, n \neq 4\}$

$= \{\in, a, aaa, aaaaa, ...\}$



Rack Your Brain

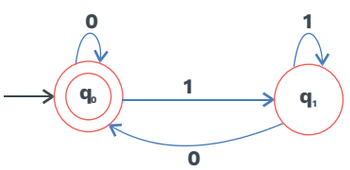
Construct the minimal DFA's for the language L = (ab | n20) u (b'a | n21}

SOLVED EXAMPLES

Q1 **Construct minimal state DFA that accept set of all binary numbers whose decimal equivalent is divisible by 2.**

Sol: Divisible by 2, so two remainders are 0, 1

DFA:



$(101)_2 = (5)_{10}$ = Not accepted by DFA:

$(110)_2 = (6)_{10}$ = Accepted by DFA

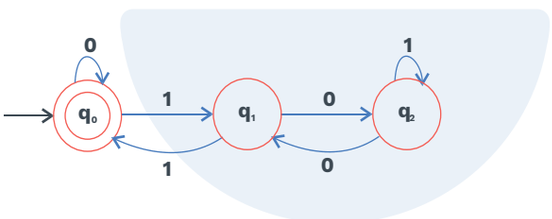
Transition table:

	0	1
→ q_0	q_0	q_1
q_1	q_0	q_1

Q2 **Construct minimal state DFA that accepts set of all binary numbers whose decimal equivalent is divisible by 3.**

Sol: Divisible by 3, so remainders are 0, 1, 2

DFA:



Transition table:

	0	1
→ q_0	q_0	q_1
q_1	q_2	q_0
q_2	q_1	q_2

Q3 **Construct minimal DFA that accepts set of all binary numbers whose decimal equivalent is divisible by 4.**

Sol: Divisible by 4, so remainders are 0, 1, 2, 3

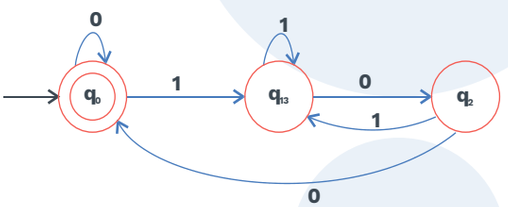
Transition table:

δ	0	1
→ q_0	q_0	q_1
q_1	q_2	q_3
q_2	q_0	q_1
q_3	q_2	q_3

Here, we can merge q_1 and q_3 states as one state.

δ	0	1
→ q_0	q_0	q_{13}
q_{13}	q_2	q_{13}
q_2	q_0	q_{13}

DFA:



Q4 **Number of states in minimal DFA that accepts all binary strings which are congruent to 2 mod 7 i.e. binary number ≡ 2 (mod 7).**

Sol: **Transition table:**

	0	1
→ q_0	q_0	q_1
q_1	q_2	q_3
q_2	q_4	q_5
q_3	q_6	q_0
q_4	q_1	q_2
q_5	q_3	q_4
q_6	q_5	q_6

⇒ Here two rows cannot be merged. Since all are different.

So, the minimum DFA will contain 7 states.

Q5 **Number of states in minimal DFA that accepts set of all binary strings whose decimal equivalent is divisible by 6.**

Sol: This problem belongs to below NOTE (Case 3).
Here n = 6 (even) but not in power of 2.
$n = 2^1 * 3$
So, minimal DFA contains (1 + 3) = 4 states
We can also prove it in descriptive way.
Transition table:

	0	1
→ q_0	q_0	q_1
q_1	q_2	q_3
q_2	q_4	q_5
q_3	q_0	q_1
q_4	q_2	q_3
q_5	q_4	q_5

⇒ Here, we can merge states q_1 and q_4 as q_{14}.
Similarly, we can merge states q_2 and q_5 as q_{25}.
Hence, transition Table will become:

	0	1
→ q_0	q_0	q_{14}
q_{14}	q_{25}	q_3
q_{25}	q_{14}	q_{25}
q_3	q_0	q_{14}

Note:

To construct minimal DFA, which accepts all binary numbers such that Binary number ≅ r (mod n), where r is remainder.

- **Case 1**

  If n is odd, then Minimal DFA will contain 'n' states.

- **Case 2**

  If n = even & if we can express n such that $n = 2^k$, then minimal DFA will contain '(k + 1) states'.

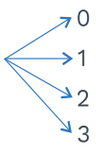
- **Case 3**

  If n = even, but $n \neq 2^k$, then if we can express n such that n = (2k) * p, where p is prime factor of n. In this case, minimal DFA contains '(k + p) states'.

SOLVED EXAMPLES

Q1 **Construct a minimal DFA that accepts all ternary numbers which are divisible by 4.**

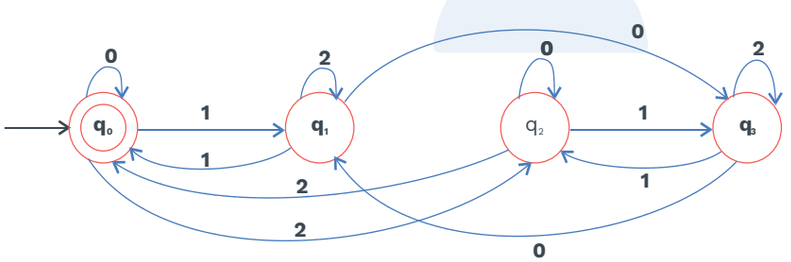
Sol: Divisible by 4, so remainders are



Input = {0, 1, 2}

Transition table:

	0	1	2
→ q_0	q_0	q_1	q_2
q_1	q_3	q_0	q_1
q_2	q_2	q_3	q_0
q_3	q_1	q_2	q_3



Operations on DFA:

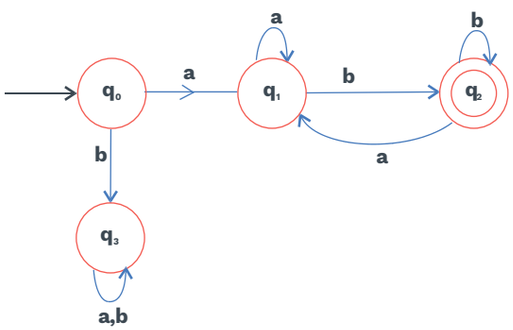
Union:

DFA which accepts all strings starting and ending with different symbols over $\Sigma = \{a, b\}$.

L_1 = {ab, aab, abb, ...}

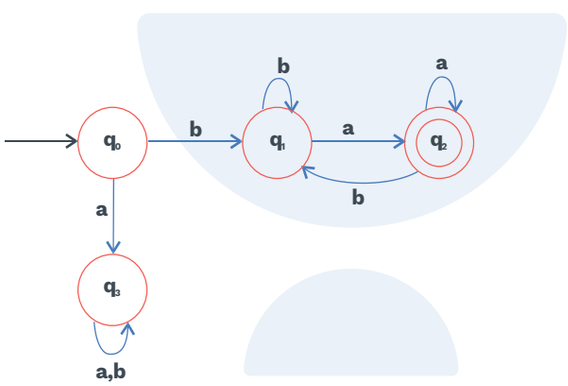
Starts with 'a' and ends with 'b'.

DFA:

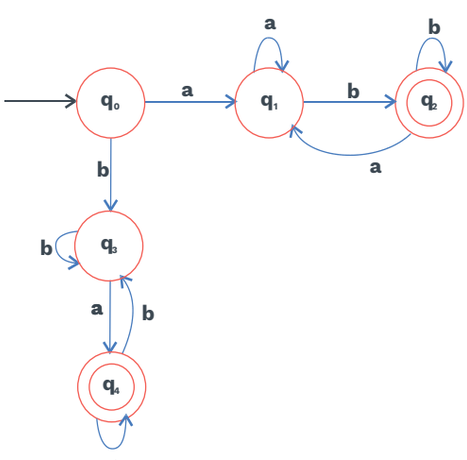


L_2 = {ba, baa, bba, ...}
= Starts with b and ends with a

DFA:



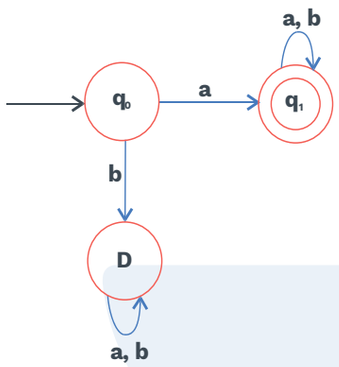
DFA for $L_1 \cup L_2$:



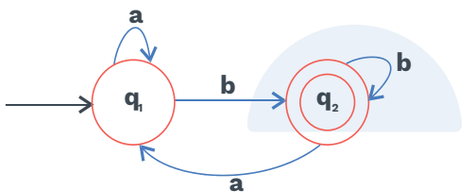
Concatenation:

DFA that accepts all strings over $\Sigma = \{a, b\}$ which starts with a and ends with b.

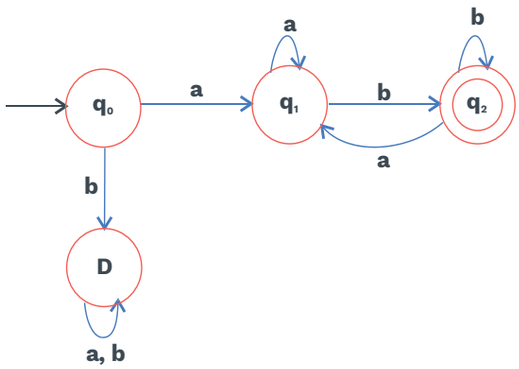
L_1 = {Starting with a}
= {a, aa, ab, aab, ...}



L_2 = {Ending with b}
= {b, ab, bb, aab, ...}



Now DFA for $L_1.L_2$:



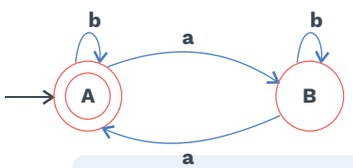
Cross product:

SOLVED EXAMPLES

Q1 **Construct a minimal DFA which accepts set of all strings over $\sum$ = {a, b} such that string of the language contain even number of a's and even number of b's.**

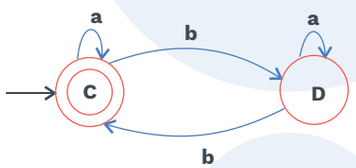
Sol: L_1 = {even number of a's} = {∈, aa, baa, ...}

DFA:



L_2 = {even number of b's} = {∈, bb, abb, ...}

DFA:



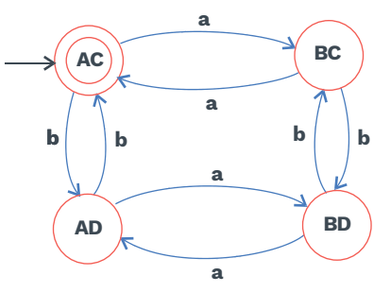
Cross product:

{A, B} × {C, D} = {AC, BC, AD, BD}

Where final state = {AC} because A and C are both final states in respective DFA for L_1 and L_2.

Initial State = {AC} because A and C are both initial states in respective DFA for L_1 and L_2.

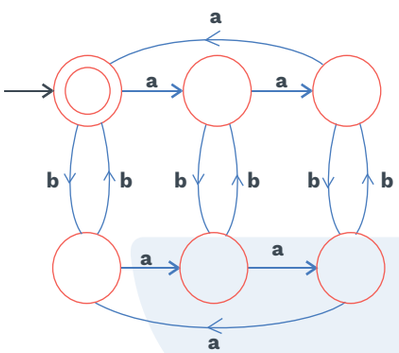
DFA for cross product:



Q2 **Construct a minimal DFA which accepts set of all strings over $\sum$ = {a, b} in which number of a's are divisible by 3 and number of b's are divisible by 2.**

Sol: $n_a(w) \cong 0 \bmod 3$ & $n_b(w) \cong 0 \bmod 2$

DFA:



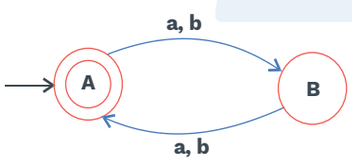
Total number of states = 3 × 2 = 6

Complementation:

eg 1: L = set of all strings over $\sum$ = {a, b} of even length

$= \{\in, ab, aa, bb, aaaa, \ldots\ldots\ldots\ldots\ldots\ldots\}$

Sol: **DFA**:



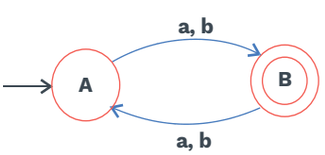
Now, L_2 = set of all strings over $\sum = \{a, b\}$ of odd length

= {a, b, aaa, aab, ...}

$$L_2 = \overline{L}_1 = \text{Complement of } L_1$$

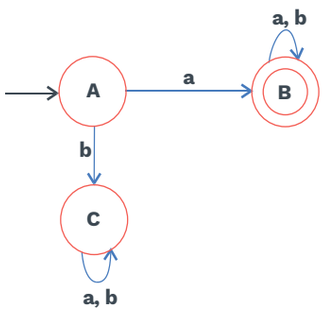
DFA for L_2:

Just complement the above DFA i.e. change the final state as Non-final state and Non-final state as Final State



eg 2: L = Set of all strings over $\sum$ = {a, b} starting with 'a'
= {a, aa, ab, ...}

Sol: **DFA**:

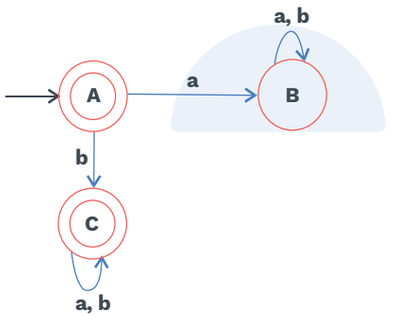


$L_2 = \overline{L}_1$ = Complement of L_1

= Set of all strings over $\sum$ = {a,b} not starting with 'a'

= {$\in$, b, ba, ...}

DFA for complement of L_1:



Note:

If a DFA is defined by (Q, Σ , δ, q_0, F) and it accepts the language L_1. Then, the DFA which accepts complement of language L_1, that is $L_2 = L_1$, will be defined as (Q, Σ , δ, q_0, Q – F).

DFA $\longrightarrow$ (Q, Σ, δ, q_0, F)

Complement

(Q, Σ, δ, q_0, Q–F)

Previous Years' Question

Let L be the language represented by the regular expression $\Sigma^*0011\Sigma^*$ where $\Sigma = \{0,1\}$

What is the minimum number of states in a DFA that recognizes $\overline{L}$ (complement of L)? **(GATE 2015 SET 3)**

1) 4 **2) 5**
3) 6 **4) 8**

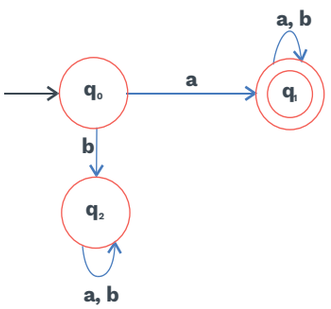
Sol: 2)

Reversal:
Reversing a language meSol: reverse all strings present in the language.
Steps:
a) Draw the state as it is.
b) Make final state as initial state and initial state as final state.
c) Reverse the edges.
d) Remove all the unreachable state and its transition from initial state.

Note:
Not every reversal of DFA Result into DFA.

DFA for the set of string over $\Sigma = \{a,b\}$ which start with 'a'.

L_1 = {a, aa, ab, aab,.....}
DFA:



Now, Let L_2 = Reversal of L_1 = {a, aa, ba, baa, ...}

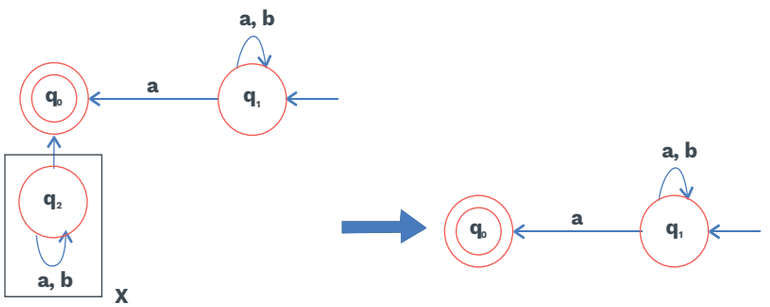


Fig 2.3

Previous Years' Question

Consider the following language: **(GATE-2020)**

L = {x ∈ {a, b}*| number of a's in x divisible by 2 but not divisible by 3}

The minimum number of states in DFA that accepts L is _____.

Sol: Range = 6 to 6

Q1 **Counting DFA:**

Number of 2-state DFA with designated initial state that can be constructed over $\Sigma = \{a, b\}$.

Sol:

- Final State can be : Zero state i.e. None of them final
  : One state i.e. either q_0 final state or q_1 final state
  : Two states i.e. q_0 and q_1 both final state

So, Total $2^2 = 4$ possibility.

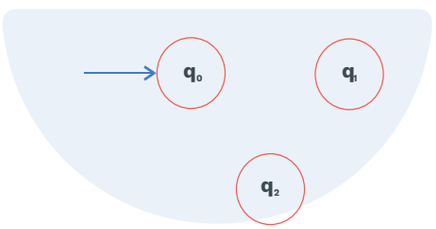
δ	a	b
→ q_0	q_0/q_1	q_0/q_1
q_1	q_0/q_1	q_0/q_1

2^4 possibility
(i.e. either go to the state q_0 or q_1)

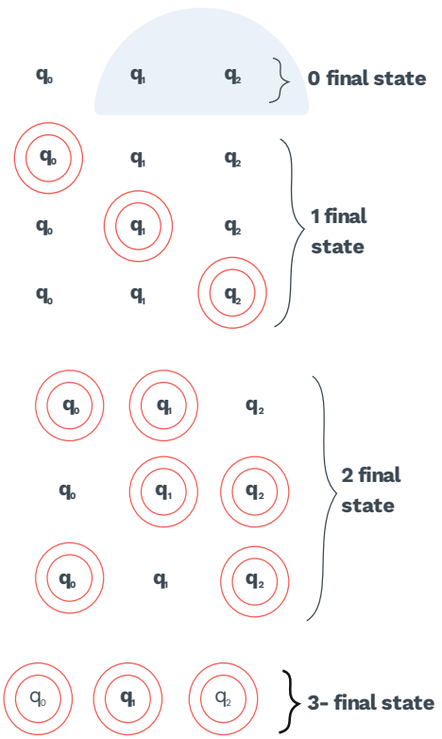
- q_0 on 'a' can either go to the state q_0 or q_1. q_0 on 'b' can either go to the state q_0 or q_1.
- Similarly, q_1 on 'a' can either go to the state q_0 or q_1. q_1 on 'b' can either go to the state q_0 or q_1.
  Total number of ways to construct 2-state DFA with designated initial state over $\sum = \{a, b\}$ = $2^2 * 2^4 = 2^6 = 64$

Q2 **Number of 3-state DFA with designated initial state that can be constructed over $\sum = \{a, b\}$.**

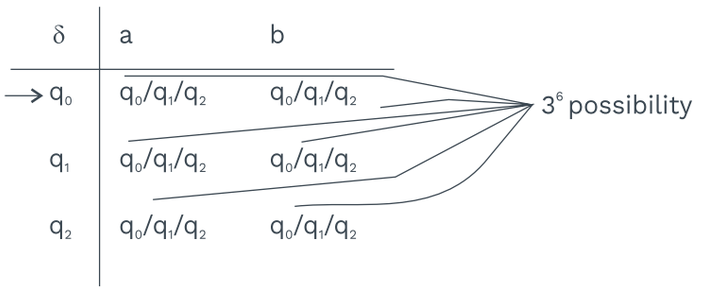
Sol:



- Final state can be:



Total possibility = 2^3 = 8 possibility



Total number of ways to construct 3-state DFA with a designated initial state over $\sum = \{a, b\}$ = $2^3 * 3^6 = 8 * 9^3 = 5832$

Note:
Number of n-state DFA with a designated initial state over alphabet containing m symbols.

Given: $|\sum| = m$

|Q| = number of states = n

- Number of ways, we can construct final states = 2^n

δ	1	2	3.....m
1	n	n	n .....n
2	n	n	n .....n
3	n	n	n .....n
.	.	.	. .....
.	.	.	. .....
.	.	.	. .....
n	n	n	n .....n

Total cells in $\delta = n \times m$
$= m \times n$
and each cell can be filled in 'n ways'
So, Total possibility = $n^{m \times n}$

$$Q \times \sum \rightarrow 2^Q$$

Total number of ways to construct n-state DFA with a designated initial state over

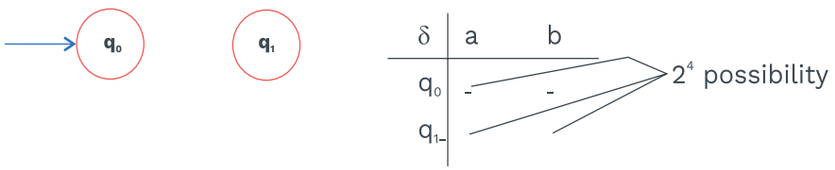
$\sum$ = {1, 2, ...m} containing m symbols

= $2^n \times n^{m \times n}$

Q3 **Number of 2-state DFA's with a designated initial state accepting ϕ over $\Sigma = \{a, b\}$**

Sol:

Zero final state



Since neither q_0 nor q_1 is final state. So, it accepts ϕ in 16 ways.

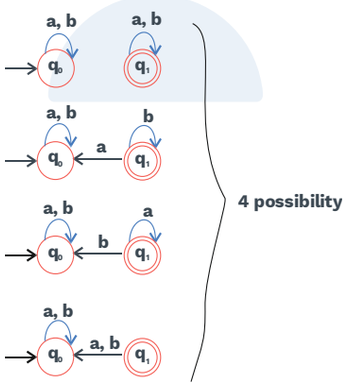
Case 2: One final states

1) 

It accepts $\in$. Thus, this cannot be the required case.

2) 

It can be the desired case but not all transition on it is desirable.



Case 3: Two final states

It cannot be the described case as it is accepting $\in$.

$\therefore$ Overall total number of 2-state DFAs with designated initial state accepting ϕ.

Over $\Sigma = \{a, b\}$ = (16 + 4) = 20

Q4 **Number of 2-state DFA with a designated initial state accepting $\sum^*$ (universal language) over $\sum = \{a, b\}$.**

Sol:

Case 1: Zero final state

It is not a favourable case as it accepts ϕ. It cannot accept $\sum^*$.

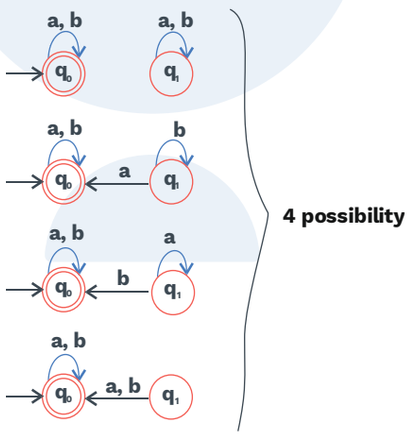
Case 2: One final state

1) 

It will not accept $\in$ which is part of $\sum^*$. Thus, it cannot accept $\sum^*$. Not favourable case.

2) 

It can be favourable case.



4 possibility

Case 3: 2 Final state

δ	a	b
→ q_0	q_0/q_1	q_0/q_1
q_1	q_0/q_1	q_0/q_1

2^4 possibility

When both q_0 and q_1 states are final states, then it will accept $\sum^*$ in 16 possible ways.

$\therefore$ Overall, total number of 2–state DFA with designated initial state accepting $\sum^*$ over $\sum = \{a, b\}$ = 4 + 16 = 20

Minimization of DFA:

Definitions

Minimization of DFA meSol: trSol:forming a given DFA into an equivalent DFA that has a minimum number of states.

i) Remove all the unreachable states (i.e. states which are not reachable from initial state).

ii) Separate all final states in one set and all non-final states in another set.

iii) Two states p and q of a DFA are called Indistinguishable if $\delta^*(p, w) \in$ Final state

$\Rightarrow \delta^*(q, w) \in$ Final State

and $\delta^*(p, w) \notin$ Final State $\Rightarrow \delta^*(q, w) \notin$ Final State

for all $w \in \Sigma^*$

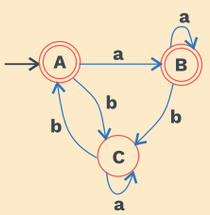
iv) On the other hand, if there exists some string $w \in \Sigma^*$ such that $\delta^*(p, w) \in$ Final State and $\delta^*(q, w) \notin$ Final State or vice-versa, then the states p and q are said to be distinguishable by a string 'w'.

v) Identify distinguishable states and use the logic known as seperating the states. It meSol: in each step if the two states in a set are distinguishable seperate them.

vi) Stop the algorithm once find no change in partition.

SOLVED EXAMPLES

Q1 **Minimize the following DFA:**



Sol:

δ	a	b
→ A	B	C
B	B	C
C	C	A

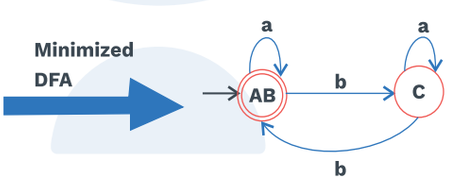
State equivalence method:

Follow steps i) and step ii), separate all final states in one set and all non final states in another set.

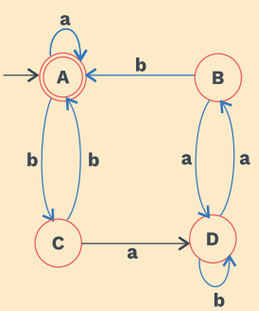
$\pi_0 = \{(A,B),(C)\}$ [0 equivalence class]

Let group 1 i.e. g_1 = (A, B) and
group 2 i.e. g_2 = (C)
Then, A on 'a' goes to the group g_1
And B on 'a' goes to the group g_1.
δ (A, a) = B (belongs to g_1)
δ (B, a) = B (belongs to g_1)
A on 'b' goes to the group g_2
B on 'b' goes to the group g_2
δ (A, b) = C (belongs to g_2)
δ (B, b) = C (belongs to g_2)
So, A and B will be in same group(set).
$\pi_1 = \pi_0$ [1 equivalence class]

δ	a	b
→ (AB)	AB	C
C	C	AB



Q2 **Minimize the following DFA:**



Sol:

δ	a	b
→ A	A	C
B	D	A
C	D	A
D	B	D

State equivalence method:

For State B and C:

$\pi_0 = \{(A), (B, C, D)\}$

δ	a	b
B	g_3	g_1
C	g_3	g_1

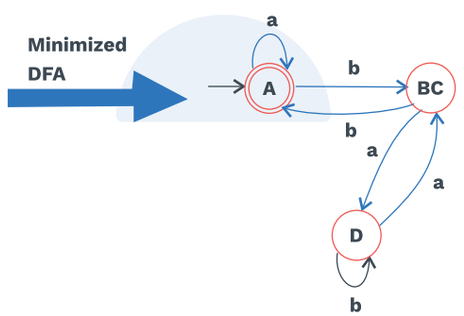
So, $\boxed{B = C}$

$\pi_1 = \begin{Bmatrix} (A), (B, C), (D) \\ g_1 \quad g_2 \quad g_3 \end{Bmatrix}$

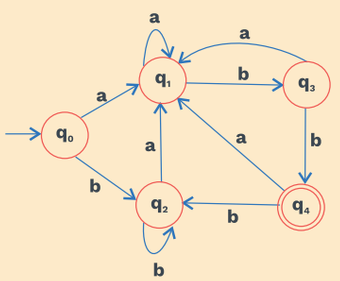
$\pi_2 = \pi_1$

δ	a	b
→ A	A	BC
BC	D	A
D	BC	D

Minimized DFA



Q3 **Minimize the following DFA:**



Sol:

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

By state equivalence method:

$$\pi_0 = \{(q_0, q_1, q_2, q_3), (q_4)\}$$
$$(g_1) \qquad (g_2)$$

For state q_0, q_1, q_2 and q_3

δ	a	b
q_0	g_1	g_1
q_1	g_1	g_1
q_2	g_1	g_1
q_3	g_1	g_2

$$\pi_1 = \{(q_0, q_1, q_2), (q_3), (q_4)\}$$
$$(g_1) \qquad (g_2) \quad (g_3)$$

For State q_0 , q_1 and q_2

δ	a	b
q_0	g_1	g_1
q_1	g_1	g_2
q_2	g_1	g_1

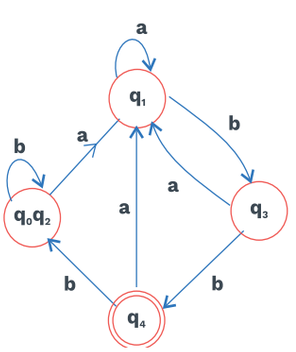
$$\pi_2 = \{(q_0, q_2), (q_1), (q_3), (q_4)\}$$

So, $q_0 = q_2$

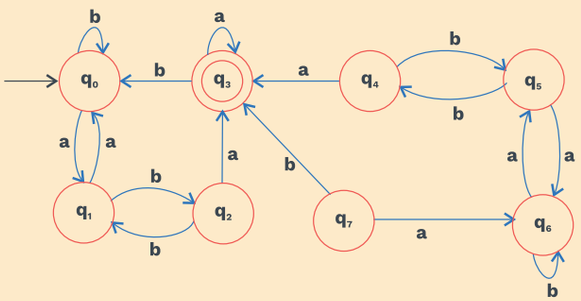
$\pi_3 = \pi_2$

δ	a	b
→ q_0q_2	q_1	q_0q_2
q_1	q_1	q_3
q_3	q_1	q_4
(q_4)	q_1	q_0q_2

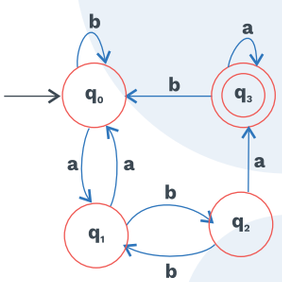
Minimized DFA



Q4 **Minimize the following DFA:**



Sol: Remove states q_4, q_5, q_6, q_7 as these are unreachable states.



δ	a	b
→ q_0	q_1	q_0
q_1	q_0	q_2
q_2	q_3	q_1
(*q_3)	q_3	q_0

State equivalence method:

$$\Pi_0 = \begin{Bmatrix} (q_0, q_1, q_2), (q_3) \\ g_1 \qquad g_2 \end{Bmatrix}$$

For state q_0, q_1 and q_2

δ	a	b
q_0	g_1	g_1
q_1	g_1	g_1
q_2	g_2	g_1

$$\Pi_1 = \left\{ \begin{matrix} (q_0, q_1), (q_2), (q_3) \\ g_1 \quad g_2 \quad g_3 \end{matrix} \right\}$$

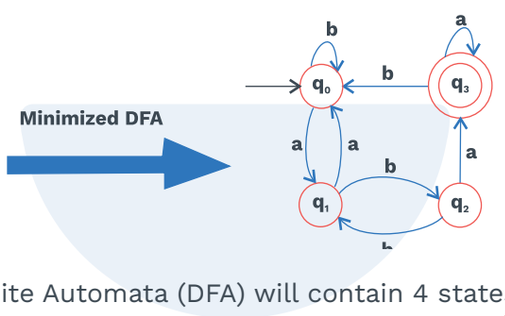
For state q_0 and q_1

δ	a	b
q_0	g_1	g_1
q_1	g_1	g_2

$$\Pi_2 = \{(q_0), (q_1), (q_2), (q_3)\} \Rightarrow \boxed{q_0 \neq q_1}$$

$$\Pi_3 = \Pi_2$$

δ	a	b
$\rightarrow q_0$	q_1	q_0
q_1	q_0	q_2
q_2	q_3	q_1
(q_3)	q_3	q_0



Minimized Deterministic Finite Automata (DFA) will contain 4 states.

Rack Your Brain

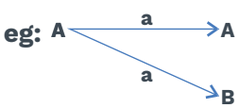
Given a regular language L, will its minimal DFA unique?

Non-Deterministic finite automata (NFA)

- In NFA, there can be 0 or 1 or more than one transition for any input symbol from any state.
- A NFA is defined as:

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

where δ: Transition function



i.e. state A on seeing input a, goes to more than one state which is A and B.
Let $Q = \{A, B\}$ and $\Sigma = \{a, b\}$

$$Q \times \Sigma \rightarrow 2^Q$$

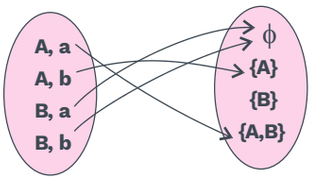


Fig. 2.4

Note:

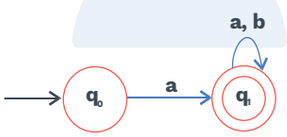
- In NFA, the transitions are not uniquely determined by their input symbol.
- In NFA, for any input symbol on a state can have many possible next states.
- NFA is easier to construct than DFA.

SOLVED EXAMPLES

Q1 **Construct NFA that accepts all strings of a's and b's where each string starts with 'a'.**

Sol: L = {a, aa, ab, ...}

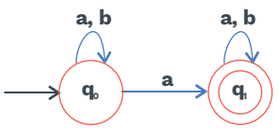
NFA:



Q2 **Construct NFA that accepts all strings of a's and b's where each string contains 'a'.**

Sol: L = {a, aa, ba, ab, ...}

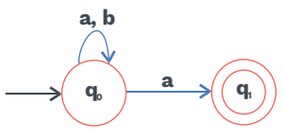
NFA:



Q3 **Construct NFA that accepts all strings of a's and b's where each string ending with 'a'.**

Sol: L = {a, aa, ba, aba, ...}

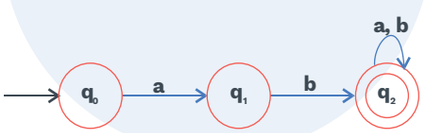
NFA:



Q4 **Construct NFA that accepts all strings of a's and b's where each string starts with 'ab'.**

Sol: L = {ab, aba, abb, ...}

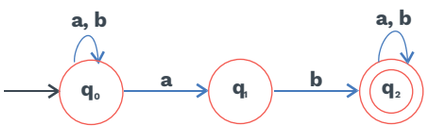
NFA:



Q5 **Construct NFA that accepts all strings of a's and b's where each string contains 'ab' as a substring.**

Sol: L={ab, aab, bab , aba ,.....}

NFA:



Q6 **Construct NFA that accepts all strings of a's and b's where each string starts with aabb.**

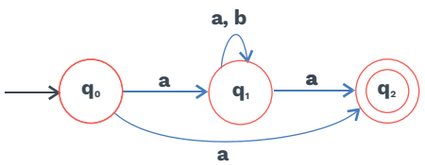
Sol: L = {aabb, aabba, aabbb, ...}

NFA:

Q7 **Construct NFA that accepts all strings of a's and b's where each string "starts and ends with a".**

Sol: L = {a, aa, aba, ...}

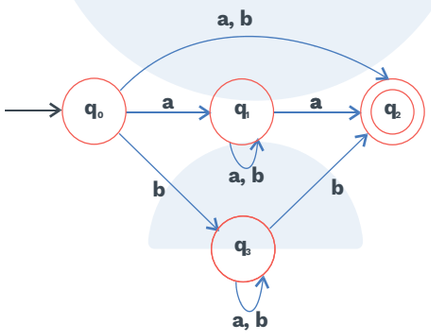
NFA:



Q8 **Construct NFA that accept all strings of a's and b's where each string "starting and ending with same symbol".**

Sol: L = {a, b, aa, bb, aba, bab, ...}

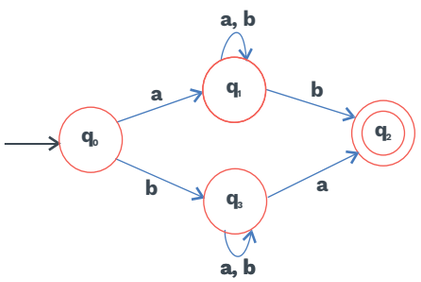
NFA:



Q9 **Construct NFA that accept all strings of a's and b's where each string "starting and ending with different symbol".**

Sol: L = {ab, ba, aab, abb, ...}

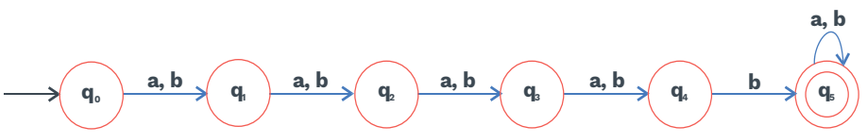
NFA:



Q10 **Construct NFA that accept all strings of a's and b's where 5th symbol is 'b' while reading the string from left to right.**

Sol: L = {aaaab, baaab, ...}

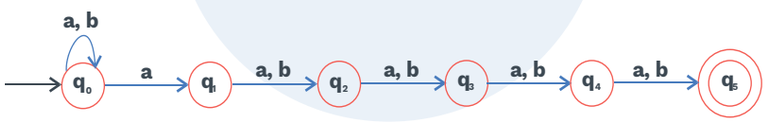
NFA:



Q11 **Construct NFA that accept all strings of a's and b's where 5th symbol is 'a' while reading string from right to left.**

Sol: L = {aaaaa, abbbb, ababb, baabab,.....}

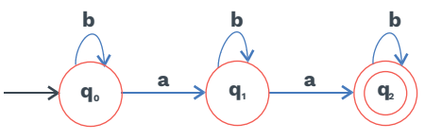
NFA:



Q12 **Construct NFA that accept all strings of a's and b's where each string contains exactly two a's.**

Sol: L = {aa, baa, aba, ...}

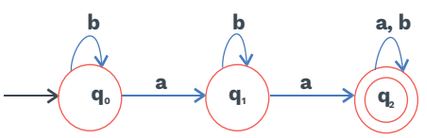
NFA:



Q13 **Construct NFA that accept all strings of a's and b's where each string contains atleast two a's.**

Sol: L = {aa, aaa, aab, baa, baaa,.....}

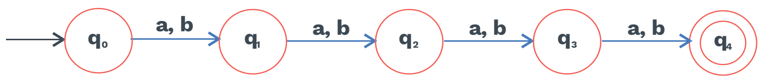
NFA:



Q14 **Construct NFA that accept all strings of a's and b's where the length of string is exactly 4.**

Sol: $L = \begin{Bmatrix} \text{aaaa, aabb, aaba, abaa, ...} \\ \text{baaa, bbbb, ...} \end{Bmatrix}$

NFA:

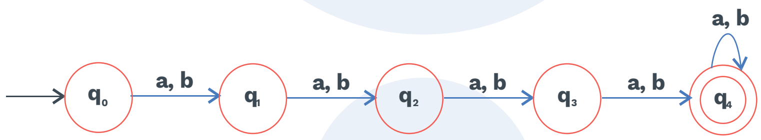


Q15 **Construct NFA that accept all strings of a's and b's where the length of string is atleast 4.**

Sol: Length of string is atleast 4 meSol: $|w| \geq 4$.

$L = \begin{Bmatrix} \text{aaaa, aaaab, ...} \\ \text{baaa, bbbbb, ...} \end{Bmatrix}$

NFA:

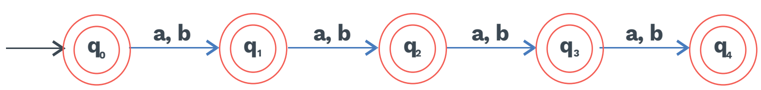


Q16 **Construct NFA that accept all strings of a's and b's where the length of string is atmost 4.**

Sol: Length of string is atmost 4 meSol: $|w| \leq 4$.

L = {∊, a, b, aa, ab, ba, bb,.......................}

NFA:



Note:

If the given NFA accept set of all the strings over $\Sigma = \{a, b\}$, where

Case I: Length of the string is exactly n

Case II: Length of the string is atleast n

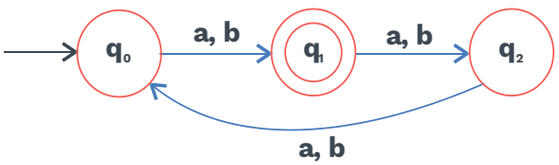
Case III: Length of the string is atmost n

In all these cases, minimum number of states in NFA = (n+1) states

Q17 **Construct NFA that accept all strings of a's and b's such that |w| = 1 mod 3.**

Sol: L = {a, b, aaaa, abab, ...}

NFA:



Rack Your Brain

Draw NFA with 3 states that accepts the language L = {a" In 21}{ba Imzo,k 20}

Previous Years' Question

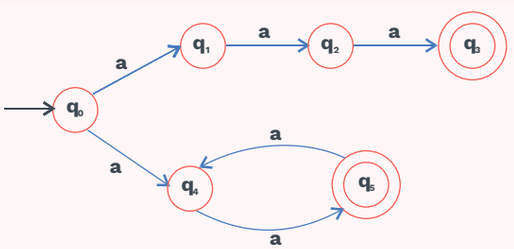
Let N be an NFA with n states. Let K be the number of states of minimal DFA which is equivalent to N. Which of the following is necessarily True? **(GATE 2018)**

1) $K \geq 2^n$ **2)** $K \geq n$

3) $K \leq n^2$ **4)** $K \leq 2^n$

Sol: 4)

Rack Your Brain



Let L be the language accepted by the above NFA. Construct an NFA that accepts LU {a").

2.2 COMPARISON OF DFA AND NFA

	Deterministic Finite Automata	Non-Deterministic Finite Automata
1) Definition	DFA is a type of finite Automata where for any input on a state, only one path is possible.	NFA is a type of Finite Automata where for any input on a state, many path can be possible.
2) Next State	In DFA, next possible state is unique.	In NFA, for any state and input pair, many next state can be possible.
3) Minimization	Minimization of DFA is possible	Minimization alogrithm is not applicable for NFA.
4) Construction	DFA is complex to construct and every DFA is a NFA.	NFA is easy to construct than DFA.

Table 2.1

Conversion from NFA to DFA:

Procedure:

Let $N = (Q, \Sigma, \delta, q_0, F) \rightarrow$ NFA

$M = (Q', \Sigma', \delta', q_0', F') \rightarrow$ DFA

i) There is no change in the initial state i.e. $\boxed{q_0' = q_0}$

ii) Start the construction of δ' with initial state and continue it for every new state which will come under input column.
Terminate this process when no more new state appears in the input column.

iii) Every set which contains final state of NFA as subset, will be the final state for corresponding DFA.

Note:

DFA obtained from NFA need not be a minimal DFA.

Q1 **Construct an equivalent DFA for the NFA that accepts all strings of a's and b's where each strnig "starts with a".**

Sol: $L_1 = \{a, aa, ab, ...\}$

NFA:

Transition table for NFA:

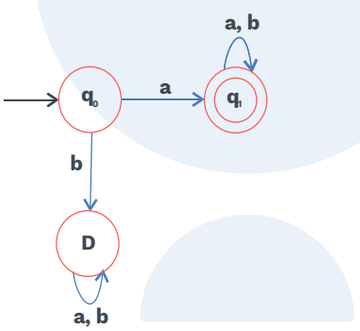
	a	b
→ q_0	q_1	ϕ
(*) q_1	q_1	q_1

Corresponding transition Table for DFA obtained from above table:

	a	b
→ q_0	q_1	D
(*) q_1	q_1	q_1
D	D	D

where D = Dead state (Introduced in DFA)

DFA:



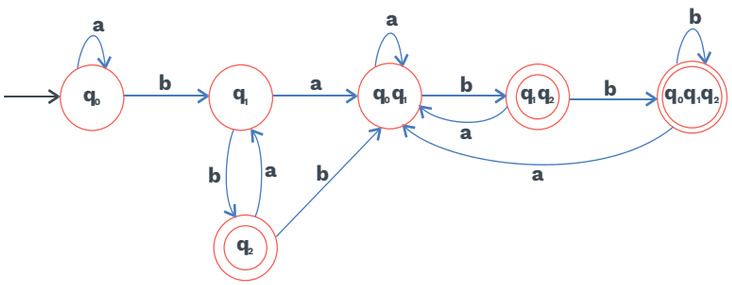
Q2 **Construct an equivalent DFA for the following NFA:**

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

Sol:

δ	a	b
→ q_0	$\{q_0\}$	$\{q_1\}$
q_1	$\{q_0q_1\}$	$\{q_2\}$
q_0q_1	$\{q_0q_1\}$	$\{q_1q_2\}$
(*) q_2	$\{q_1\}$	$\{q_0q_1\}$
(*) q_1q_2	$\{q_0q_1\}$	$\{q_0q_1q_2\}$
(*) $q_0q_1q_2$	$\{q_0q_1\}$	$\{q_0q_1q_2\}$

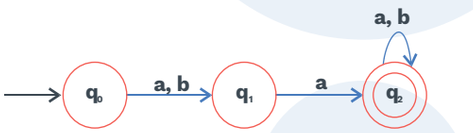
DFA diagram:



Q3 **Construct an equivalent DFA for the NFA that accepts all strings of a's and b's where 2nd symbol is 'a' while reading the string from left to right.**

Sol: $L = \{aa, ba, bab, aab, ...\}$

NFA:



Transition table for NFA:

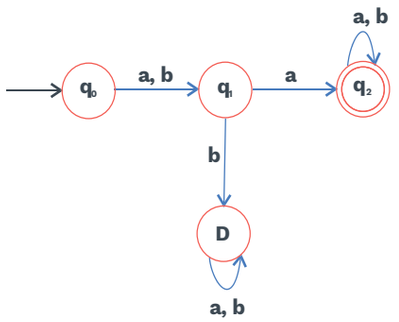
δ	a	b
→ q_0	q_1	q_1
q_1	q_2	ϕ
(*q_2)	q_2	q_2

Corresponding transition Table for DFA obtained from above NFA table:

δ	a	b
→ q_0	q_1	q_1
q_1	q_2	D
(*q_2)	q_2	q_2
D	D	D

Here 'D' is dead state.

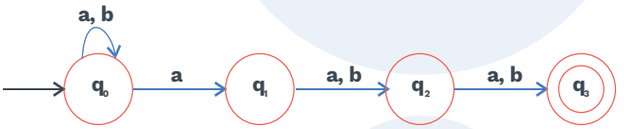
DFA diagram:



Q4 **Construct an equivalent DFA for the NFA that accepts all strings of a's and b's where 3rd symbol is 'a' while reading the string from right to left.**

Sol: $L = \{aaaa, aaab, baab, babb, ...\}$

NFA:

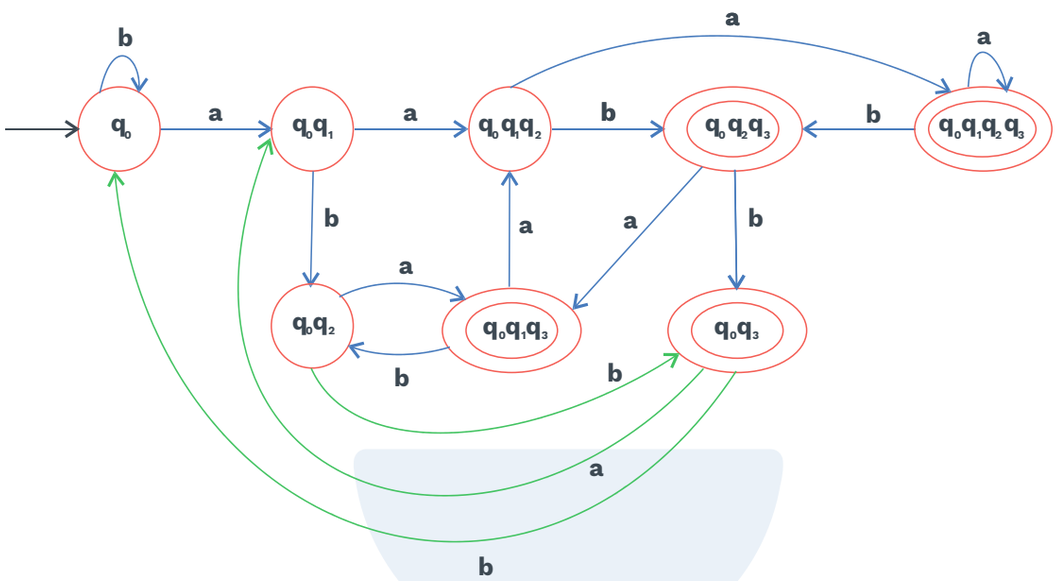


Transition table:

δ	a	b
→ q_0	$\{q_0, q_1\}$	$\{q_0\}$
q_1	$\{q_2\}$	$\{q_2\}$
q_2	$\{q_3\}$	$\{q_3\}$
(*) q_3	ϕ	ϕ

Corresponding transition table for DFA obtained from above table:

δ	**a**	**b**
→ q_0	$\{q_0q_1\}$	$\{q_0\}$
q_0q_1	$\{q_0q_1q_2\}$	$\{q_0q_2\}$
$q_0q_1q_2$	$\{q_0q_1q_2q_3\}$	$\{q_0q_2q_3\}$
q_0q_2	$\{q_0q_1q_3\}$	$\{q_0q_3\}$
(*) $q_0q_1q_2q_3$	$\{q_0q_1q_2q_3\}$	$\{q_0q_2q_3\}$
(*) $q_0q_2q_3$	$\{q_0q_1q_3\}$	$\{q_0q_3\}$
(*) $q_0q_1q_3$	$\{q_0q_1q_2\}$	$\{q_0q_2\}$
(*) q_0q_3	$\{q_0q_1\}$	$\{q_0\}$



Q5 **Construct the minimal DFA that accept all the strings of 0's and 1's where each string do not contain "101" as a substring.**

Sol: Since constructing NFA is easy, first Construct NFA where each string contain "101" as a substring

NFA:

	0	1
→ q_0	$\{q_0\}$	$\{q_0q_1\}$
q_1	$\{q_2\}$	ϕ
q_2	ϕ	$\{q_3\}$
(*) q_3	$\{q_3\}$	$\{q_3\}$

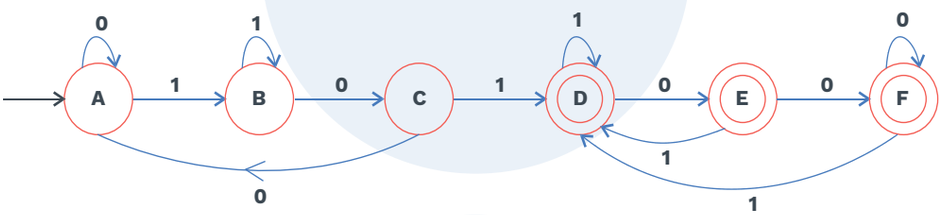
Conversion from NFA to DFA:

	0	1
→ q_0	$\{q_0\}$	$\{q_0q_1\}$
q_0q_1	$\{q_0q_2\}$	$\{q_0q_1\}$
q_0q_2	$\{q_0\}$	$\{q_0q_1q_3\}$
$q_0q_1q_3$	$\{q_0q_2q_3\}$	$\{q_0q_1q_3\}$
$q_0q_2q_3$	$\{q_0q_3\}$	$\{q_0q_1q_3\}$
q_0q_3	$\{q_0q_3\}$	$\{q_0q_1q_3\}$

Can replace states as

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

DFA:



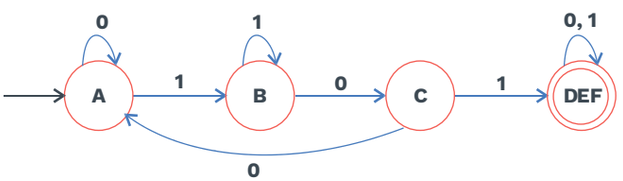
Minimal DFA:

$$\Pi_0 = \{(A,B,C),(D,E,F)\}$$

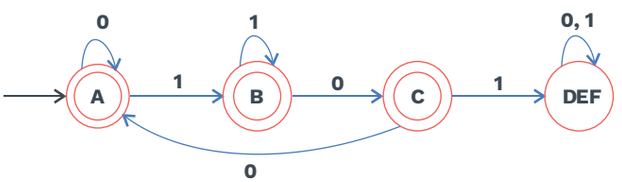
$$\Pi_1 = \{(A,B),(C),(D,E,F)\}$$

$$\Pi_2 = \{(A),(B),(C),(D,E,F)\}$$

$$\Pi_3 = \Pi_2$$



Now, complement the above DFA to get the minimal DFA that accepts all the strings over $\Sigma = \{0,1\}$ where each string "do not contain "101" as a substring."



2.3 EPSILON NON-DETERMINISTIC FINTE AUTOMATA (∈-NFA)

Specification of epsilon NFA:

- Epsilon- NFA $(\in -\text{NFA})$ is the NFA which contains $\in$ - moves/Null moves.
- This automation allows $\in$ as possible transition.
- $\in$-NFA is defined as 5-tuple:

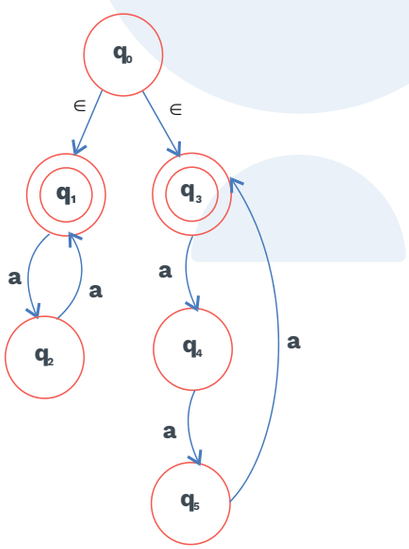
$(Q, \Sigma, \delta, q_0, F)$

Where transition function:

δ: $\boxed{Q \times \Sigma \cup \{\in\} \to 2^Q}$

Q: finite set of states
Σ: finite set of input symbols
q_0: Initial state
F: set of final states

Example: $\{a^n \mid n \text{ is even or divisible by 3}\}$

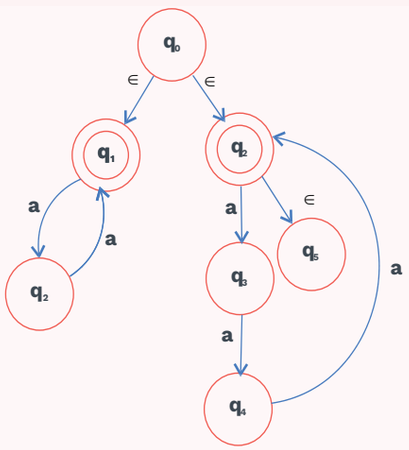


∈-closure

Definitions

The $\in$-closure of the state q, is the set that contains q along with all states which can be reached by only getting any number of $\in$ from state q.

Rack Your Brain



∈–closure (q_0) = ?
∈–closure (q_2) = ?

Conversion from ∈-NFA to NFA

- Why need ∈–NFA, when already NFA is present?

NFA: Easier to construct
∈–NFA: Even more easier to construct.

Procedure:

Let $N_1 = (Q', \Sigma', \delta', q_0', F')$ is the ∈–NFA.
$N_2 = (Q, \Sigma, \delta, q_0, F)$ is the equivalent NFA.

i) Initial state:

$$q_0 = q_0'$$

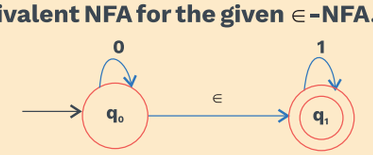
ii) δ Construction:

$$\forall_{i \in \Sigma}\ \forall_{q \in Q}\ \delta(q, i) = \in\text{-closure}\ [\delta'\ (\in\text{-closure}(q), i)]$$

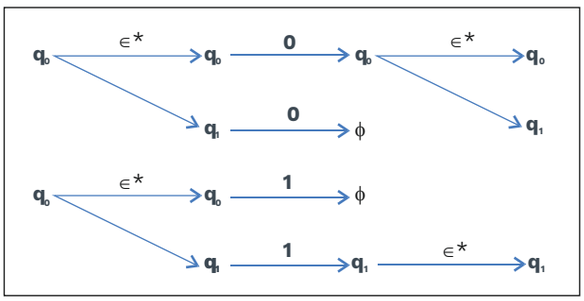
iii) Final state: Find ∈-closure of each state, if it contains any of the final state of N_1, then those state(s) will be the final state(s) in equivalent NFA N_2.

SOLVED EXAMPLES

Q1 **Construct an equivalent NFA for the given ∈-NFA.**



Sol: $\in$–closure $(q_0) = \{q_0, q_1\}$
$\in$–closure $(q_1) = \{q_1\}$

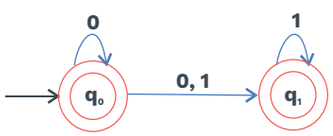


$\in$-NFA to NFA

Transition table:

δ	0	1
$\rightarrow q_0$	$\{q_0 q_1\}$	$\{q_1\}$
q_1	ϕ	$\{q_1\}$

NFA:

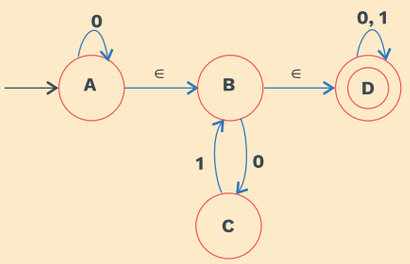


Final states in the NFA: Find the $\in$-closure of each state, if it contains any of the final state, then include this state as part of final states of equivalent NFA.
That's why here q_0 and q_1 both are final states.

Note:

In the conversion of $\in$-NFA to NFA, number of final states may increase.

Q2 **Construct an equivalent NFA for the given ∈-NFA.**



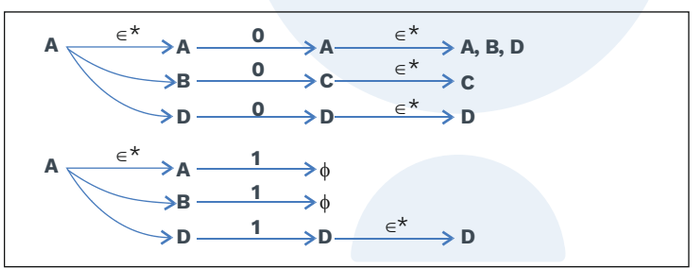
Sol:

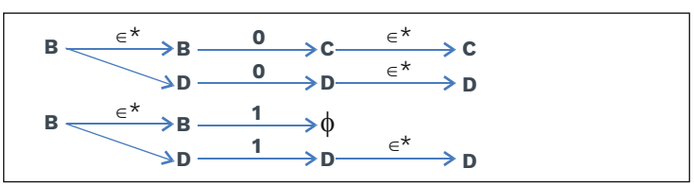
∈-closure A) = {A, B, D}

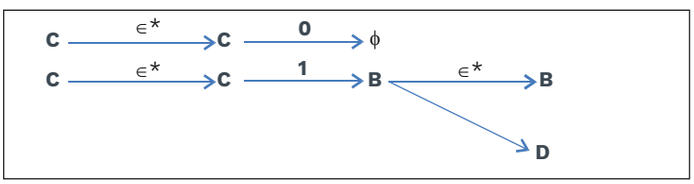
∈-closure B) = {B, D}

∈-closure C) = {C}

∈-closure D) = {D}





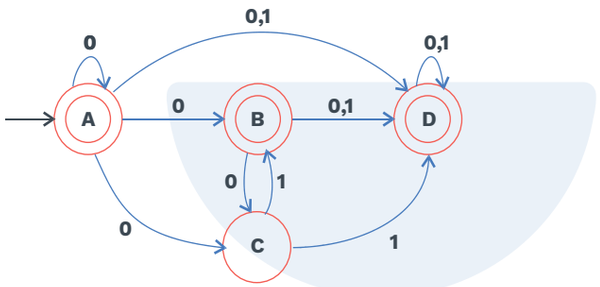


∈-NFA to NFA:

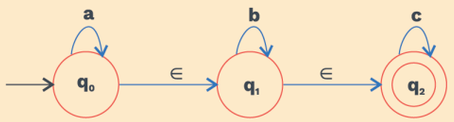
Transition table:

δ	0	1
→ (A)	{A, B, C, D}	{D}
(B)	{C, D}	{D}
C	ϕ	{B,D}
(D)	{D}	{D}

NFA:



Q3 **Construct an equivalent NFA for the given ∈-NFA.**

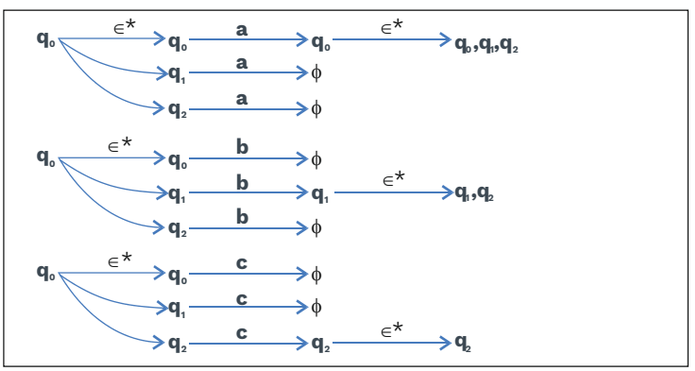


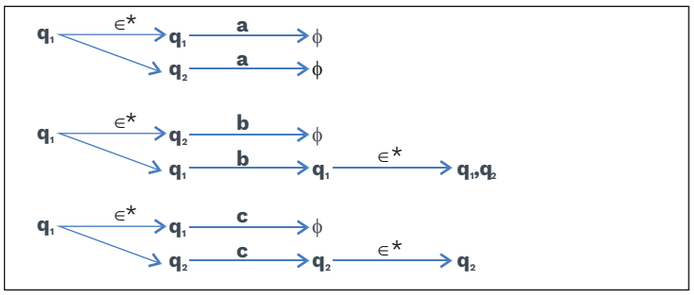
Sol:

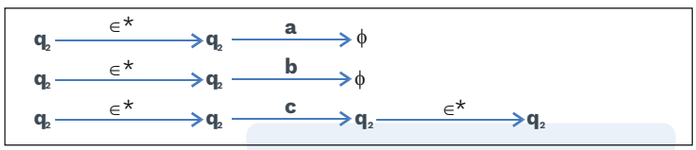
∈-closure (q_0) = {q_0, q_1, q_2}

∈-closure (q_1) = {q_1, q_2}

∈-closure (q_2) = {q_2}





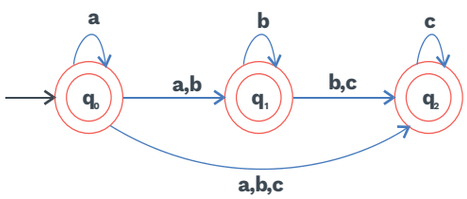


∈-NFA to NFA

Transition table:

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

NFA:



Rack Your Brain

Is the language preserved in all the steps while eliminating e-transition from a NFA?

Note:

Conversion from ∈–NFA to NFA shows that ∈–NFA and NFA are equivalent in power.

Previous Years' Question

Let δ denote the transition function and $\hat{\delta}$ denotes the extended transition function of the $\in$-NFA whose transition table is given below: **(GATE-2017 Set-2)**

δ	$\in$	a	b
$\rightarrow q_0$	$\{q_2\}$	$\{q_1\}$	$\{q_0\}$
q_1	$\{q_2\}$	$\{q_2\}$	$\{q_2\}$
q_2	$\{q_0\}$	ϕ	ϕ
q_3	ϕ	ϕ	$\{q_2\}$

Then $\hat{\delta}(q_2, aba)$ is

1) ϕ **2)** $\{q_0, q_1, q_3\}$

3) $\{q_0, q_1, q_2\}$ **4)** $\{q_0, q_2, q_3\}$

Sol: 3)

Conversion from epislon-NFA to DFA:

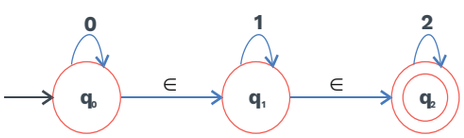
Let $N' = (Q', \sum', \delta', q_0', F')$ is the $\in$-NFA.

$M = (Q, \sum, \delta, q_0, F)$ is the equivalent DFA

i) Initial State:

$q_0 = \in$–closure$\left(q_0'\right)$

eg:



$\in$–closure $(q_0) = \{q_0, q_1, q_2\}$

ii) δ Construction:

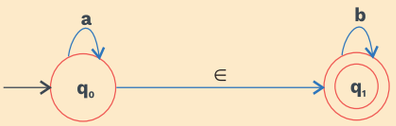
- Start Constructing δ with initial state and continue for every new state that appears under the input column.
- Terminate this process when no new state appears.

iii) Final state:

All states in equivalent DFA which contain final state of $\in$-NFA as a subset, is going to be the final state of DFA.

SOLVED EXAMPLES

Q1 **Construct an equivalent DFA for given $\in$-NFA.**



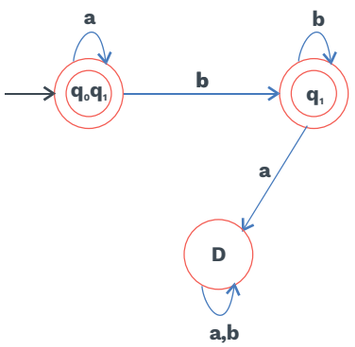
Sol: $\in$ – Closure $(q_0) = \{q_0, q_1\}$ = Initial state of DFA

Transition Table for DFA:

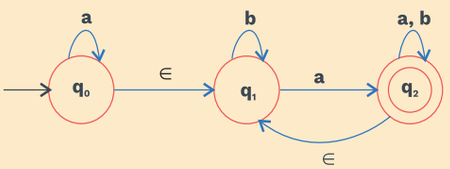
δ	a	b
$\rightarrow q_0q_1$	$\{q_0q_1\}$	$\{q_1\}$
q_1	{D}	$\{q_1\}$
D	{D}	{D}

Every state which contains q_1 will be the final state in DFA.

DFA:



Q2 **Construct an equivalent DFA for given ∈-NFA.**



Sol: ∈–closure (q_0) = {q_0, q_1}

We can construct the transition table for DFA as:

δ	a	b
→ q_0q_1	{$q_0q_1q_2$}	{q_1}
$q_0q_1q_2$	{$q_0q_1q_2$}	{q_1q_2}
q_1	{q_1q_2}	{q_1}
q_1q_2	{q_1q_2}	{q_1q_2}

Initial state

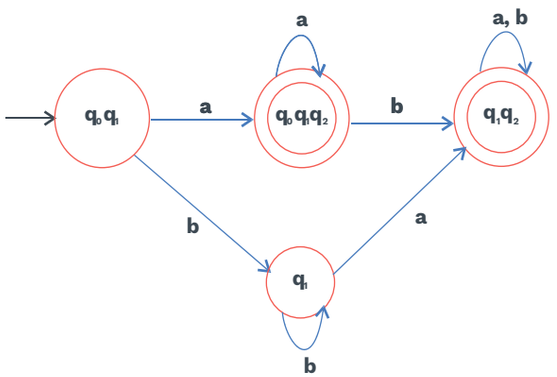
$q_0q_1 \xrightarrow{a} q_0,q_1, q_2$
$q_0q_1 \xrightarrow{b} q_1$

$q_0q_1q_2 \xrightarrow{a} q_0,q_1, q_2$
$q_0q_1q_2 \xrightarrow{b} q_1,q_2$

$q_1 \xrightarrow{a} q_1,q_2$
$q_1 \xrightarrow{b} q_1$

$q_1q_2 \xrightarrow{a} q_1,q_2$
$q_1q_2 \xrightarrow{b} q_1,q_2$

DFA:



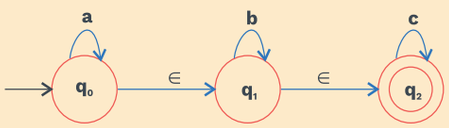
Here, $q_0q_1q_2$ and q_1q_2 are final states in DFA as it contains q_2 which is final states for given $\in$-NFA.

Note:

In the conversion from $\in$ –NFA to DFA:

- There might be change in initial state.
- There might be change in final state.
- There might be change in total number of states.

Q3 **Construct an equivalent DFA for given $\in$-NFA.**



Sol: $\in$–closure $(q_0) = \{q_0, q_1, q_2\}$

We can construct the transition table for DFA as:

δ	**a**	**b**	**c**
$\rightarrow q_0q_1q_2$	$\{q_0q_1q_2\}$	$\{q_1q_2\}$	$\{q_2\}$
q_1q_2	$\{D\}$	$\{q_1q_2\}$	$\{q_2\}$
q_2	$\{D\}$	$\{D\}$	$\{q_2\}$
D	$\{D\}$	$\{D\}$	$\{D\}$

Initial state

$q_0q_1q_2 \xrightarrow{a} q_0q_1q_2$

$q_0q_1q_2 \xrightarrow{b} q_1q_2$

$q_0q_1q_2 \xrightarrow{c} q_2$

$q_1q_2 \xrightarrow{a}$ Dead State 'D'

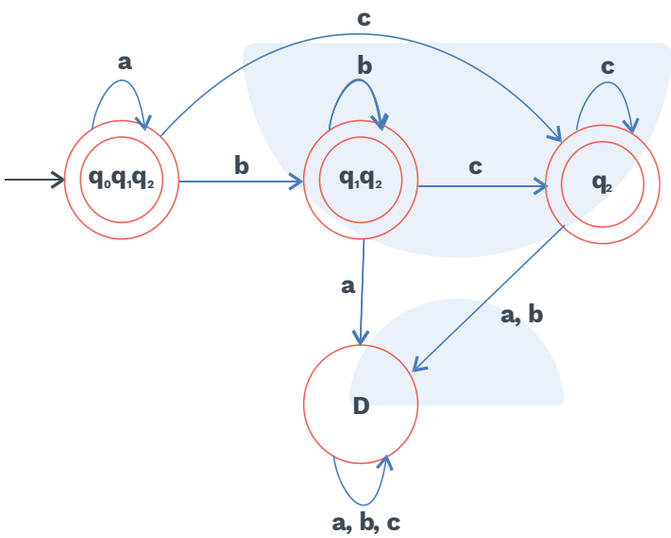
$q_1q_2 \xrightarrow{b} q_1,q_2$

$q_1q_2 \xrightarrow{c} q_2$

$q_2 \xrightarrow{a}$ Dead State 'D'
$q_2 \xrightarrow{b}$ Dead State 'D'
$q_2 \xrightarrow{c} q_2$

D $\xrightarrow{a}$ Dead State 'D'
D $\xrightarrow{b}$ Dead State 'D'
D $\xrightarrow{c}$ Dead State 'D'

DFA:

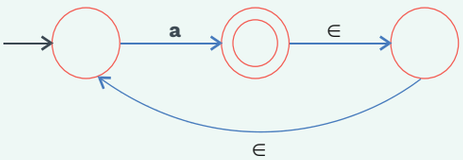


Note:

The DFA obtained from ∈ –NFA need not be minimal.

Previous Years' Question

What is the complement of the language accepted by the NFA shown below? Assume $\Sigma = \{a\}$ and $\in$ is the empty string. **(GATE-2012)**



1) ϕ **2)** $\{\in\}$

3) a^* **4)** $\{a, \in\}$

Sol: 2)

Note:

- Conversion from $\in$–NFA to DFA shows that $\in$–NFA and DFA are equivalent in power.
- $\in$–NFA, NFA and DFA all are equivalent in power.

2.4 FINITE TRSol:DUCER

Definitions

Finite state trSol:ducer is a finite state automaton which produces output on reading any input.

- A finite state trSol:ducer (FST) can be thought of as trSol:lator or relator between strings in a set.
- It is very useful in parsing.

Moore machines

Definitions

Moore Machines are finite state machines with output value associated with states.

It can be defined as –

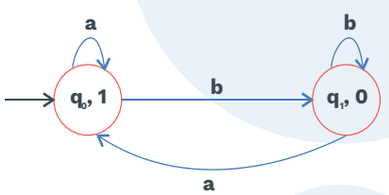
$(Q, \Sigma, \delta, q_0, \Delta, \lambda)$ where

Q: Finite set of states
Σ: Input alphabets
δ: Transition function

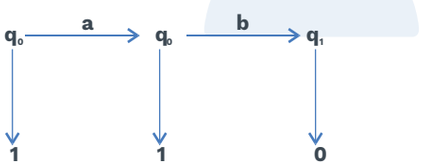
$$\boxed{Q \times \Sigma \rightarrow Q}$$

q_0 Initial state
Δ: output alphabets
λ: output function which maps $Q \rightarrow \Delta$

eg:



Input: ab



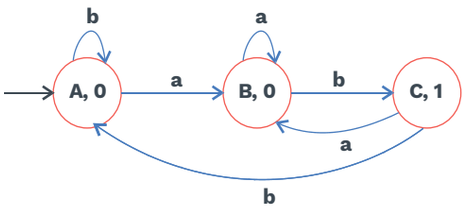
Note:
In Moore machines if input is of 'n bits' then output will be of 'n+1 bits'.

SOLVED EXAMPLES

Q1 **Construct a moore machine that takes set of all strings over {a,b} as input and prints '1' as output for every occurrence of 'ab' as a substring.**

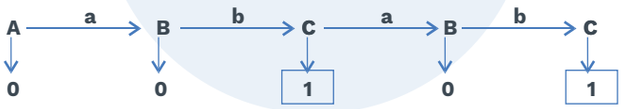
Sol: Whenever see 'ab' print '1' as output

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



Take example and verify it:
Input1: ab

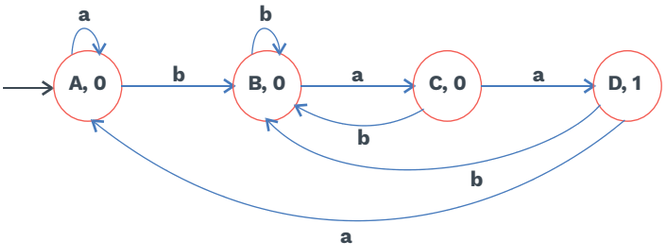
Input2: abab



Q2 **Construct a Moore machine that takes set of all strings over {a,b} and count number of occurrences of substring 'baa'.**

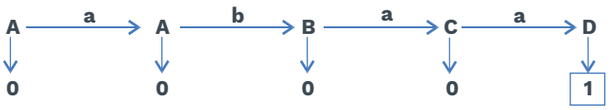
Sol: $\Sigma = \{a, b\}$

Take $\Delta = \{0, 1\}$



Take example and verify it

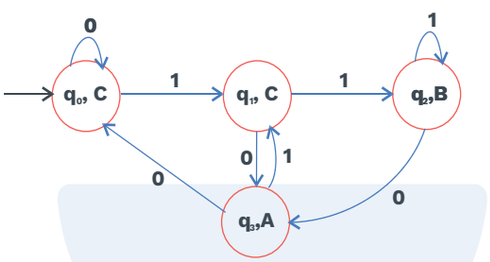
Input: abaa



Q3 **Construct a Moore machine that takes set of strings over Σ = {0, 1} and produces 'A' as output if input ends with '10' or produces 'B' as output if input ends with '11' otherwise produces 'C' as output.**

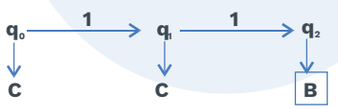
Sol: $\Sigma = \{0, 1\}$

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



Take example and verify it:

Input 1: 11



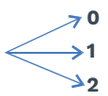
Input 2: 1 1 0

Q4 **Construct a moore machine that takes binary number as input and produces 'residue module 3' as output.**

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

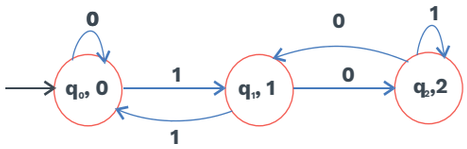
$\Delta = \{0, 1, 2\}$

Remainder: Binary number %3



δ	0	1	Δ
→ q_0	$\{q_0\}$	$\{q_1\}$	0
q_1	$\{q_2\}$	$\{q_0\}$	1
q_2	$\{q_1\}$	$\{q_2\}$	2

Represents states equal to the number of remainders when any binary number divisible by 3



Rack Your Brain

Construct a moore machine that takes base 4 numbers as input and produces 'reakdus modulo 6 output?

Mealy machine:

Definitions

Mealy machines are finite state machines, where output depends upon the present input symbol and present state of the machine.

- It can be defined as: $(Q, \Sigma, \delta, q_0, \Delta, \lambda)$, where

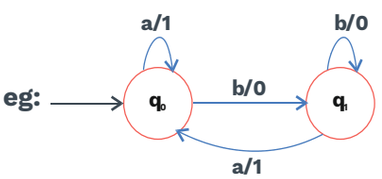
Q: Finite set of states
Σ: Input alphabets
δ: Transition function

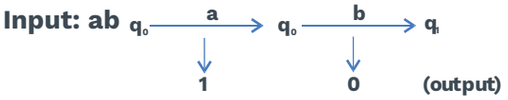
$$Q \times \Sigma \rightarrow Q$$

q_0: Initial state

Δ: Output alphabet

λ: Output function $Q \times \Sigma \rightarrow \Delta$





Note:
In Mealy machine, if input is of 'n bits' then output will be of 'n bits'.

SOLVED EXAMPLES

Q 1 **Construct a mealy machine that takes binary number as input and produces 2's complement of that number as output. Assume the string is read from LSB to MSB and end carry is discarded.**

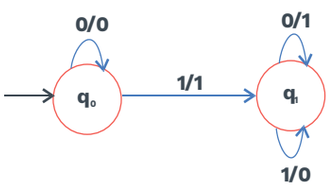
Sol: $\Sigma = \{0, 1\}$

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

To calculate: 2's complement of binary number 1011
1's complement of this number = 0100

2's complement of this number = $\begin{array}{r} 0\ 1\ 0\ 0 \\ +\ 1 \\ \hline 0\ 1\ 0\ 1 \end{array}$

Observation:
From LSB whenever you see 0, leave it as it is, whenever you see 1st 1, leave as it is. After this, whatever digits come, just complement it, will give 2's complement



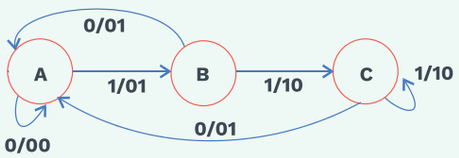
Rack Your Brain

Construct a mealy machine that takes binary umber as input and produces 1's complement of that number as output

Previous Years' Question

The finite state machine described by the following state diagram with A as starting state, where an arc label is x/y and x stands for 1-bit input and y stands for 2-bit output

(GATE-CS-2002)



1) Outputs the sum of the present and the previous bits of the input.
2) Outputs 01 whenever the input sequence contains 11
3) Outputs 00 whenever the input sequence contains 10
4) None of the above

Sol: 1)

Rack Your Brain

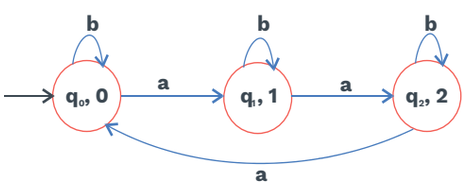
Construct a mealy machine that takes binary umber as input and produces output = $(\text{input})_2 \bmod 3$.

Conversion of Moore machine to Mealy machine

SOLVED EXAMPLES

Q1 **Convert a Moore machine to Mealy machine which count the number of a's divisible by 3.**

Sol: We have to construct a Moore machine which counts the number of a's is divisible by 3.

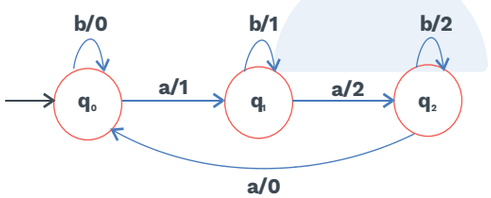


	a	b	Δ
→ q_0	$\{q_1\}$	$\{q_0\}$	{0}
q_1	$\{q_2\}$	$\{q_1\}$	{1}
q_2	$\{q_0\}$	$\{q_2\}$	{2}

When convert moore to mealy, then q_0 state on input a goes to q_1 states and output associated with state q_1 is 1 so, it will go onto the transition in equivalent mealy machine. Apply similar concepts for every state.

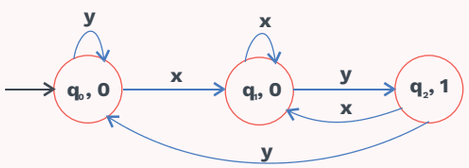
Transition: Diagram for mealy machine:

	a	b
→ q_0	$\{q_1,1\}$	$\{q_0,0\}$
q_1	$\{q_2,2\}$	$\{q_1,1\}$
q_2	$\{q_0,0\}$	$\{q_2,2\}$



Rack Your Brain

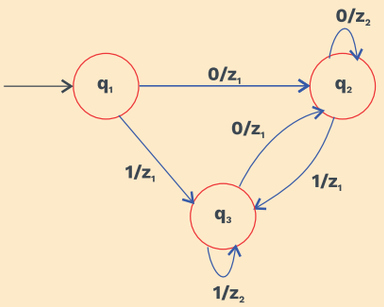
Convert the following given Moore machine to Mealy machine:



Conversion from mealy to moore machine:

SOLVED EXAMPLES

Q1 **Convert the following given Mealy machine to Moore machine:**

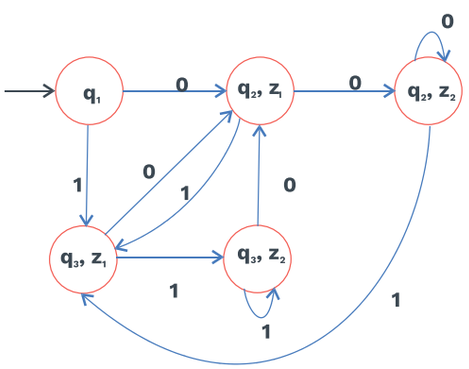


Sol:

i) State q_1 on getting input 0 goes to state q_2 and the output present in this transition gets associated with states q_2 in the corresponding equivalent moore machine.

ii) State q_1 on getting input 1 goes to state q_3 and the output present in this transition i.e. 'z_1' gets associated with state q_3 in the corresponding equivalent moore machine.

iii) We will follow this above procedure for other states as well as the new states we are getting.

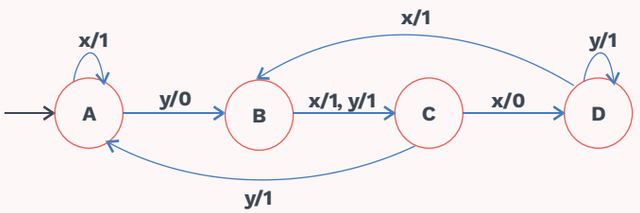


Note:

- In the conversion from Moore to Mealy machine, the number of states remain same.
- In the conversion from Mealy to Moore machine, the number of states may increase.

Rack Your Brain

Convert the following given Mealy machine to equivalent Moore machine.



Chapter Summary

- **Finite Automata:** Finite automata is a mathematical model which involves states and transitions among states in response to inputs.
- **Types of Finite Automata:** Finite automata without output and Finite automata with output.
- **Classification of Finite Automata without output:** DFA, NFA, $\in$–NFA.
- **Classification of Finite Automata with output:** Moore machine, Mealy machine.
- **DFA:** A DFA has a finite set of states and a finite set of input symbols. DFA is defined as quintuple (Q, $\sum$, δ, q_0, F) where $\delta : Q \times \sum \to Q$
- **Minimization of DFA:** TrSol:forming a given DFA into an equivalent DFA having the minimum number of states.
- **NFA:** A NFA is defined as Quintuple (Q, $\sum$, δ, q_0, F) where $\delta : Q \times \sum \to 2^Q$
- **Comparison of DFA and NFA:** Mainly the NFA differs from the DFA in that the NFA can have any number of transitions (including zero) to the next states from a given state on a given input symbol.
- **Conversion from NFA to DFA:** It is possible to convert any NFA to DFA that accepts the same language. All DFAs are NFA.
- DFA obtained from NFA need not be a minimal DFA.
- $\in$**-Transition:** It is possible to extend the NFA by allowing transitions on an empty input (i.e. no input symbol at all).
- $\in$**-NFA:** NFA in which transition is also defined for an empty input is known as $\in$-NFA.
- Conversion from $\in$-NFA to DFA as well as NFA is possible.
- **Finite State TrSol:ducer:** A finite state automaton which produces output on reading any input.
- **Moore Machines:** Finite state machine with output value associated with states. It is defined as (Q, $\sum$, δ, q_0, Δ, λ) where output function $\lambda : Q \to \Delta$.
- **Mealy Machines:** Finite state machines where output is associated with the transition.
  It is defined as (Q, $\sum$, δ, q_0, Δ, λ) where output function $\lambda : Q \times \sum \to \Delta$.

3 Regular Expression, Grammar and Language

3.1 REGULAR EXPRESSION

- Regular expression represents a regular language.
- For every regular expression, regular language can be generated and for every regular langauge, a regular expression is possible.
- Similar to arithmetic, where operations + and × are used to build up expressions such as (5 + 3) × 4;
  Regular operations are used to build up expressions which describe languages are known as a regular expression.
  Example: $(0+1)0^*$
- The notation involves a combination of strings of symbols from some alphabets $\sum$, parentheses and the operators +, . and *

Example: If L = {∈, aa, aaaa, aaaaaa, ...} then L can be expressed as (aa)*.

Application of regular expression:

a) In Unix grep or equivalent commands for finding strings.

b) In lexical analyser generators such as Lex or flex.

Introduction: Formal definition:

Definition

Let $\sum$ be a given alphabet, then

a) $\phi, \in, a \in \sum$ are all regular expressions.

These are called as primitive regular expressions.

b) If r_1 and r_2 are regular expressions, then $r_1 + r_2$, r_1r_2, r_1^* are also regular expressions.

c) A string is a regular expression if and only if it can be derived from the primitive regular expression by a finite number of applications of the rules mentioned in b).

Note:

Don't confuse between the Regular expression ∈ and ϕ.

- The regular expression ∈ represents the language containing a single string which is empty string.
- The regular expression ϕ represents the language that does not contain any string.

Operators of regular expression:

1) Union

- If R_1, R_2 are any two regular expressions, then their union $(R_1 + R_2)$ will also be a regular expression.

2) Concatenation

- If R_1, R_2 are any two regular expressions, then their concatenation (R_1. R_2) will also be a regular expression.

3) Kleene closure

- If R_1 is any regular expression, then Kleene closure of R_1 (R_1^*) will also be a regular expression.

There are three operations on languages that the operators of regular expressions represent. These operations are:

1) Union

- The union of two languages L_1 and L_2 denoted as $L_1 \cup L_2$.
- $L_1 \cup L_2$ means a set of strings are either in L_1 or L_2 or both.

Example: $L_1 = \{10, 111\}$
$L_2 = \{\in, 01\}$, then $L_1 \cup L_2 = \{\in, 01, 10, 111\}$

2) Concatenation

- The concatenation of two languages L_1 and L_2 is denoted as $L_1 . L_2$.
- $L_1 .L_2$ is the set of all strings that can be formed by taking any string from language L_1 and concatenating it with any string from language L_2.

Example: $L_1 = \{001, 10, 111\}$
$L_2 = \{\in\}$
$L_1 .L_2 = L_1 L_2 = \{001, 10, 111\}$

Note:

$\in$ is the identity which concatenating with any string resulting same string.

Example: String 100 concatenating with epsilon, $100.\in = 100$ over $\sum = \{0, 1\}$.

3) Closure (Star closure/Kleene closure)

- The closure of any language L is represented by L^*.
- Where L^* represents the set of all strings that can be formed by taking any number of strings from L, with repetitions and concatenating all of them.

$L = \{0, 1\}$
L^* = all strings of 0's and 1's over {0, 1}

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

$L^\circ = \{\in\}$
$L^1 = L$
$L^2 = L.L$
$L^i = L.L.L$..... i times

Language associated with regular expressions:

- Regular expressions are used to describe regular languages.

- The language L(r) denoted by any regular expression 'r' is defined as:
  - **i)** ϕ is regular expression denoting { } i.e empty set.
  - **ii)** $\in$ is a regular expression denoting $\{\in\}$.
  - **iii)** For every $a \in \Sigma$, a is a regular expression denoting {a}.
- If r_1 and r_2 are two regular expressions, then:
  - **i)** $L(r_1 + r_2) = L(r_1) \cup L(r_2)$
  - **ii)** $L(r_1.r_2) = L(r_1).L(r_2)$
  - **iii)** $L((r_1)) = L(r_1)$
    {If r_1 is a regular expression, then (r_1), a parenthesized r_1, is also a regular expression denoting the same language as r_1.}
  - **iv)** $L(r_1^*) = (L(r_1))^*$

These four rules are used to reduce L(r) to simpler components recursively.

Examples:

i) $r = a^*$
$L(r) = L(a^*) = \{\in, a, aa, aaa, ...\}$

ii) $r = a + b$
$L(r) = \{a, b\}$

iii) $r = ab$
$L(r) = \{ab\}$

iv) $r = (a + b)^*$
$L(r) = \{\in, a, b, aa, ab,.....\}$

v) $r = a^+ = a.a^*$
$L(r) = \{a, aa, ...\}$

vi) $r = a^* + ba$
$L(r) = \{\in, a, aa, aaa, ...,ba\}$

Rack Your Brain

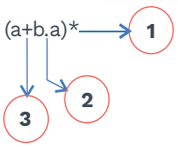
Find all strings in L((a+b)b(a+ab)*) of length less than four?

Order of precedence of regular-expression operators:

i) The star operator (*) is of the highest precedence.
ii) Next is concatenation or "dot" operator, which comes in precedence.
iii) Next is the union (+ operator) comes in precedence..

Order of precedence : * > · > +

Example: (a+b.a)* → 1, 2, 3



Note:

Since concatenation and union are associative operator. It does not matter in what order we group consecutive concatenations and consecutive unions but we shall assume grouping from left.

SOLVED EXAMPLES

Q1 **Which of the following is correct?**

1) $(r^*)^+ = r^+$ **2)** $(r^*)^* = r^*$

Sol:

1) **LHS:** $(r^*)^+ = \{\in, r, rr, ...\}$

RHS: $r^+ = \{r, rr, ...\}$

LHS ≠ RHS

2) **LHS:** $(r^*)^* = \{\in, r^*, r^*r^*, ...\}$

$= \{\in, r^*, r^*, ...\}$

$= \{\in, r, rr, ...\}$

RHS: $r^* = \{\in, r, rr, ...\}$

LHS = RHS

Option 2) is correct.

Q2 **Which of the following is incorrect?**

1) $r^* + r^+ = r^*$ **2)** $r^*.r^* = r^+$

Sol:

1) LHS: $r^* + r^+ = \{\in, r, rr, ...\} \cup \{r, rr, ...\}$

$= \{\in, r, rr, ...\}$

RHS: $r^* = \{\in, r, rr, ...\}$

LHS = RHS

2) LHS: $r^*. r^* = \{\in, r, rr, ...\}. \{\in, r, rr, ...\}$

$= \{\in, r, rr, ...\}$

RHS: $r^+ = \{r, rr, rrr, ...\}$

LHS ≠ RHS

Option 2) is incorrect.

Q3 **r^* is finite iff $r = \phi$ or $r = \in$**

1) True **2) False**

Sol: If $r = \phi$, then $r^* = \phi^* = \{\in\}$
If $r = \in$, then $r^* = \in^* = \{\in\}$
Option 1) True.

Q4 **Which of the following is correct?**

Let $r_1 = (a^*b)^*$ and $r_2 = (a + b^*)^*$

1) $L(r_1) = L(r_2)$

2) $L(r_1) \subset L(r_2)$

3) $L(r_1) \supset L(r_2)$

4) None of these

Sol: **Given:**
$r_1 = (a^*b)^*$ and $r_2 = (a+b^*)^*$
$L(r_1) = \{\in, b, ab, aab,.....\}$
$L(r_2) = \{\in, a, b, aa, ab, ba, bb, aab,.....\}$
So, $L(r_1) \subset L(r_2)$
Option 2)

Rack Your Brain

Identify which of the following are identical?

I) a* **II)** (aa)* **III)** (aa*)a **IV)** (a + ∈)a*

1) I and III only **2)** I and IV only **3)** I, III and IV only **4)** None of these

Q5 **Which of the following regular expression does not generate string containing "baa" as substring?**

1) $a^*(ba)^*$ **2) $a^*b^*(ba)^*a$** **3) $(ab^*+a)^*(ab)^*b^*a^*$** **4) $(bba^* +b)^*a$**

Sol:

Option 1) : $a^*(ba)^* = \{\in, a, ba, aba, baba, ...\}$
It cannot generate string containing 'baa' as a substring.

Option 2) : $a^*b^*(ba)^* a = \{a, aa, ba, baa, ...\}$
It can generate string containing 'baa' as a substring.

Option 3) : $(ab^*+ a)^* (ab)^* b^*a^* = \{\in, a, b, aa, bb, ab, ba, baa, abba,.....\}$
It can generate string containing 'baa' as a substring.

Option 4) : $(bba^*+ b)^* a = \{a, ba, bba, bbaa, ...\}$
It can also generate string containing 'baa' as a substring.

Previous Years' Question

Which of the following statements is **TRUE** about the regular expression 01*0?

(GATE IT-2005)

1) It represents a finite set of finite strings
2) It represents an infinite set of finite strings
3) It represents a finite set of infinite strings
4) It represents an infinite set of infinite strings

Sol: 2)

Equivalence between regular language and regular expression:

- For every regular language, there is a regular expression and for every regular expression, there is a regular language.

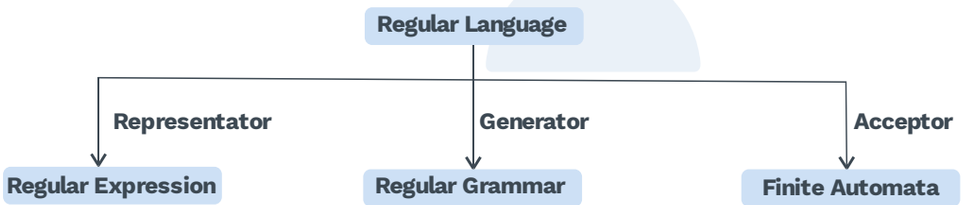


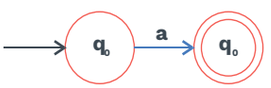
Fig. 3.1

Theorem

"Let r be a regular expression, then there exists some non-deterministic finite acceptor that accepts L(r). Consequently, L(r) is a regular language."

Example:

NFA accepts ϕ



NFA accepts {a}

Examples of equivalence between Regular language and regular expression:

SOLVED EXAMPLES

Q1 **Find the regular expression over $\sum$ = {a, b} for the language :**

i) contains all strings including $\in$

ii) contains all strings excluding $\in$

Sol:

i) L = {$\in$, a, b, aa, ab, ba, bb, ...}

$$\boxed{RE = (a+b)^*}$$

ii) L = {a, b, aa, ab, ba, bb, ...}

$$\boxed{RE = (a+b)^+}$$

Q2 **Regular expressions for the language over $\sum$ = {a, b} where:**

i) set of all strings start with a

ii) set of all strings that ends in 'b'

iii) set of all strings contains substring 'ab'

Sol:

i) L = {a, aa, ab, ...}

$$\boxed{RE = a(a+b)^*}$$

ii) L = {b, ab, bb, ...}

$$\boxed{RE = (a+b)^*b}$$

iii) L = {ab, aab, bab, ...}

$$\boxed{RE = (a+b)^*ab(a+b)^*}$$

Q3 **Find the regular expression for the language over $\sum$ = {a, b} where**

i) all the strings having 3rd symbol from left is 'b'

ii) all the strings having 3rd symbol from right is 'a'

Sol:

i) L = {bab, bbb, aab, abba, baba, ...}

$$\boxed{RE = (a+b)(a+b)b(a+b)^*}$$

ii) L = {aab, aba, abb, baba, ...}

$$\boxed{RE = (a+b)^*a(a+b)(a+b)}$$

Q4 **Find the regular expression for the language over $\sum$ = {a, b}**

i) where strings start and end with same symbol.

ii) where strings start and end with different symbol.

Sol:

i) L = {a, b, aa, aba, bb, bab, ...}

$$\boxed{RE = a(a+b)^*a + b(a+b)^*b + a + b}$$

ii) L = {ab, ba, aab, ...}

$$\boxed{RE = a(a+b)^*b + b(a+b)^*a}$$

Q5 **Find the regular expression for the language over $\sum$ = {a, b} where,**

i) set of all strings having length exactly 2

ii) set of all strings having length atleast 2

iii) set of all strings having length atmost 2

Sol:

i) L = {aa, ab, ba, bb}

RE = ab + aa + bb + ba = a(a+b)+b(a+b)

$$\boxed{RE = (a+b)(a+b)}$$

Similarly, set of all strings having length exactly 3 over $\sum$ = {a, b}

RE = (a + b) (a + b) (a + b)

ii) L = {aa, ab, ba, bb, aaa, aab,.....}

$$\boxed{RE = (a+b)(a+b)(a+b)^*}$$

iii) L = {$\in$, a, b, aa, ab, ba, bb}

RE = $\in$+ a + b + ab + aa + ba + bb

$$\boxed{RE = (a+b+\in)(a+b+\in)}$$

Q6 **Find the regular expression over $\sum$ = {a, b} for the language which accepts**

i) Even length string

ii) Odd length string

iii) All the strings divisible by 3

iv) All the strings where w $\cong$ 2 (mod 3)

v) All the strings where number of a's are exactly equal to 2.

vi) All the strings where number of a's are 'atleast 2'

vii) All the strings where number of a's are 'atmost 2'.

viii) All the strings where number of a's are even.

Sol:

i) Even length string

L = {∈, aa, ab, ba, bb, ...}

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

ii) Odd length string

L = {a, b, aaa, ...}

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

iii) all the strings divisible by 3:

L = {aaa, aab, aba, ...}

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

iv) all the strings where w ≅ 2 (mod 3)

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

v) all the strings the where the number of a's are exactly 2.

L={aa,aab,aba,baa.....}

$$RE = b^*ab^*ab^*$$

vi) all the strings where the number of a's are 'at least 2'.

L = {aa, aab, aba, aaa, ...}

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

vii) all the strings where the number of a's are 'atmost 2'.

L = {∈, a, ab, ba, aa, ...}

$$RE = b^*(\in +a)b^*(\in +a)b^*$$

viii) all the strings where the number of a's are even.

L = {∈, b, bb, aa, aab, aba .....}

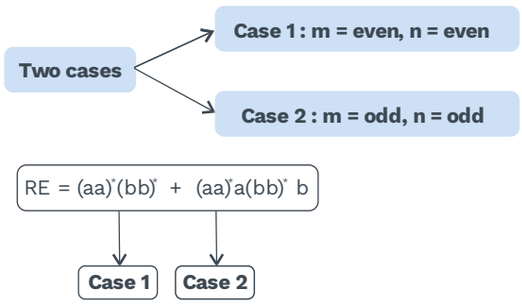
$$RE = (b^* ab^* ab^*)^* + b^*$$

- $(b^* ab^* ab^*)^*$ → **represents number of a's = 2, 4, 6.....**
- b^* → **For 0 number of a's**

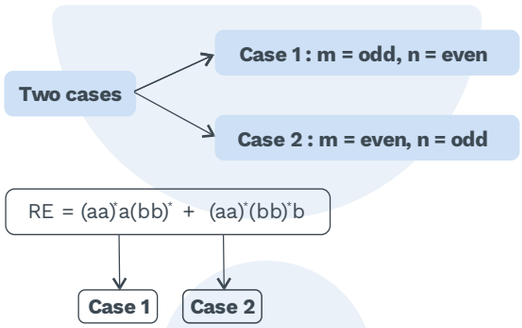
Q7 **Find the regular expression for the language :**

i) L = {$a^m b^n$ | m + n = even} **ii) L = {$a^m b^n$ | m + n = odd}**

Sol: **i)** $L = \{a^m b^n \mid m + n = \text{even}\}$



ii) $L = \{a^m b^n \mid m + n = \text{odd}\}$



Q8 **Find the regular expression for the language over $\sum$ = {a, b} :**

i) string starts with 'a' and |w| = even

ii) strings starts with 'a' and |w| = odd

Sol: **i)** $L = \{aa, ab, \ldots\ldots\ldots\ldots\}$

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

ii) $L = \{a, aaa, aab, \ldots\ldots\ldots\}$

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

Note:

A regular expression is not unique for a language. It means for any regular language, there are more than one (Infinite) regular expression possible.

Properties of regular expressions:

Let r, s and t be some regular expression.

i) Commutativity:

In this property, after changing the position of operands result will remain the same.

$$r + s = s + r$$
$$r.s \neq s.r$$

For some case:

r.s = s.r Example: r = a, s = a*

ii) Associativity:

Properties which allow us to regroup the operands when the operand is applied twice.

$$r + (s + t) = (r + s) + t$$
$$r.(s.t) = (r.s).t$$

iii) Distributive:

Property which involves two operators and states that one operator can be pushed down to be applied to each argument of the other operator individually.

Left distributive : $r(s + t) = rs + rt$
Right distributive : $(s + t)r = sr + tr$

Note:

$r + s.t \neq (r + s) . (r + t)$

iv) Identity:

When an operation is done between operand in the set and the identity element it results in the operand itself.

$\phi + r = r + \phi = r$ (ϕ is identity for union)
$\in .r = r. \in = r$ ($\in$ is identity for concatenation)

v) Annihilator:

$\phi.r = r.\phi = \phi$ (ϕ is annihilator for concatenation)

vi) Idempotent:

An operator is idempotent; if we apply it to two same operands, the output will be that operand itself.

$$r + r = r \text{ (Idempotence law for union)}$$

vii) Closure:

- $\phi^* = \in$
- $\phi^+ = \phi$
- $\in^+ = \in$
- $\in^* = \in$
- $\in + rr^* = r^*$
- $(r^*)^* = r^*$
- $r^+ = rr^* = r^*r$
- $(r^+)^* = r^*$
- $(r^*)^+ = r^*$
- $(r^+)^+ = r^+$
- $r^*r^* = r^*$
- $r^+r^* = r^+$
- $r^*r^+ = r^+$
- $(pq)^*p = p(qp)^*$ but $\begin{Bmatrix}(pq)^*q \neq p(qq)^* \\ (pq)^*r \neq p(qr)^*\end{Bmatrix}$

Note:

- $(a+b)^* = (a^*+b^*)^* = (a+b^*)^* = (a^*+b)^* = (a^*b^*)^*$
  $= (b^*a^*)^* = a^*(ba^*)^* = b^*(ab^*)^*$
- $(r+s)^* = (r+s)^* r^* = r^*(r+s)^*$

Previous Years' Question

Which one of the following regular expressions represents the language; set of all binary strings having two consecutive 0's and two consecutive 1's? **(GATE-2016 (set-1))**

1) (0+1)* 0011(0+1)*+ (0+1)*1100(0+1)*

2) (0+1)* (00(0+1)* 11+ 11(0+1)*00)(0+1)*

3) (0+1)* 00(0+1)*+ (0+1)*11(0+1)*

4) 00(0+1)* 11+11(0+1)*00

Sol: 2)

Rack Your Brain

Which of the following are equivalent?

A) (ab)*b **B)** a(bb)* **C)** a(ba)* **D)** (ab)*a

1) A) and C) only **2)** A) and D) only

3) C) and D) only **4)** B) and C) only

Rack Your Brain

Which of these are equivalent?

I) ∈ + r + rrr* **II)** ∈ + rr* **III)** r*

1) I and III only **2)** II and III only

3) All are equal **4)** None of these

Previous Years' Question

Which one of the following languages over the alphabet {0,1} is described by the regular expression :

(O+1)* 0(0+1)*0(0+1)*? **(GATE-2009)**

1) The set of all strings containing the substring 00

2) The set of all strings containing at most two 0's

3) The set of all strings containing at least two 0's

4) The set of all strings that begin and end with either 0 or1

Sol: 3)

3.2 EQUIVALENCE BETWEEN FINITE AUTOMATA AND REGULAR EXPRESSION

- Regular Expressions and Finite Automata are equivalent in their descriptive power.
- We can convert any Regular Expression into a finite automaton that recognises the language it describes and vice-versa.
- Definition of Regular language itself says the regular language is one that is recognised by some finite automaton.
- In other to show that the regular expressions define the same class of languages (i.e. regular language) accepted by finite automata, we must show that:

i) For every language, for which we have finite automata, there exists a regular expression also.

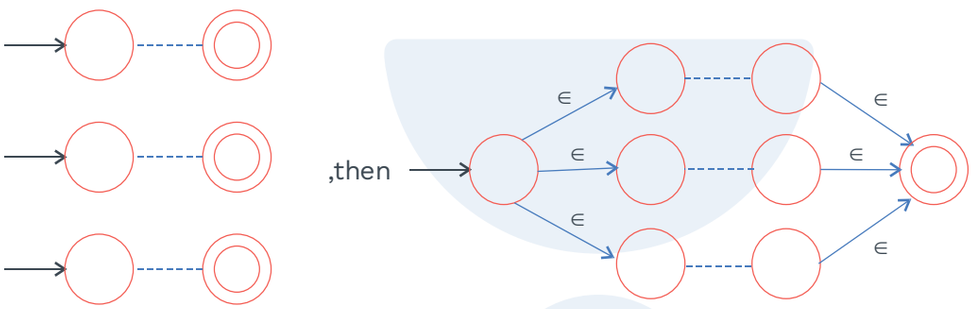
ii) Similarly, for every language, for which we have a regular expression there exists a finite automata also.

State elimination method:

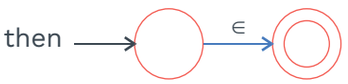
- State Elimination method can be applied for NFA, DFA, ∈-NFA to convert the finite machine into a regular expression.

Rules:

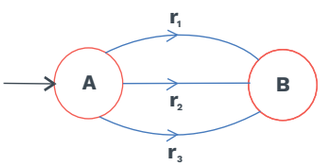
1) There should be only one initial and final state.



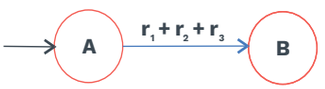
2) Separate initial state and accepting (final) state using ∈ transition,



3) Simplify parallel edge (parallel edge can be combined to a single edge),

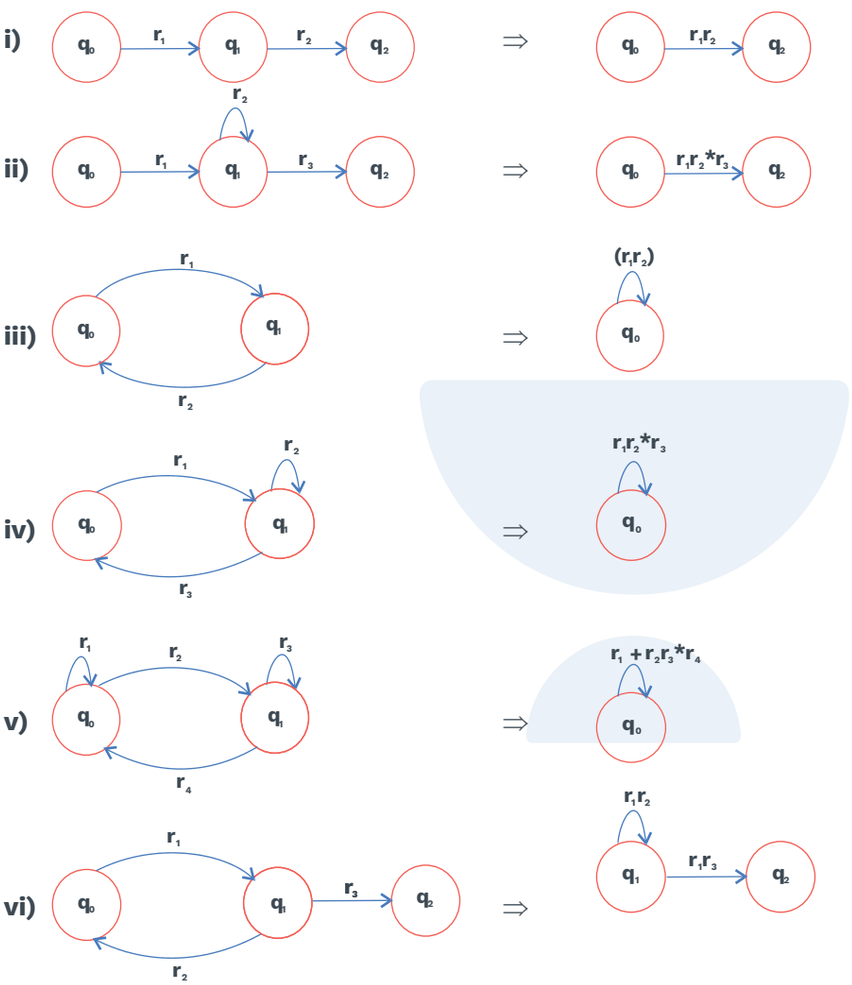


Then,

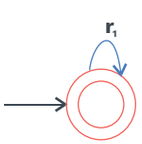


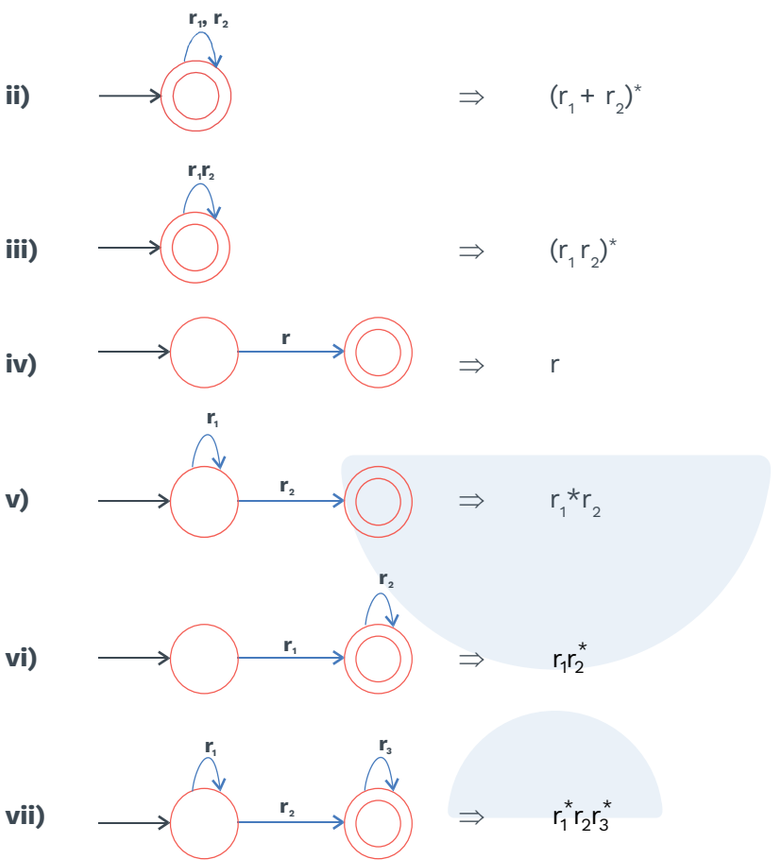
4) Elimination of nodes/states.

Example:



5) Continue to do the elimination of states till the transition graph takes any of these from:

Finite Automata	**Regular Expression**
i) 	$\Rightarrow \quad r_1^*$



Arden's method:

Let P, Q be the regular expression over $\sum$.

If P does not contain $\in$, then Regular expression R can be written as:

$\boxed{R = Q + RP}$ and it has a unique solution $\boxed{R = QP^*}$

It means that if we get an equation in the form of R = Q + RP, then the regular expression we get as a solution will be $\boxed{R = QP^*}$.

Note:

If P contains $\in$, then regular expression R will have an infinite number of solutions.

Proof of arden's theorem:

$R = Q + RP$

$= Q + (Q+RP)P$

$= Q + QP + RPP$

Put R = Q + RP again. On solving, it recursively

$R = Q + QP + QP^2 + QP^3 + \ldots\ldots\ldots\ldots$

$R = Q(\in + P + P^2 + \ldots\ldots\ldots\ldots)$

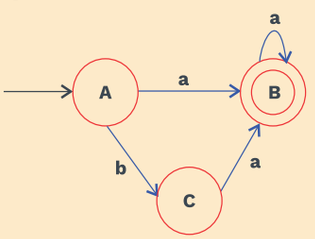
$\boxed{R = QP^*}$

Note:

We can write equation for every other state and start state based on all incoming arrows.

SOLVED EXAMPLES

Q1 **Calculate regular expression for A, B and C using Arden's lemma.**



Sol:

$A = \in$

$B = Aa + Ba + Ca$

$C = Ab$

$\boxed{A = \in}$

$\boxed{C = \in .b = b}$

$B = \in .a + Ba + ba$ (Put values of A and C in B's equation)

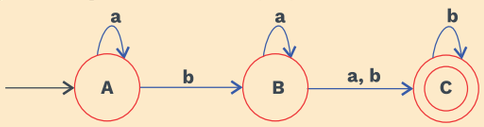
$B = \in .a + ba + Ba$

$B = \underset{R}{\square} \underbrace{a + ba}_{Q} + \underbrace{\square a}_{RP}$ (P does not contain $\in$)

$\therefore \quad B = QP^*$

$B = (a + ba)a^*$

Q2 **Calculate regular expression for A, B, C?**



Sol: Write equations based on given Finite Automata:

$A = \in + Aa$

$B = Ab + Ba$

$C = B(a + b) + Cb$

$\Rightarrow$ $\underset{R}{A} = \underset{Q}{\in} + \underset{RP}{Aa}$ (P does not contain $\in$)

$A = \in a^*$

$\boxed{A = a^*}$

$\Rightarrow$ $\underset{R}{B} = Ab + Ba = \underset{Q}{a^*b} + \underset{RP}{Ba}$ (P does not contain $\in$)

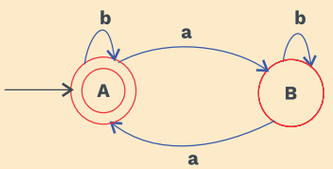
$\boxed{B = (a^*b)a^*}$

$\Rightarrow$ $C = B(a+b) + Cb$

$\underset{R}{C} = \underbrace{(a^*b)a^*(a+b)}_{Q} + \underset{RP}{Cb}$ (P does not contain $\in$)

$\boxed{C = a^*ba^*(a+b)b^*}$

Q3 **Calculate regular expression for A and B?**



Sol: Write equations based on given Finite Automata:

$A = \in + Ab + Ba$

$B = Aa + Bb$

$\Rightarrow$ $\underset{R}{B} = \underset{Q}{Aa} + \underset{RP}{Bb}$ (P does not contain $\in$)

$B = Aab^*$

$\Rightarrow$ $A = \in + Ab + Ba$

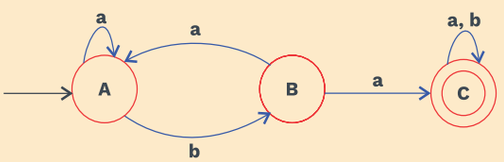
$A = \in + Ab + Aab^*a$

$\underset{R}{A} = \underset{Q}{\in} + \underbrace{A(b + ab^*a)}_{RP}$ (P does not contain $\in$)

$\boxed{A = \in (b + ab^*a)^*}$

$\boxed{B = (b + ab^*a)^* ab^*}$

Q 4 **Calculate regular expression for A , B and C?**



Sol: Write equations based on given Finite Automata:

$A = \in + Aa + Ba$

$B = Ab$

$C = Ba + C\ (a + b)$

$\Rightarrow \quad A = \in + Aa + Ba$

$A = \in + Aa + Aba$

$\underset{R}{A} = \underset{Q}{\in} + \underbrace{A(a + ba)}_{RP}$ (P does not contain $\in$)

$\boxed{A = \in (a + ba)^*}$

$\Rightarrow \quad B = Ab$

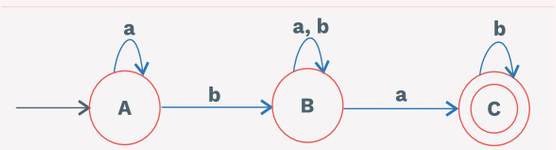
$\boxed{B = (a + ba)^* b}$

$\Rightarrow \quad C = Ba + C\ (a+b)$

$\underset{R}{C} = \underbrace{(a + ba) * ba}_{Q} + \underbrace{C(a + b)}_{RP}$

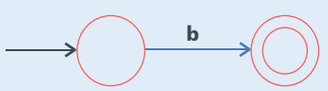
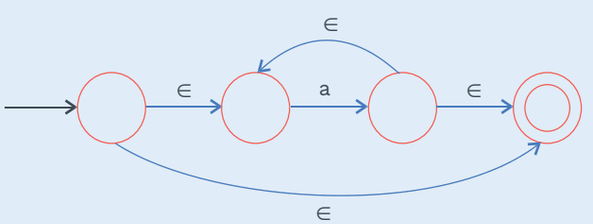
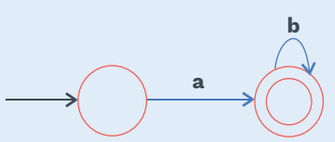
$\boxed{C = (a + ba)^* ba(a + b)^*}$

Rack Your Brain



Calculate regular expression for A, B, C using Arden's Theorem?

Conversion of finite automata to regular expression and vice-versa:
Conversion from regular expression to finite automata:

	Regular Expression	Finite Automata
i)	ϕ	
ii)	a	
iii)	b	
iv)	ab	
v)	a^*	
vi)	ab^*	

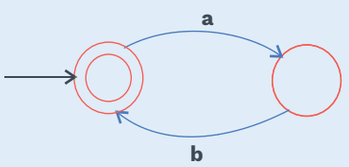
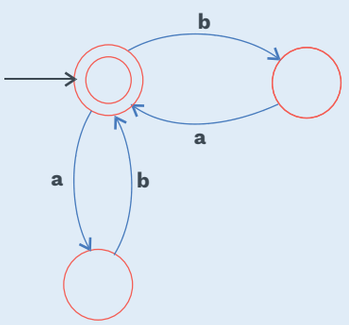
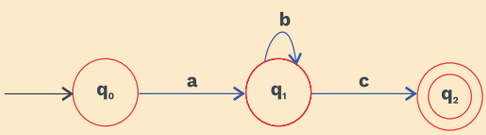
	Regular Expression	Finite Automata
vii)	$(ab)^*$	
viii)	$(ab + ba)^*$	

Table 3.1

Conversion from Finite Automata to Regular Expression:

SOLVED EXAMPLES

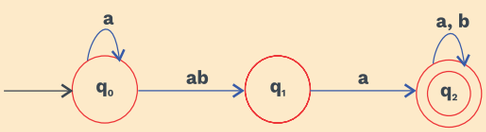
Q1 **Using State Elimination Method, find the regular expression for the given Finite Automata:**



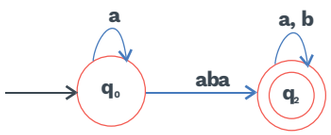
Sol: 

$RE = ab^*c$

Q2 **Using State Elimination Method, find the regular expression for the given Finite Automata:**

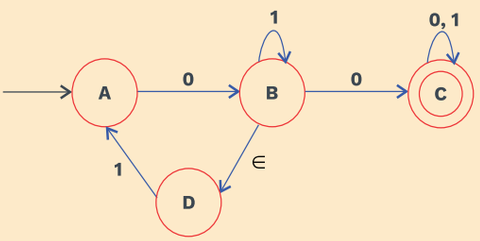


Sol:

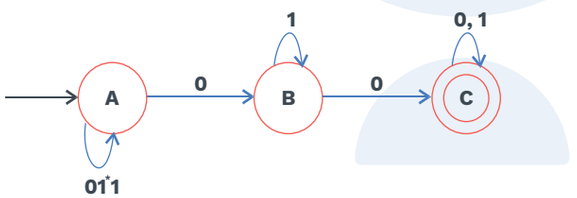


$$\boxed{RE = a^*aba(a+b)^*}$$

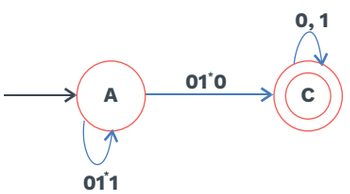
Q3 **Using State Elimination Method, find the regular expression for the given Finite Automata:**



Sol:

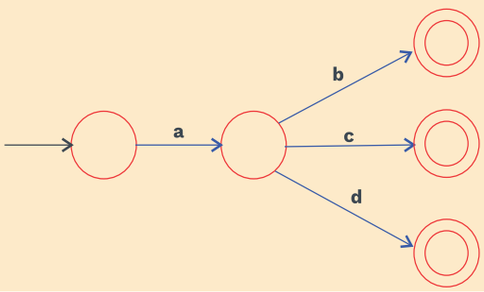


$\Rightarrow$

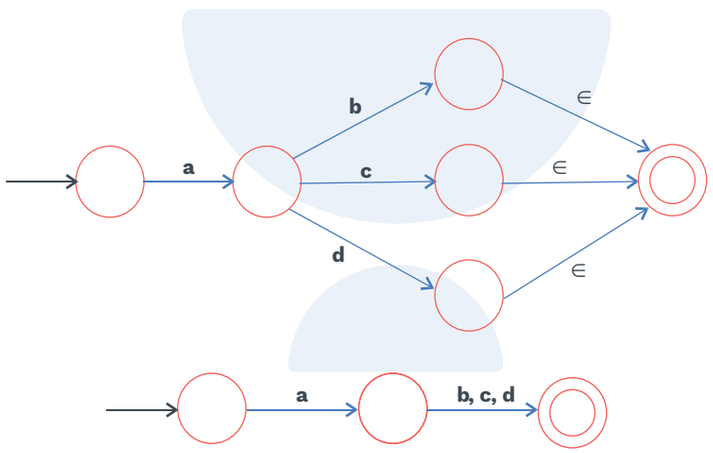


$\Rightarrow$ $\boxed{RE = (01^*1)^*01^*0(0+1)^*}$

Q4 **Using State Elimination Method, find the regular expression for the given Finite Automata:**

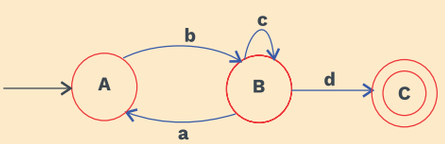


Sol:

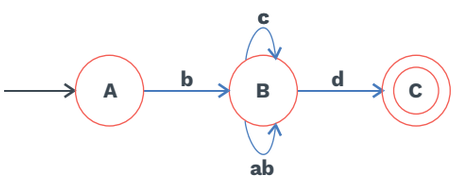


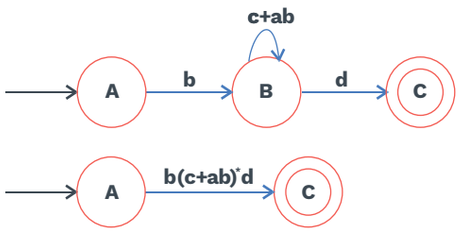
RE = a(b + c + d)

Q5 **Using State Elimination Method, find the regular expression for the given Finite Automata:**



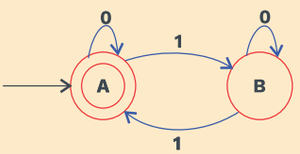
Sol:



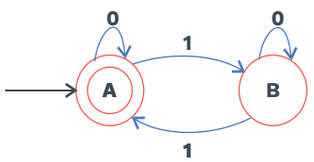


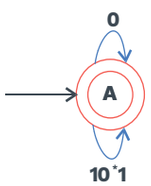
$RE = b(c + ab)^* d$

Q6 **Using State Elimination Method, find the regular expression for the given Finite Automata:**



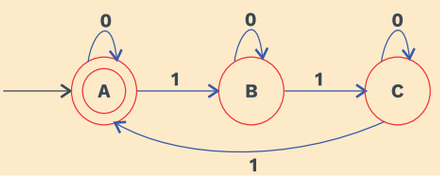
Sol:



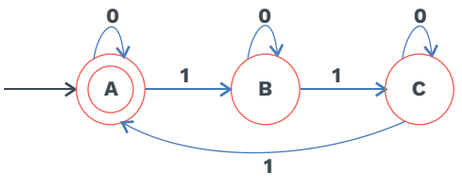


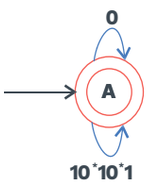
$RE = (0 + 10^*1)^*$

Q7 **Using State Elimination Method, find the regular expression for the given Finite Automata:**



Sol:





$\Rightarrow \quad (0+10^*10^*1)^*$

$$RE = \left(0 + 10^*10^*1\right)^*$$

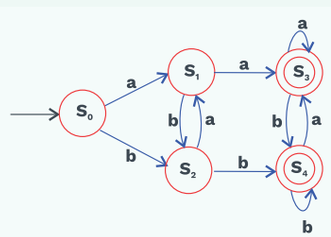
Previous Years' Question

Consider the regular expression:

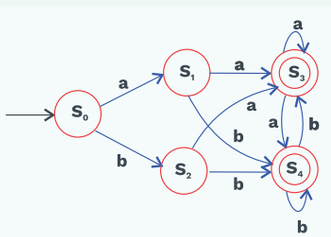
$$R = (a + b)^* (aa+bb) (a + b)^*$$

which deterministic Finite Automaton accepts the language represented by the regular expression R? **(GATE (IT) 2007)**

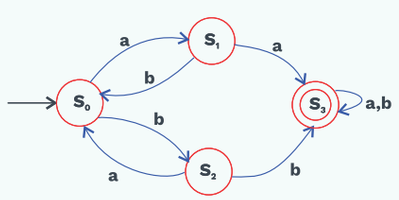
1)



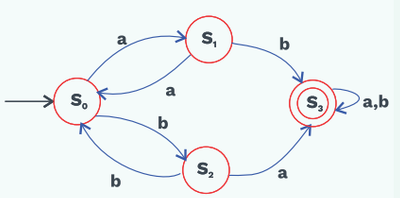
2)



3)



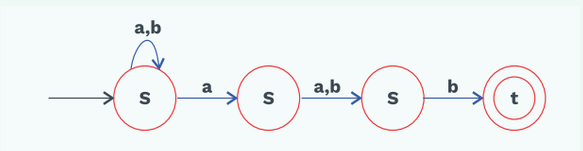
4)



Sol: 1)

Previous Years' Question

Which regular expression best describes the languages accepted by the non-deterministic automaton below? **(GATE (IT) 2006)**



1) (a + b)* a(a + b) b
2) (abb)*
3) (a+b)*a(a + b)*b (a+b)*
4) (a+b)*

Sol: 1)

3.3 REGULAR GRAMMAR

- Regular grammar and Regular languages are equal in power i.e., for every Regular language there exists a Regular grammar and vice–versa.
- **Types of regular grammar:**

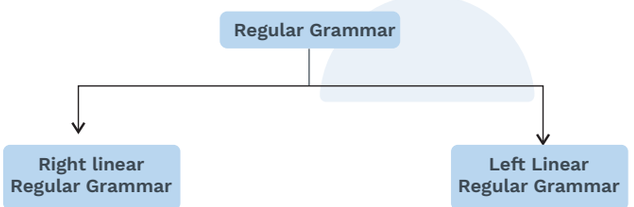


Fig. 3.2

Right and left linear regular grammar

Right linear regular grammar:

"A Grammar G = (V, T, S, P) is said to be right linear if all productions are in the form of:

$$A \rightarrow xB$$

$$A \rightarrow x$$

where A, B $\in$ V (Variables) and x $\in T^*$ (T = terminals)"

Left linear regular grammar:

"A Grammar G = (V, T, S, P) is said to be Left Linear Grammar if all productions are in the form of:

$$A \rightarrow Bx$$

$$A \rightarrow x$$

where A, B $\in$ V (Variables) and x $\in T^*$ (T = terminals)"

Note:

A regular grammar is one that is either right linear or left linear.

Example 1: Grammar $G_1 = (\{S\}, \{a, b\}, S, P_1)$ with P_1 given as:
$S \rightarrow abS/a$ is right linear
Grammar $G_2 = (\{S, S_1, S_2\}, \{a, b\}, S, P_2)$ with P_2 given as:
$S \rightarrow S_1ab$
$S_1 \rightarrow S_1ab|S_2$
$S_2 \rightarrow a$
is left linear.
Both G_1 and G_2 are regular grammar.

Example 2: The Grammar G = ({S, A, B}, {a, b}, S, P) with productions:
$S \rightarrow A$
$A \rightarrow aB/\in$
$B \rightarrow Ab$
Is not regular grammar.

Reason:
Even though here every production is in right or left linear form, but the grammar itself is neither right linear nor left linear. So, it is not Regular grammar.

Note:

The above example is linear grammar.

Linear grammar:
"A linear grammar is a grammar in which at most one variable can occur on the right side of any production, without restriction on the position of this variable."

Grey Matter Alert!

A regular grammar is always linear, but not all linear grammars are regular

Note:

The language generated by the right linear grammar is always regular.

Equivalence of regular grammar and regular language:

Theorem:

"A language L is regular if and only if there exists a regular grammar G such that L = L(G)"

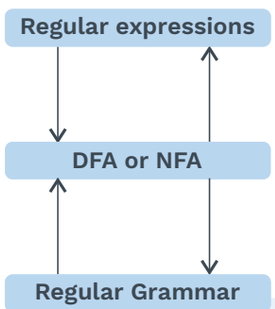


Fig. 3.3 Conversion of Regular expressions, Regular Grammar and Finite automata

Previous Years' Question

Language L_1 is defined by the grammar: **(GATE 2016 (SET-2))**

$S_1 \rightarrow aS_1b|\in$

Language L_2 is defined by the grammar:

$S_2 \rightarrow abS_2|\in$

Consider the following statements:

P : L_1 is regular.

Q : L_2 is regular.

Which one of the following is "TRUE"?

1) Both P and Q are true **2)** P is true and Q is false

2) P is false and Q is true **4)** Both P and Q are false

Sol: 3)

Rack Your Brain

Find a Regular Grammar that generates the language L $(aa^* (ab + a)^*)$

Rack Your Brain

Find a Regular Grammar for the language $L = \{a^n b^m \mid n + m \text{ is even}\}$

SOLVED EXAMPLES

Q1 **Construct the regular grammar for the following languages :**

I) $L = a^*$ **ii)** $L = a^+$

Sol:

i) $L = a^*$

$S \rightarrow Sa \mid \in$

(OR)

$S \rightarrow aS \mid \in$

ii) $L = a^+$

$S \rightarrow Sa \mid a$

(OR)

$S \rightarrow aS \mid a$

Q2 **Construct the regular grammar for the language that contains set of all the strings of a's and b's including $\in$.**

Sol: $\sum = \{a, b\}$

Regular expression = $(a + b)^*$, $L = \{\in, a, b, aa, ab, ...\}$

$S \rightarrow aS \mid bS \mid \in$

Q3 **Construct the regular grammar for the language that contains set of all the strings of a's and b's excluding $\in$.**

Sol: $\sum = \{a, b\}$

Regular expression = $(a + b)^+$, $L = \{a, b, aa, ab, ...\}$

$S \rightarrow aS \mid bS \mid a \mid b$

Q4 **Construct the regular grammar for the language that generates set of all the strings of a's and b's where every string starts with 'a'.**

Sol: Regular expression = a (a + b)*
L = {a, aa, ab, ...}

Grammar:
S → aA
A → aA|bA|∈

Conversion from finite automata to regular grammar:

Regular Grammar Corresponding to the given Finite Automata:

A → aB (There will be one production for every outgoing edge.
B → aB | bB |∈ For the final state, we will add null production in the grammar.)

It produces the Right Linear Grammar.
If we reverse the Right-Hand Side of the production of Right Linear Grammar, we will get Left Linear Grammar.

A → Ba
B → Ba | Bb |∈ (Left Linear Grammar)

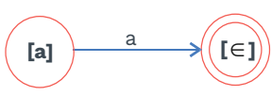
Right linear grammar to ∈-NFA:

1) If S → α is a production, then [S] is a start/initial state and all the suffix of RHS of the production will be the other states of ∈–NFA.

2) If S → α is a production, then δ (S, ∈) = [α]

3) If S → α is a production, then δ (aα, a) = [α]

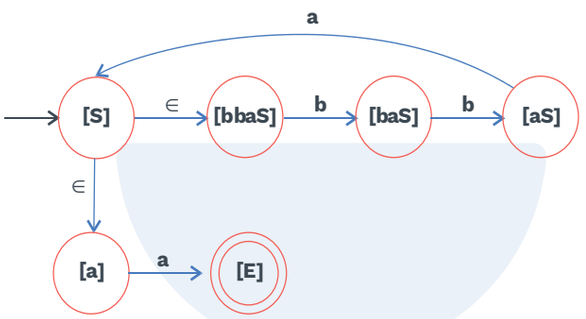
4) If S → α is a production, then for every terminal, a ∈ T
δ (a, a) = ∈ which is final state.



SOLVED EXAMPLES

Q1 **S ⟶ bbaS|a**

Sol: Q(States) = {[S], [bbaS], [baS], [aS], [a], [E]}
Initial State = q_0 = [S]
Final state = F = [E]



$\delta(S, \in) = [bbaS]$

$\delta(S, \in) = [a]$

$\delta(bbaS, b) = [baS]$

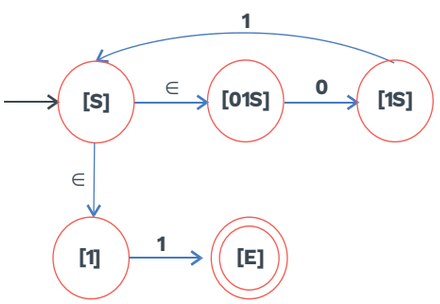
$\delta(baS, b) = [aS]$

$\delta(aS, a) = [S]$

$\delta(a, a) = [E]$

Q2 **S ⟶ 01S|1**

Sol: Q(States) = {[S], [01S], [E], [1], [1S]}
Initial State = [S]
Final state = [E] (End State)



$\delta(S, \in) = [01S]$

$\delta(S, \in) = [1]$

$\delta(01S, 0) = [1S]$

$\delta(1S, 1) = [S]$

$\delta(1, 1) = [E]$

Left linear grammar to $\in$-NFA:

Step 1: Reverse the Right-hand side of every production.
Step 2: obtain $\in$–NFA.
Step 3: Interchange initial and final state.
Step 4: Change the direction of the edges.

SOLVED EXAMPLES

Q1 **S $\longrightarrow$ S110|0**

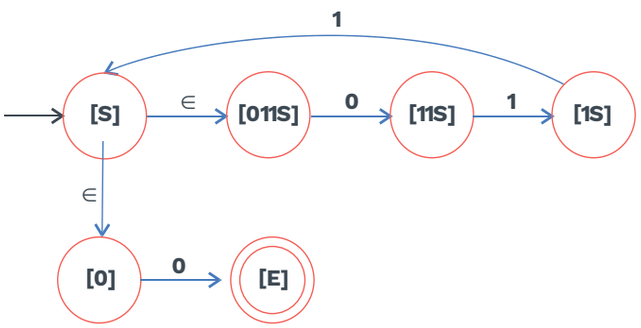
Sol:

Step-1: Reverse the right-hand side of every production and get the right linear Grammar.

S $\rightarrow$ 011S|0

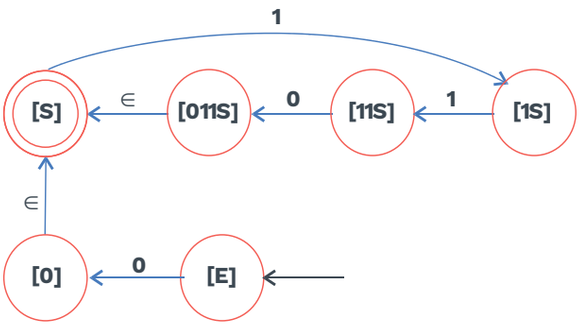
Step-2: Obtain $\in$–NFA

Q(States) = {[S], [011S], [11S], [1S], [0], [E]}



Step-3: Make the initial state as final state and the final state as initial state.

Step-4: Change the direction of the edges.



Q2 S ⟶ S100|1

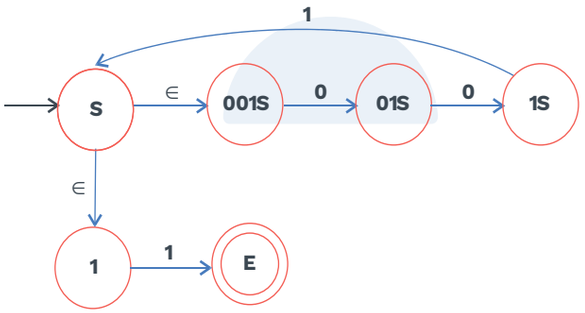
Sol:

Step-1 : Reverse the right-hand side of every production and get Right Linear Grammar.

S → 001S|1

Step-2: Convert to ∈–NFA

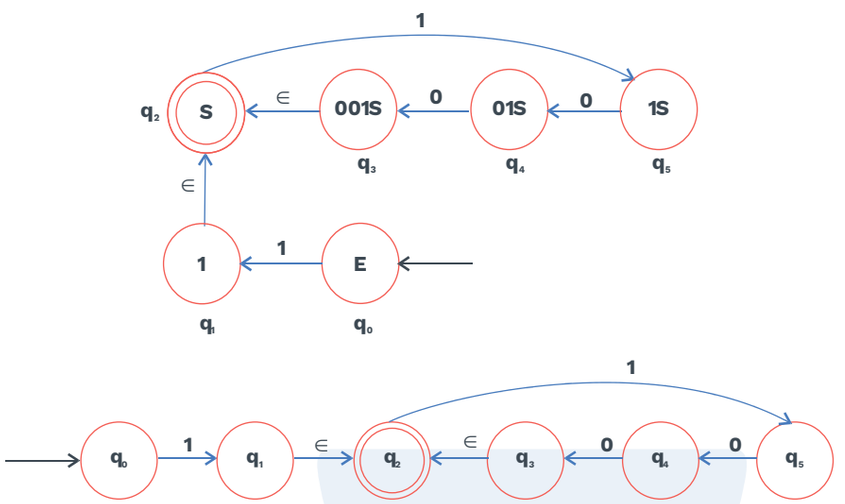
Q(States) = {[S], [001S], [01S], [1S], [1], [E]}



δ (S, ∈) = [001S]
δ (S, ∈) = [1]
δ (001S, 0) = [01S]
δ (01S, 0) = [1S]
δ (1S, 1) = [S]
δ (1, 1) = [E]

Step-3: Interchange initial and final states.

Step-4: Change the direction of edges:



3.4 REGULAR LANGUAGES

The language which is accepted by Finite Automata (or) the language which is generated by regular expression (or) the language which is generated by Regular grammar is called Regular Language.

Definition

A language L is called regular if and only if there exists some deterministic finite acceptor M such that L = L(M).

Note:

- Every formal languages must be either Regular (or) Non regular language.
- The language which is not accepted by Finite Automata is known as Non-Regular Language.
- Every finite language is Regular.
  Example : L = {b, ba, bab} is finite language then Regular .
- $L = \phi$ (Empty Language) is Regular language.
- $L = \sum^*$ (Universal Language) is Regular language.
- Regular Language can be finite (or) infinite.
- Every Non-Regular Language is Infinite.
- Every subset of a Regular language need not to be Regular.
  Example : $L_1 = \{a^m b^n | m, n \geq 1\}$ is Regular language.
  But $L_2 = \{a^m b^n | m = n\}$ which is a subset of L_1 i.e non Regular Language.

Example : $L_1 = \{a^m b^n | m, n \geq 1\}$ is Regular language.

But $L_2 = \{a^m b^n | m = n\}$ which is a subset of L_1 i.e non Regular Language.

- Every Finite subset of Regular language is a Regular Language.
- Every Finite subset of Non-Regular language is a Regular Language.
- Every subset of Non-Regular Language need not be Non-Regular.

  $L = \{a^n b^n | n \geq 0\}$ is Non-Regular Language.

  $L_1 = \{\in, ab, aabb\}$ is Regular Language.
- Superset of Regular Language can be Regular (or) cannot be Regular.

  Example : $L_1 = \{\in\}$ Regular Language.

  $L_2 = \{\in, ab\}$ is Regular Language.

  $L_3 = \{a^m b^m | m \geq 0\}$ is Non-Regular Language.
- Superset of Non-Regular Language need not be Non-Regular.

  Example : $L = \{a^n b^n | n \geq 1\}$ is Non-Regular Language.

  $L_1 \supset L$

  $L_1 = \{a^m b^n | m, n \geq 1\}$ is Regular Language.
- Every language defined over alphabet $\sum$ has a regular subset.

Closure properties of regular language

Closure properties of regular languages are defined as the certain operation on regular language, which when applied to the language, results into the language of the same type.

i) Closure under union:

Theorem:

"If L_1 and L_2 are regular languages, then $L_1 \cup L_2$ will also be a regular language only."

Proof:

Since L_1 and L_2 are regular, they have regular expressions, say $L_1 = L(R)$ and $L_2 = L(S)$, then $L_1 \cup L_2 = L(R + S)$ by the definition of the + operator for regular expressions.

ii) Closure under complementation:

Theorem:

"If L is a regular language over alphabet $\sum$, then $\overline{L} = \sum^* - L$ is also a regular language."

Proof:

Let L = L(A) for some DFA A = $(Q, \sum, \delta, q_0, F)$, then $\overline{L} = L(B)$, where B is the DFA $(Q, \sum, \delta, q_0, Q - F)$ i.e., B is exactly like A, but the accepting states of A become non-accepting states of B and vice-versa.

iii) Closure under intersection:

Theorem:

"If L_1 and L_2 are regular languages, then so is $L_1 \cap L_2$."

Proof:

By Demorgan's law:

$L_1 \cap L_2 = \overline{(\overline{L_1} \cup \overline{L_2})}$

If L_1 and L_2 are regular, then by closure under complementation, so are $\overline{L_1}$ and $\overline{L_2}$.

Using closure under union, $\overline{L_1} \cup \overline{L_2}$ is regular.

Using closure under complementation once more

$\overline{(\overline{L_1} \cup \overline{L_2})} = L_1 \cap L_2$ is regular.

iv) Closure under difference:

Theorem:

"If L_1 and L_2 are regular languages, then so is $L_1 - L_2$."

Proof:

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

As we know, $\overline{L_2}$ is regular, and using the closure of intersection $L_1 \cap \overline{L_2}$ is also closed. Thus $L_1 - L_2$ is also regular.

v) Closure under reversal:

Theorem:

"If L is a regular language, so is L^R."

Proof:

Let L is Regular Language. We can construct a Non-Deterministic Finite Automata with a single final state for it.

In the transition graph for this Non-Deterministic Finite Automata,

a) Make the initial state as the final state.

b) Make the final state as the initial state.

c) Reverse the direction of all the edges.

Now the modified Non-Deterministic Finite Automata accepts w^R if and only if the original Non-Deterministic Finite Automata accepts w.

vi) Closure under homomorphism:

Homomorphism:

Definition

A string homomorphism is a function on strings that works by substituting a particular string for each symbol.

Example: The function h given by h(0) = abb and h(1) = ba is a homomorphism which replaces each 0 by abb and each 1 by ba. Thus, h(1011) = baabbbaba.

Theorem:

"If L is a regular language over alphabet $\sum$, and h is a homomorphism on $\sum$, then h(L) is also regular."

vii) Closure under Inverse homomorphism:

Inverse homomorphism:

We can apply homomorphism backwards also.

Regular languages are closed under this.

The inverse of homomorphic image of a language L is h^{-1} (L) = set of strings w in $\sum^*$ such that h(w) is in L; if h is a homomorphism from some alphabet $\sum$ to strings in another alphabet T, and L be a language over alphabet T.

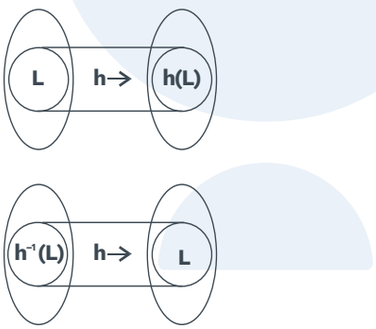


Fig. 3.4 A Homomorphism Applied in the Forward and Inverse Direction

Example 1. Let $\sum$ = {0, 1, 2} and Δ = {a, b}.

Defined h by h(0) = a, h(1) = ab, h(2) = ba.

Let L_1 = {ababa}

$h^{-1}(L_1)$ = {110, 022, 102}

Example 2. Let $\sum$ = {0, 1} and Δ = {a, b}.

Defined h by h(0) = aa, h(1) = aba.

Let L = $(ab + ba)^*a$

= {a, aba, baa, ababa, ...}

- Inverse homophoric image for a, is not possible.
- Inverse homophoric image for aba is possible.

h^{-1} (aba) = {1}

Rack Your Brain

Consider the homomorphism h from the alphabet {0, 1, 2} to {a, b} defined by : h(0) = ab, h(1)=b, h(2)=aa

a) What is h(2201)?

b) If L is the language 1*02, what is h(L)?

viii) Closure under quotient operation:

Regular languages are closed under Quotient operation.

Right quotient:

Let L_1 and L_2 be language on the same alphabet, then the right quotient of L_1 with L_2 is defined as:

$L_1/L_2 = \{x : xy \in L_1 \text{ for some } y \in L_2\}$

Example 1. $L_1 = \{01, 001, 101, 0001, 1101\}$

$L_2 = \{01\}$

$L_1/L_2 = \{\in, 0, 1, 00, 11\}$

Example 2. $L_1 = 10^*1$; $L_2 = 0^*1$

$L_1 = \{11, 101, 1001, 10001, ...\}$

$L_2 = \{1, 01, 001, 0001, ...\}$

$L_1/L_2 = \{1, 10, 100, 1000, ...\}$

$= 10^*$

Note:

Few more examples of Right Quotient:

$a/abc = \Phi$

$abc/c = ab$

$abc/abc = \in$

ix) Closure under INIT operation:

Regular sets are closed under INIT (all the prefixes of given language L) operation.

Example: L = {a, ab, aabb}

INIT(L) = {∈, a, ∈, a, ab, ∈, a, aa, aab, aabb}

= {∈, a, aa, ab, aab, aabb}

x) Closure under substitution:

Substitution: Let ∑, Δ two alphabets, then substitution is a mapping defined over the alphabet '∑' such that symbols of ∑ can be replaced

by regular language of another alphabet 'Δ'
Regular sets are closed under substitution.

Example: $\sum = \{a, b\}$

$f(a) = 0^*$

$f(b) = 01^*$

$L = a + b^*$

$f(L) = 0^* + (01^*)^*$

which is again a regular expression, which will accepted by some finite Automata.

Note:

- Finite Union of Regular languages is Regular language.
- Infinite Union of Regular language need not be a Regular language i.e. Regular language is not closed under Infinite Union.

Example: $L_1 = \{ab\}$

$L_2 = \{a^2b^2\}$

$L_3 = \{a^3b^3\}$ .....so on

$L = L_1 \cup L_2 \cup L_3 \cup$ .....

$= \{a^nb^n \mid n \geq 1\}$ which is not regular.

Note:

- Finite Intersection of Regular Language is Regular Language.
- Infinite Intersection of Regular Language need not be Regular Language.

xi) Closure under symmetric difference:

$L_1 \oplus L_2 = L_1 \Delta L_2$

$= (L_1 \cup L_2) - (L_1 \cap L_2)$

$= (L_1 - L_2) \cup (L_2 - L_1)$

As we know, Regular language is closed under union, intersection and set-difference.

Therefore, It is also closed under symmetric difference.

Rack Your Brain

The nor of two languages is :

$$\text{nor}(L_1, L_2) = \{w : w \notin L_1 \text{ and } w \notin L_2\}$$

Find whether the family of regular language is closed under the nor-operation.

Regular language (L) is closed under:

Operations	Yes/No
1) Union	Yes
2) Substitution	Yes
3) Complementation	Yes
4) Set-difference	Yes
5) Concatenation	Yes
6) Reversal	Yes
7) Kleene Closure	Yes
8) Positive Closure	Yes
9) Subset (L)	No
10) Prefix (L)	Yes
11) Suffix (L)	Yes
12) Substring (L)	Yes
13) Substitution	Yes
14) Homomorphism	Yes
15) Symmetric difference	Yes

Table 3.2 Closure Properties of Regular languages

Decision properties of regular language:

A decision property for a class of languages is an algorithm which takes a formal description of a language and says whether some property holds or not.

Example: Is language L empty?

Example: Is a particular string w in the described language?

Example: Do two descriptions of a language actually describe the same language?

i) Testing emptiness of regular languages:

Emptiness problem of Regular Language is decidable.

E_{DFA} = {<D>| D is a DFA and L(D) = ϕ}

Algorithm:

Step 1: Select the states which are not reachable from the initial state and delete those states and corresponding transitions.

Step 2: In the remaining finite Automata, if we find at least one final state, then the language accepted by the given finite Automata is non-empty otherwise, empty.

ii) Testing finiteness of regular language:

The finiteness problem of Regular language is decidable.

Algorithm:

Step 1: Remove the state which is not reachable from initial state and corresponding transitions.

Step 2: Delete the states and corresponding transitions from which we cannot reach the final state.

Step 3: In the remaining Finite Automata, if we find at least one loop and one final state, which means it accepts infinite language.

Note:

Finite Automata will accept a infinite language only when there is a loop exists and loop should be reachable from initial state, and from the loop, able to reach final state.

iii) Testing equality problem of regular languages:

The equality problem of Regular language is decidable.

EQ_{DFA} = {<D_1, D_2 >| D_1 and D_2 are DFA and L(D_1) = L(D_2)}

Is $L_1 = L_2$?

FA_1 FA_2

For regular language L_1, we can make a Finite Automata FA_1 and similarly for regular language L_2. We can make finite Automata FA_2. So, corresponding DFA will exist for both languages, which can be further minimised.

Thus, whether two regular languages are equal or not is decidable.

Note:

Consider G is a regular grammar.

$w \in L(G) \rightarrow$ Membership Problem

$L(G) = \Phi \rightarrow$ Emptiness Problem

L(G) is finite? ->Finiteness Problem

$L(G) = \Sigma^* \rightarrow$ Completeness(Totality) Problem

$L(G1) = L(G2) \rightarrow$ Equivalence of two regular language Problem

All these problems are decidable for regular language.

Pumping lemma:

- It is evident from the many examples that Regular languages can be infinite.
- However, the regular language associated with automata having finite memory imposes restrictions on the structure of Regular language.
- Pumping lemma is a theorem about regular languages which is used to prove that some of the languages are not regular.

Theorem:

"If L is an infinite regular language. Then there exists some positive integer n such that any $w \in L$, $|w| \geq n$ can be decomposed as:

w = xyz such that

i) $|y| \geq 1$

ii) $|xy| \leq n$

iii) for all $i \geq 0$, the string xy^iz is also in L."

Note:

- Every regular language satisfies the pumping lemma property.
- If a language 'L' does not satisfy the pumping lemma property, then 'L' is non-regular.

Example: $L = \{a^nb^n | n \geq 1\}$

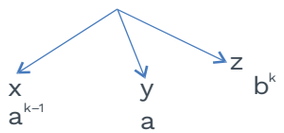
Sol: Let's assume L is a regular language (proof by contradiction)

$w \in L$

$w \in a^kb^k$

Choose a 'k' such that $|a^kb^k|$. Here $2k \geq n$

$w = a^kb^k$



$|xy| = |a^{k-1}a| = |a^k|$ (Here $k \leq 2k$)

$|y| \neq 0$

Hence, $xy^iz \in L$; $\forall\ i \geq 0$

For $i = 2$, $a^{k-1}a^2b^k = a^{k+1}b^k \notin L$

$\therefore$ L is Non-Regular.

Note:

Special Case of pumping Lemma.

When $\sum = \{a\}$ i.e. singleton set. Then lengths of string must follow arithmetic progression for a language to be a regular language.

Example 1 : $L = \{a^{2n+1} \mid n \geq 0\}$
$= \{a, aaa, aaaaa, ...\}$
Regular language. **(Following arithmetic progression)**

Example 2 : $L = \{a^{3n} \mid n \geq 1\}$
$= \{aaa, aaaaaa, ...\}$
Regular language. **(Following arithmetic progression)**

Example 3 : $L = \{a^{2^n} \mid n \geq 0\}$
$= \{a, aa, aaaa, ...\}$ **(Not following arithmetic progression)**
Not Regular language.

Which of the following languages are regular?

1) $L = \{a, ab, aba\} \rightarrow$ Finite set
It is Regular Language.

2) $L = \{w \in \{a, b\}^*, |w| = 20\} \rightarrow$ Regular Language

3) $L = \{w \in \{a, b\}^*, |w| = r \pmod n\} \rightarrow$ Regular Language

4) $L = \{w \in \{a, b\}^*, |w| \geq 20\} \rightarrow$ Regular Language

5) $L = \{a^mb^n \mid m, n \geq 0\} \rightarrow$ Regular Language

6) $L = \{a^mb^n \mid m \geq 2020, n \geq 2021\} \rightarrow$ Regular Language

7) $L = \{a^mb^n \mid 1 \leq m \leq 200, 20 \leq n \leq 400\} \rightarrow$ Regular Language

8) $L = \{a^mb^n \mid m + n = 20\} \rightarrow$ Finite $\rightarrow$ Regular Language

9) $L = \{a^mb^n \mid mn = \text{Finite}\} \rightarrow$ Regular Language

10) $L = \{a^mb^n \mid 1 \leq m \leq n \leq 40\} \rightarrow$ Regular Language

11) $L = \{a^nb^n \mid n \geq 1\} \rightarrow$ Non-Regular Language

12) $L = \{a^m b^n | \ m = n, m, n \geq 0\} \rightarrow$ Non-Regular Language
13) $L = \{a^m b^n | \ m < n\} \rightarrow$ Non-Regular Language
14) $L = \{a^m b^n | \ m \neq n, m, n \geq 0\} \rightarrow$ Non-Regular Language
15) $L = \{a^m b^n | \ m = n^2\} \rightarrow$ Non-Regular Language
16) $L = \{a^m b^n | \text{gcd}(m, n) = 1\} \rightarrow$ Non-Regular Language
17) $L = \{wcw^R | \ w \in (a, b)^*\} \rightarrow$ Non-Regular Language
18) $L = \{wxw^R | \ w, x \in (0, 1)^+\} \rightarrow$ Regular Language
19) $L = \{a^n b^n c^n | \ n \geq 1\} \rightarrow$ Non-Regular Language
20) $L = \{wxw^R | \ w, x \in (0, 1)^+, |x| = 5\} \rightarrow$ Non-Regular Language

Rack Your Brain

Is the language $L = \{a^n | \ n \text{ is a perfect square}\}$ Regular or Non-Regular?

Previous Years' Question

Which of the following language is/are Regular? **(GATE 2015 SET-2)**

L_1 : $\{wxw^R | \ w, x, \in \{a, b\}^*$ and $|w|, |x| > 0\}$, w^R is the reverse of string w.

L_2 : $\{a^n b^m | \ m \neq n$ and $m, n \geq 0\}$

L_3 : $\{a^p b^q c^r | \ p, q, r \geq 0\}$

1) L_1 and L_3 only
2) L_2 only
3) L_2 and L_3 only
4) L_3 only

Sol: 1)

Rack Your Brain

Find whether $L = \{a^p | \ p \text{ is a prime number}\}$ is Regular or Non-Regular?

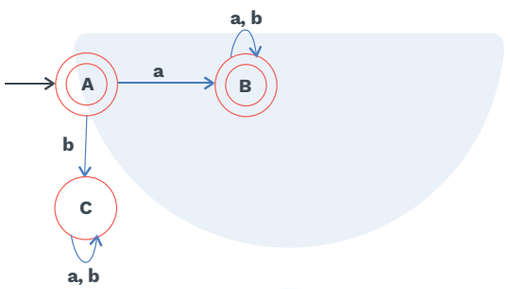
Myhill-nerode theorem

- Myhill-Nerode Theorem provides a necessary and sufficient condition for a language to be a regular.
- The Myhill-Nerode Theorem states that L is regular if and only if R_L has a finite number of equivalence classes and moreover that the number

of states in the minimal DFA recognizing L is equal to the number of equivalences classes in R_L.

- The language L defines over the alphabet '$\sum$' is Regular iff the number of equivalence classes with respect to 'L' is finite.
- In minimal finite automata, some language can be defined at each and every state. These languages are mutual disjoint and they are known as equivalence classes.
- Union of all these equivalence classes = $\sum^*$
- Union of some of the equivalence classes = Language 'L' accepted by finite automata.

Example 1:



$A = \{\in\}$
$B = a(a+b)^*$ } Equivalence Classes
$C = b(a+b)^*$

Union of all these equivalence classes = $A \cup B \cup C = \sum^*$

Union of some of the equivalence classes = $A \cup B$

= language accepted by finite automata

- The language 'L' is Regular if and only if 'L' is accepted by finite automata.

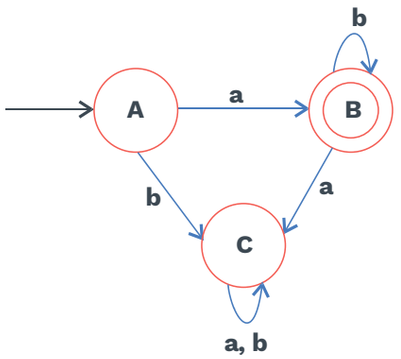
$\Leftrightarrow$ The number of states is finite.

$\Leftrightarrow$ The number of equivalence classes with respect to 'L' is finite.

- The language 'L' is not regular $\Leftrightarrow$ The number of equivalence classes with respect to 'L' is infinite.

Example 2: Let $L = ab^*$

The minimal DFA for the $L = ab^*$ is:



The equivalence classes are:

$A = \{\in\}$

$B = ab^*$ (State B refers to the strings in the language)

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

- L is regular (There are 3 equivalence class that and has a minimal DFA with 3 states).
- L itself is contained entirely in one equivalence class that corresponds to the fact that the minimal DFA has only one accepting state.

Chapter Summary

- **Regular expression:** Regular expression is one way to describe the regular languages via the notation.
- **Operators of regular expression:** Union, Concatenation, Closure (Kleene Closure/ Star Closure).
- **Order of precedence of regular expression operators:**

$$* > \bullet > +$$

- **Equivalence between regular language and regular expression:** For every regular language, there exists a regular expression and for every regular expression, there is a regular language.
- **Properties of regular expression:** Commutativity, Associativity, Distributive, Identity, Idempotent, Closure.
- **Equivalence between regular expression and finite automata:** Every language defined by a regular expression and a regular expression is defined by Finite Automata.
- **Conversion of finite automata to regular expression:** i) State Elimination Method and ii) Arden's Method.
- **Regular grammar:** Another way of describing the regular languages. It is also known as Type 3 grammar in Chomsky Hierarchy.
- The Grammar 'G' is said to be type 3 or Regular if every production is in the form: $A \rightarrow xB \mid x$ or $A \rightarrow Bx \mid x$, where $A, B \in V, x \in T^*$
- **Types of regular grammar:** Right linear Grammar and left linear Grammar.
- **Regular Language:** The language which is accepted by finite Automata, generated by Regular expression and generated by Regular Grammar is known as Regular language.
- **Myhill-nerode theorem:** "Myhill-Nerode Theorem provides a necessary and sufficient condition for a language to be a regular.

4 Context-Free Language and Pushdown Automata

4.1 CONTEXT-FREE GRAMMAR

- Context-free grammars are a more powerful method of describing languages.
- The productions in a regular grammar are restricted in two ways:
  The left side must be a single variable, while the right-hand side of the production has a constraint as we know for RLG: $A \rightarrow xB|x$ and for LLG: $A \rightarrow Bx|x$ where $A, B \in V$ and $x \in T^*$.
- To create grammars which are more powerful, we must relax some of these restrictions.
- By retaining the restriction on the left side but permitting anything on the right, we get context-free grammar.

Definition:

- A grammar G = (V,T,S,P) is said to be context-free if all productions in P have the from:

$$A \rightarrow x$$

Where, $A \in V$ and $x \in (VUT)^*$

Where,

T = Terminals.
V = Non-Terminals or variables.
S = Start symbol.
P = Finite set of rules or productions that represent the recursive definition of a language.

- Each production consists of:
  - **a)** A variable that is being (partially) defined by the production. This variable is often called the head of the production.
  - **b)** The production symbol →.
  - **c)** A string of zero or more terminals and variables. This string is called the body of the production.

Note:

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

Example: The Grammar G = ({S}, {a,b}, S,P) with productions:

$S \rightarrow aSa$,
$S \rightarrow bSb$,
$S \rightarrow \in$
is context-free.

A typical derivation in this grammar is:

$S \Rightarrow aSa$
$\Rightarrow aaSaa$
$\Rightarrow aabSbaa$
$\Rightarrow aabbaa$

i.e., $L(G) = \{ww^R \mid w \in \{a,b\}^*\}$ which is context-free language.

Derivation of string: left most derivation and right most derivation

- In a grammar that is not linear, a derivation may involve sentential forms with more than one variable.
- In such cases, the variables can be replaced in order of choice

Example: The grammar G = ({A,B,S} , {a,b}, S, P) with productions.

$S \rightarrow AB$
$A \rightarrow aaA$
$A \rightarrow \in$
$B \rightarrow Bb$
$B \rightarrow \in$

The above grammar generates the language $L(G) = \{a^{2n}b^m \mid n \geq 0, m \geq 0\}$.

Left most derivation for the string "aab"

Step 1 : $S \Rightarrow AB$
Step 2 : $S \Rightarrow aaAB$
Step 3 : $S \Rightarrow aaB$
Step 4 : $S \Rightarrow aaBb$
Step 5 : $S \Rightarrow aab$

Right most derivation for the string "aab"

Step 1 : $S \Rightarrow AB$
Step 2 : $S \Rightarrow ABb$
Step 3 : $S \Rightarrow Ab$
Step 4 : $S \Rightarrow aaAb$
Step 5 : $S \Rightarrow aab$

Definition

A derivation is said to leftmost if in each step, the leftmost variable in the sentential form is replaced. If in each step, the rightmost variable is replaced, derivation is known as rightmost.

Previous Years' Question

Identify the language generated by the following grammar, where S is start variable.

(GATE-2017 (Set-2))

S→XY

X→zX|a

Y→aYb|∈

1) $\{a^m b^n | m \geq n, m > 0\}$ **2)** $\{a^m b^n | m \geq n, n \geq 0\}$

3) $\{a^m b^n | m > n, m \geq 0\}$ **4)** $\{a^m b^n | m \geq n, n > 0\}$

Sol: Option 3)

Previous Years' Question

Which of the following languages is generated by the given grammar?

(GATE-2016 (Set-1))

S→aS|bS|∈

1) $\{a^n b^m\} | n, m \geq 0\}$

2) {w∈{a, b}* |w has equal number of a's and b's}

3) $\{a^n | n \geq 0\} \cup \{b^n | n \geq 0\} \cup \{a^n b^n | n \geq 0\}$

4) {a, b}*

Sol: Option 4)

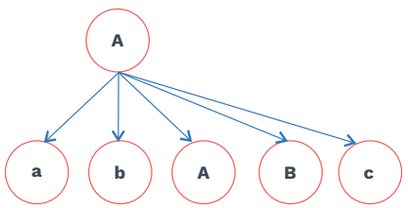
Derivation trees:

A second way of showing derivations, independent of the order in which productions are used; is by a derivation tree or parse tree.

Definition

A derivation tree is an ordered tree in which nodes are labelled with the left sides of production and in which the chileren of a node represent its corresponding right sids

Example: A → abABc



Grey Matter Alert!

Let G = (V, T, S, P) be a context free grammar. An ordered tree is a derivation tree for G if and only if it has the following propertie:

a) The root is labeled S.

b) Every leaf has a label from T∪{∈}.

c) Every interior vertex (a vertex that is not a leaf) has a label from V.

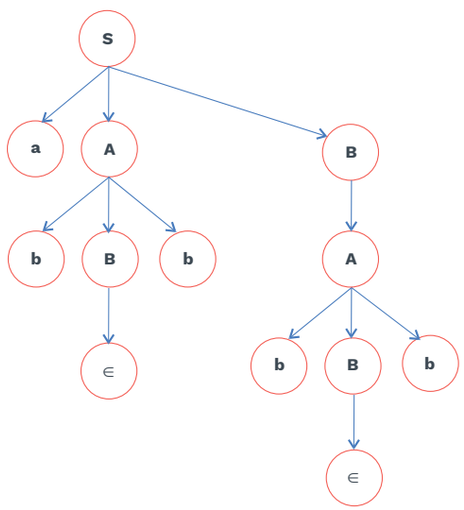
d) If a vertex has label A ∈ V, and its children are labelled (from left to right) a_1, a_2, ... a_n, then P mustcontain a production of the form.
$A \rightarrow a_1, a_2, \ldots\ldots\ldots a_n$

e) A leaf labeled ∈ has no siblings, that is a vertex with a child labeled ∈ can have no other children.

Example: consider the grammar G, with productions-

$S \rightarrow aAB$,
$A \rightarrow bBb$,
$B \rightarrow A|\in$

Parse tree for "abbbb":



Rack Your Brain

Which language is generated by grammer S → aSb|SS|∈

Previous Years' Question

A CFG G is given with the following productiions where S is the start symbol, A is non-terminal and a and b are terminals. **(GATE-IT-2008)**

S→aS|A

A→aAb|bAa|∈

Which of the following strings is generated by the grammar above?

1) aabbaba **2)** aabaaba

3) ababab **4)** aabbaab

Sol: Option 4)

Ambiguity in grammar:

When a grammar fails to provide a unique structure, it is sometimes possible to redesign the grammar to make the structure unique for each string in the language; but sometimes we cannot. That is known as ambiguity in grammar.

Ambiguous grammar:

Definition

A context free grammar G is said to be ambiguous if there exists some w∈L(G) that has atleast two distinct derivation trees (Parse trees).

Note:

Ambiguity implies the existence of two or more leftmost or rightmost derivations.

Example: E → E+E | E * E

Consider the sentential form E + E * E.

It has two derivations from E:

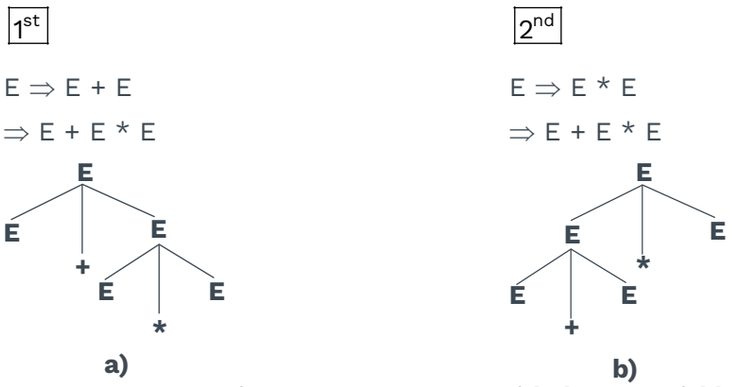


Fig. 4.1 Two Parse Trees with the Same Yield.

- Derivation 1st says that the second and third expressions are multiplied and the result is added to the first expression.
- Derivation 2nd adds the first two expressions and multiplies the result by the third.
  Example: 1+2*3
  1st derivation gives output = 1+(2*3)=7
  2nd derivation gives output = (1+2)*3 = 9
- Hence, we can say that there is an ambiguity.

Rack Your Brain

1) Find whether the following grammar is ambiguous or not?
$S \rightarrow aSbS|bSaS|\in$

2) Is it possible for a regular grammar to be ambiguous?

Causes of ambiguity in the grammar:
There are two causes of ambiguity in the grammar:
1) The precedence of operators is not respected.
Example: As explained in figure 4.1, both are valid parse trees, and ambiguity comes only because precedence is not followed. We need to force only the structure of figure 4.1 (a) to be legal in unambiguous grammar.
2) A sequence of identical operators can group either from the left or from the right.
Example: Two different parse tree exists for the string E+E+E. Since addition is associative, doesn't matter whether we group from left or right, but to eliminate ambiguity, we must pick one.

Example of an unambiguous grammar:

Example 1:	**Example 2:**	**Example 3:**
$E \rightarrow E + T$	$S \rightarrow aAB$	$X \rightarrow AB$
$T \rightarrow T * F$	$A \rightarrow bBb$	$A \rightarrow Aa\|a$
$F \rightarrow id$	$B \rightarrow A\|\in\Omega$	$B \rightarrow b$

Rack Your Brain

Is deterministic context free grammars is always unambiguous?

Grey Matter Alert!

Context free languages, generated only by ambiguous grammars, is known as inherently ambiguous.

Previous Years' Question

A CFG G is given with the following productions where S is the start symbol, A is non-terminal and a and b are terminals. **(GATE-IT-2008)**

S→aS|A

A → aAb|bAa|∈

For the string, "aabbaab" how many steps are required to derive the string and how many parse trees are there?

1) 6 and 1 **2)** 6 and 2

3) 7 and 2 **4)** 4 and2

Sol: Option 1)

Previous Years' Question

A context free grammar is ambiguous if: **GATE (CS)-1987)**

a) The grammar contains useless non-terminals

b) It produces more than one parse tree for some sentence.

c) Some production has two non-terminals side by side on the right-hand side.

d) None of the above

Sol: Option b)

Conversion of CFG into simplified CFG:

To get the Chomsky Normal Form, we need to make a number of preliminary simplifications, i.e. we need to convert CFG into simplified CFG.

i) We must eliminate $\in$-productions, those of the form $A \rightarrow \in$ for some variable A.

ii) We must eliminate unit productions, those of them from $A \rightarrow B$ for variables A and B.

iii) We must eliminate useless symbols, those variables, or terminals that do not appear in any derivation of a terminal string from the start symbol.

$\in$ productions:

Definition

Any production of a context free grammar of the form $A \rightarrow \in$ is called as $\in$-production

Note:

A variable is nullable if it can derive E directly or indirectly.

Eliminating $\in$ productions:

- A grammar may generate a language not containing $\in$, but can have some $\in$-production or nullable variables. In such a case, the $\in$-productions can be removed.

Example: $S \rightarrow ABC$
$A \rightarrow BC|a$
$B \rightarrow bAC|\in$
$C \rightarrow cAB|\in$
contains $\in$–productions.

Step 1: Find all the nullable variables or null variables. Remove all the null production from the grammar.

Step 2: Replace the right-hand side of the production with and without $\in$; wherever nullable variables or null variables are present on the RHS of production.

Step 3: After this, whatever grammar we get, that grammar is without $\in$- production.

Note:

If start symbol ($S \rightarrow \in$) contain $\in$ - production i.e.
if the language contain $\in$, then we cannot eliminate $\in$ - productions, totally.

Q1 **S → AbaC**
A → BC
B → b|∈
C → D|∈
D → d
Eliminate ∈- productions from the given grammar

Sol: **Step 1:** Nullable or null Variable = {B, C, A}
B contains ∈ production, so it is a nullable variable, similarly C is also a nullable variable.
A contains production A → BC, where B and C both are null variables. Thus A is also deriving epsilon. Hence A is also a nullable variable.

Remove B → ∈ , C → ∈ productions from the given grammar.

Step 2:
S → AbaC
S → Aba [Substitute C → ∈]
S → baC [Substitute A → ∈]
S → ba [Substitute A → ∈, C → ∈]

A → BC
A → B [Substitute C → ∈]
A → C [Substitute B → ∈]
B → b
C → D
D → d

Step 3: The final Grammar is:
S → AbaC | Aba | baC | ba
A → BC | B | C
B → b
C → D
D → d

Q2 **S → ABC**
A → BC|a
B → bAC|∈
C → cAB|∈
Eliminate ∈-productions from the given grammar.

Sol: **Step 1:** Nullable and null variables = {A,B,C}

B contains $\in$ production, so it is a nullable variable, similarly C is also nullable variable.

A contains production A → BC, where B and C both are null variables. Thus A is also deriving epsilon. Hence A is also a nullable variable.

Step 2:

S → ABC	
S → AB	[Substitute C → $\in$]
S → BC	[Substitute A → $\in$]
S → AC	[Substitute B → $\in$]
S → A	[Substitute B → $\in$, C → $\in$]
S → B	[Substitute A → $\in$, C → $\in$]
S → C	[Substitute A → $\in$, B → $\in$]
A → BC	
A → B	[Substitute C → $\in$]
A → C	[Substitute B → $\in$]
A → a	
B → bAC	
B → bA	[Substitute C → $\in$]
B → bC	[Substitute A → $\in$]
B → b	[Substitute A → $\in$, C → $\in$]
C → cAB	
C → cA	[Substitute B → $\in$]
C → cB	[Substitute A → $\in$]
C → c	[Substitute A → $\in$, B → $\in$]

Step 3: Final Grammar is:

S → ABC | AB | BC | AC | A | B | C

A → BC | B | C | a

B → bAC | bA | bC | b

C → cAB | cA | cB | c

Rack Your Brain

Find a context free grammar without $\in$-production equivalent to the grammar defined by:

S → ABaC

A → BC

B → b|$\in$

C → D|$\in$

D → d

Eliminating unit productions:

Definition

Unit Production: Any production of a context-free grammar of the form $A \to B$, where $A, B \in V$ (variables), is called a unit production.

To remove unit production, we use the substitution rule:
Step 1: Write the given grammar without unit production.
Step 2: Write the given grammar's all those production which goes to the unit production and further unit production produce terminals.
Step 3: Add all those new productions in the grammar obtained in step 1.

Note:
Meaning of Grammar should not be changed while doing elimination of unit production.

SOLVED EXAMPLES

Q1 $S \to Aa \mid B$
$B \to A \mid bb$
$A \to a \mid bc \mid B$
Eliminate unit-production from the given grammar.

Sol: **Step 1**: Grammar without unit production.
$S \to Aa$
$B \to bb$
$A \to a \mid bc$

Step 2: Productions which gives unit production:

$S \to B \to bb$
$S \to B \to A \to a$
$S \to B \to A \to bc$
} So add all these productions ir grammar obtained in step 1.

$B \to A \to a$
$B \to A \to bc$
} add these productions in grammar obtained in step 1.

$A \to B \to bb$ } add these productions in grammar obtained in step 1.

Step 3: Final grammar without unit production.

$S \rightarrow Aa \mid bb \mid a \mid bc$
$B \rightarrow bb \mid a \mid bc$
$A \rightarrow a \mid bc \mid bb$

Q2 $S \rightarrow AB$
$A \rightarrow a$
$B \rightarrow C|b$
$C \rightarrow D$
$D \rightarrow E$
$E \rightarrow a$
S, A, B, C, D, E E Variables and T = {a, b}
Eliminate Unit - Productions from the given grammar.

Sol:

Step 1: Productions without unit production.

$S \rightarrow AB$
$A \rightarrow a$
$B \rightarrow b$
$E \rightarrow a$

Step 2: Productions which give unit – production.

i) $B \rightarrow C \rightarrow D \rightarrow E \rightarrow a$
ii) $C \rightarrow D \rightarrow E \rightarrow a$
iii) $D \rightarrow E \rightarrow a$

} Add all this production in grammar obtained in step 1

Step 3: Grammar without unit productions

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

$C \rightarrow a$
$D \rightarrow a$
$E \rightarrow a$

} Here C, D, E are not reachable from the star So, these are useless symbols which will be removed.

Final Grammar will be:

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

Eliminating useless symbols:

- Symbol X is a useless symbol for a grammar G = (V,T,P,S) if that symbol can never take part in any derivation.

Example: $S \rightarrow aSb|\in|A$
$A \rightarrow aA$

- Here, the production $S \rightarrow A$ is useless as A cannot be transformed into a terminal string.
- Removing this production $S \rightarrow A$ leaves the language unaffected and is a simplification by any definition.

Note:
A variable is useful if it is reachable from the start state and is also able to derive some terminal or some strings of terminals.

SOLVED EXAMPLES

Q1 **$S \rightarrow A$**
$A \rightarrow aA \mid \in$
$B \rightarrow bA$
Eliminate useless symbol from the given grammar.

Sol: Here, we can remove $B \rightarrow bA$, since B is not reachable from the start state.
Final grammar is
$S \rightarrow A$
$A \rightarrow aA|\in$

Q2 **$S \rightarrow aAa|aBC$**
$A \rightarrow aS|bD$
$B \rightarrow aBa|b$
$C \rightarrow abb|DD$
$D \rightarrow aDa$
Eliminate useless symbol from the given grammar.

Sol:

- Initially, only B and C be the non-terminal symbols that directly generates terminal strings; hence useful symbol.
- S is useful because the rule S → aBC
- A is useful because the rule A → aS
- D does not generate terminal strings; thus, it is a useless symbol.
- Eliminate D and those productions involving D, we get grammar:
  $S \rightarrow aAa|aBC$

A → aS
B → aBa | b
C → abb

Q3 **S → AB|AC**
A → aAb|bAa|a
B → bbA|aaB|AB
C → abCA|aDb
D → bD|aC
Eliminate useless symbol from the given grammar.

Sol: **Another way**

- Terminals are always useful symbols. Since here, a and b are terminal. So, they are useful.
- Combination of terminal symbols generated by any rules then non-terminals also useful.
- As A → a, so, A is useful, B → bbA which produces B → bba so, B also useful.
- Similarly, S → AB is a combination of two useful symbols A and B. So, S is also useful.
  A useful symbols = {a, b, A, B, S}
- C and D must be removed because they are not producing terminals.
  So, remove S → AC, C → abCA | aDb, D→ bD | aC from the grammar.

Final grammar:
S → AB
A → aAb | bAa | a
B → bbA | aaB| AB
All are reachable from the start symbol S.

Rack Your Brain

What will be the final grammar after removing the useless symbols from the given grammar:
S → aS|A|C
A → a
B → aa
C → aCb

Theorem:
Let G = (V,T,S,P) be a context-free grammar. Then there exists an equivalent grammar $\hat{G} = (\hat{V}, \hat{T}, \hat{S}, \hat{P})$ that does not contain any useless variables or productions.

Note:
When we want to convert any CFG into an equivalent CFG that has no useless symbols, $\in$-productions and unit-productions. Some care must be taken in order of application of the consturction.
A safe order is:

i) Eliminate E-production.
ii) Eliminate unit production.
iii) Elminiate useless-symbols.

Rack Your Brain

Does removal of $\in$-productions introduces previously non-existent unit-productions?

Normal forms:
When working with context-free grammar, it is often convenient to have them in simplified form. There are two types of normal forms of context-free grammar.

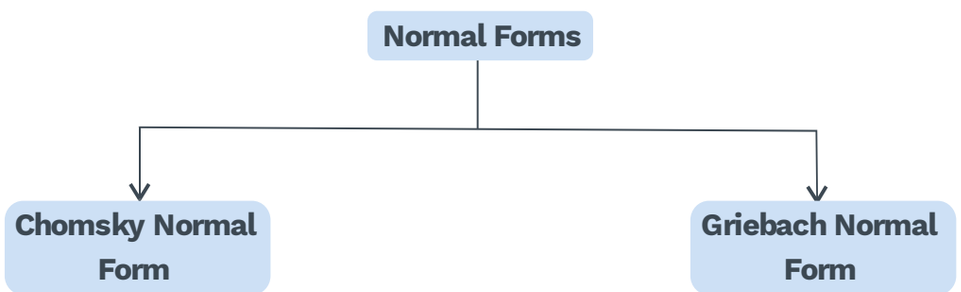


Fig. 4.2 Types of Normal Forms

Chomsky's normal form:

Definition

A context free grammar is in Chomsky Normal Form, if every rule is of the form:

$A \rightarrow BC$

OR

$A \rightarrow a$

where A, B, C are in V (variables) and a is any terminal.

Note:

In addition, we permit the rule $S \rightarrow \in$, where S is the start symbol in Chomsky Normal Form.

Example: $S \rightarrow AS \mid a$

$A \rightarrow SA \mid b$

is in Chomsky Normal Form.

Whereas the grammar:

$S \rightarrow AS \mid AAS$

$A \rightarrow SA \mid aa$

is not in Chomsky Normal Form; both productions $S \rightarrow AAS$ and $A \rightarrow aa$ violate the condition of the definition of Chomsky Normal Form.

Theorem:

Any context-free language is generated by a context-free grammar in Chomsky's normal form.

Advantages of Chomsky's normal form:

i) Length of each production is restricted.

ii) Derivation tree (parse tree) obtained from CNF is always a binary tree.

iii) The number of steps required to derive a string of length |w| is (2|w|-1).

Example: $S \rightarrow AB$

$A \rightarrow a$

$B \rightarrow b$

Let us have to derive the string w = ab.

So, |w| = length of string = 2.

$S \Rightarrow AB$

$\Rightarrow aB$

$\Rightarrow ab$

3 steps are needed. Thus a number of steps are required to derive a string 'ab' in the given grammar

= 2|w| – 1
= 2×2 – 1 = 3

iv) It is easy to apply the CYK membership algorithm, which is of the polynomial type having time complexity = $O(n^3)$

Note:

Given any Chomsky Normal Form Grammar, CYK membership algorithm can parse it.

Previous Years' Question

If G is a context free grammar and w is a string of length l in L(G), how long is a derivation of w in G, if G is in Chomsky Normal Form? **(GATE - 1992)**

1) 1) 2l **2)** 2l+1
3) 2l – 1 **4)** l

Sol: Option 3)

CYK algorithm:

This algorithm is only applicable if the given grammar is in CNF.
Check whether the string "baaba" is a valid member of the following CFG.

S → AB|BC
A → BA|a
B → CC|b
C → AB|a

	1	2	3	4	5
	b	a	a	b	a

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

Let's see how to fill the above table:

1) (1,1) → Starting with '1' ('b') and ending with '1' ('b')
'b' is generated by B.
Similarly we can compute for (2,2), (3,3), (4,4) (5,5)

2) (1,2) → Starts with 1('b) and ends with 2('a')
(1,2) = (1,1) (2,2)
= {B} {A,C}
= {BA, BC}

'BA' is generated by 'A'
'BC' is generated by 'S'

3) (2,3) = (2,2) (3,3)
= {A,C} {A,C}
= {AA, AC, CA, CC}
| | | |
X X X generated by 'B'

4) (3,4) = {3,3} {4,4}
= {A,C} {B}
= {AB, CB}
| |
generated X
by 'S' and 'C'

5) (4,5) = {4,4} {5,5}
= {B} {A,S}
= {BA, BC}
/ \
generated generated
by 'A' by 'S'

6) (1, 3) = = {1,1} {2,3} or {1,2} {3,3}
= {B} {B} = {A,S} {A,C}
= {BB} = {AA, AC, SA, SC}
| | | | |
X X X X X

$\therefore$ (1,3) = ϕ
Similarly, we can fill the other entries in the table.
(1,5) can be generated by 'S' (Starting symbol)
$\therefore$ baaba can be generated by grammar.

What is the substring of "baaba" that can be derived from this grammar?

From the table entries, we select the entries that have 'S' (starting symbol)

(1, 2) → 'S' can generate 'ba'
(2, 5) → 'S' can generate 'aaba'
(3, 4) → 'S' can generate 'ab'

Time complexity:
We are filling n^2 cells.
For each cell, in the worst case, we might need O(n) time.

$$\text{Total complexity} = O(n^2) \times O(n)$$
$$= O(n^3)$$

Greibach normal form:
A context-free grammar is in GNF (Greibach Normal Form) if all the productions are of the form:

$$\boxed{A \to a \propto}$$

where $\propto \in v^*$ (non-terminal)
$a \in T$ (Terminals), $A \in V$
Example: $A \to aBCDE$
$A \to a$

Note:

Every context free grammar can be transformed into an equivalent grammar in Greibach Normal Form.

Advantages of Greibach normal form:
1) The number of steps required to generate a string of length |w| is |w|.
Example: $A \to aB$
$B \to b$

Let we have to derive the strings w = 'ab', |w| = length of strings = 2
Step 1: $A \Rightarrow aB$
Step 2: $A \Rightarrow ab$

So, the number of steps required to derive the string w = 'ab' in the given form is 2
2) Greibach Normal Form (GNF) is useful in order to convert a CFG into PDA.

Converting context-free grammar to Chomsky's normal form:

SOLVED EXAMPLES

Q1 **Convert the grammar with productions:**
S → ABa,
A → aab,
B → Ac
to Chomsky Normal Form.

Sol: As required, the grammar does not have any $\in$-productions or any unit productions.

Step 1: Introduce new variables N_a, N_b, N_c and replace a with N_a, b with N_b, c with N_c.

$S \rightarrow ABN_a$
$A \rightarrow N_aN_aN_c$
$B \rightarrow AN_c$
$N_a \rightarrow a$
$N_b \rightarrow b$
$N_c \rightarrow c$

Step 2: Introduce additional variables to get the first two productions into Normal form:

$S \rightarrow AD_1$
$D_1 \rightarrow BN_a$
$A \rightarrow N_aD_2$
$D_2 \rightarrow N_aN_b$
$B \rightarrow AN_c$
$N_a \rightarrow a$
$N_b \rightarrow b$
$N_c \rightarrow c$

Note:

CNF produces the same language which is generated by CFG (Context Free Grammar).

Q2 **Convert the following grammar with production:**
$S \rightarrow bA \mid aB$
$A \rightarrow bAA \mid aS \mid a$
$B \rightarrow aBB \mid bs \mid b$
to Chomsky Normal Form.

Sol: As required, the grammar does not have any $\in$-productions or any unit productions.

Step 1: Introduce new variables N_a, N_b.

$S \rightarrow N_bA \mid N_aB$
$A \rightarrow N_bAA \mid N_aS \mid a$
$B \rightarrow N_aBB \mid N_bS \mid b$

$N_a \rightarrow a$
$N_b \rightarrow b$

Step 2: Introduce additional variables C_1 and C_2.

$S \rightarrow N_bA \mid N_aB$
$A \rightarrow N_bC_1 \mid N_aS \mid a$
$B \rightarrow N_aC_2 \mid N_bS \mid b$
$N_a \rightarrow a$
$N_b \rightarrow b$
$C_1 \rightarrow AA$
$C_2 \rightarrow BB$

Note:

For a given grammar, more than one CNF can be possible.

Grey Matter Alert!

Given a grammar G of length n, we can find an equivalent Chomsky Normal Form Grammar for G in time $O(n^2)$, the resulting grammar has length $O(n^2)$.

Converting context-free grammar to Greibach normal form conversion (CFG → GNF)

Variable → Terminal
Variable → Terminal variable
Variable → Terminal variable variable variable..

SOLVED EXAMPLES

Q1 **Convert the following context free grammar**
S → aSb | aB
B → b
to Griebach Normal Form

Sol: **In GNF:**

S → aSB | aB
B → b

Q2 **Convert the following context free grammar**

S → bAc | cB

A → b|c

B → c

where A, B, S ∈ variables and b, c ∈ terminal

to Griebach Normal Form

Sol:

In GNF:

S → bAB | cB

A → b | c

B → c

Q3 **Convert the following context free grammar**

S → mSn | mn

to Griebach Normal Form

Sol:

In GNF:

S → mSA | mA

A → n

Q4 **Convert the following context free grammar**

S → aBb | bCa

B → bBa | b

C → bCa | a

Sol:

In GNF:

S → aBX | bCY

B → bBY | b

C → bCY | a

X → b

Y → a

4.2 PUSHDOWN AUTOMATA

Definition

A pushdown Automata is a non-deterministic finite automaton coupled with a stack that can be used to store a string of arbitrary length. The stack can be read and modified only at its top.

(OR)

The automaton which defines context-free languages is known as pushdown Automata.

A Non-deterministic pushdown Acceptor (NPDA) is defined by the set tuples:

$$M\ (Q, \Sigma, \tau, \delta, q_0, Z, F)$$

Where,

Q : A finite set of states,

Σ : A finite set of input symbols.

τ : A finite stack alphabet.

(set of symbols that we are allowed to push into the stack)

$\delta : Q \times (\Sigma \cup \{\in\}) \times \tau \rightarrow$ set of finite subsets of $Q \times \tau^*$ is the transition function.

$q_0 \in Q$ is the initial state of the control unit.

$Z_0 : Z_0 \in \tau$ is the stack start symbol.

$F : F \subseteq Q$ is the set of final states.

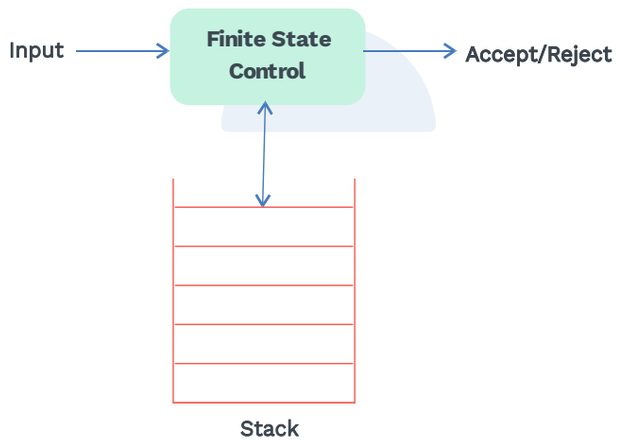


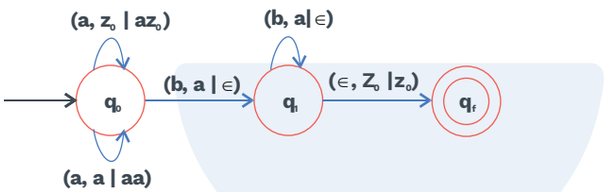
Fig. 4.3 Pushdown Automata

- Pushdown Automata is basically a finite Automaton with a stack.
- A pushdown Automata can write symbols on the stack and read them back later.
- Writing a symbol on the stack is referred to as pushing the symbol.
- Removing a symbol is referred to as popping it.

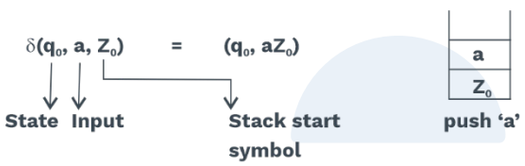
Grey Matter Alert!

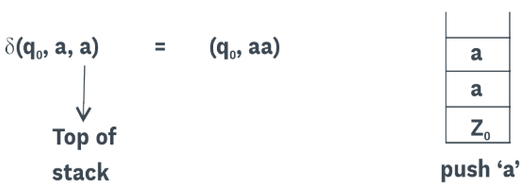
A stack is valuable part of Pushdown Automata because it can hold an unlimited amount of information.

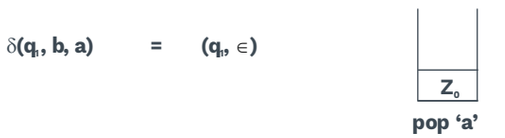
Example: L = $\{a^n b^n \mid n \geq 1\}$
We cannot construct the finite Automata corresponding to this language.
L is context-free language.
We can construct a push down Automata for Language L.



Consider a string w = aabb







$\delta(q_1, \in, Z_0)$ = $(q_2, \in)$ or (q_2, Z_0)

$(q_2, \in)$ ↓ Acceptance by empty stack

(q_2, Z_0) ↓ Acceptance by final state

Pushdown automata acceptance can be done in 2 ways:

1) acceptance by final state:

Let P = (Q, ∑, τ, δ, q_0, z_0, F) be a PDA.

Then language accepted by P is the set

$$L(P) = \{W \mid (q_0, w, z_0) \vdash_P^* (q_f, \in, s)\}$$

Where q_f is a final state and s is any stack string.

i.e. language accepted by P is the set of all strings that can put P into a final state at the end of the string.

2) Acceptance by empty stack:

An NPDA P = (Q, ∑, τ, δ, q_0, z_0, F) is said to accept the language N(P) by empty stack if,

$N(P) = \{W \mid (q_0, w, z_0) \vdash_P^* (q, \in, \in)\}$ for any state q.

That is, N(P) is a set of inputs W that P can consume, and when the end of the input string is reached, PDA has emptied the stack.

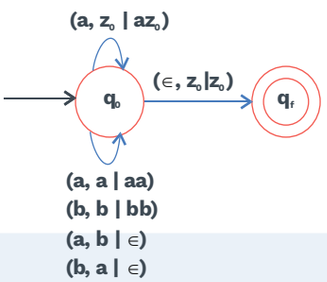
Deterministic and non deterministic pushdown automata:

- Deterministic pushdown Automata and Non-Deterministic pushdown differ in terms of their expressive power.
- Non-Deterministic pushdown automata are more powerful than Deterministic pushdown Automata.
- Transition function (δ) for Deterministic Pushdown Automata:
  $\delta: Q \times \{\sum \cup \in\} \times \tau \rightarrow Q \times \tau^*$
- Transition function (δ) for Non-Deterministic Pushdown Automata:
  $\delta : Q \times \{\sum \cup \in\} \times \tau \rightarrow$ set of finite subsets of $Q \times \tau^*$ is the transition function.

SOLVED EXAMPLES

Q1 **Construct Pushdown Automata for the language L = {w|$n_a(w) = n_b(w)$, w ∈ {a, b}*}.**

Sol:



Consider a string w = abbbaa
Here, number of a's in w = number of b's in w

$\delta(q_0, a, z_0) = (q_0, az_0)$

push 'a'

$\delta(q_0, b, a) = (q_0, \in)$

pop 'a'

$\delta(q_0, b, z_0) = (q_0, bz_0)$

push 'b'

$\delta(q_0, b, b) = (q_0, bb)$

push 'b'

$\delta(q_0, a, b) = (q_0, \in)$

pop 'b'

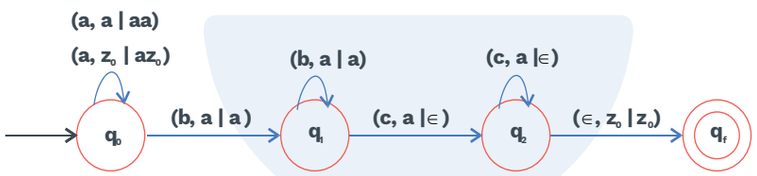
$\delta(q_0, a, b) = (q_0, \in)$

pop 'b'

$\delta(q_0, \in, z_0) = (q_f, z_0)$

Q2 **Construct a PDA for the language : L = {$a^n b^m c^n$ | n, m ≥ 1}**

Sol:

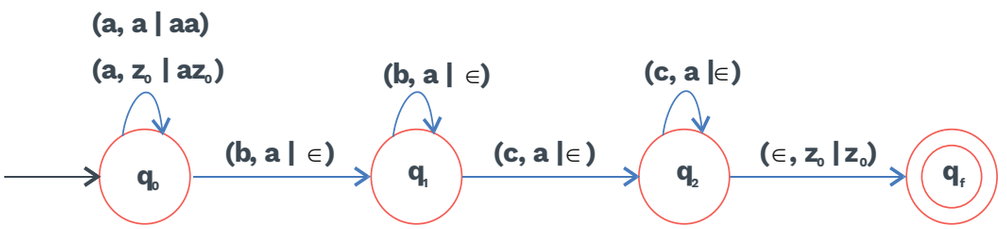


Consider a string w = aabcc

$\delta(q_0, a, z_0)$	=	(q_0, az_0)	push 'a' onto the stack
$\delta(q_0, a, a)$	=	(q_0, aa)	push 'a' onto the stack
$\delta(q_0, b, a)$	=	(q_1, a)	Do Nothing (skip)
$\delta(q_1, c, a)$	=	$(q_2, \in)$	pop 'a'
$\delta(q_2, c, a)$	=	$(q_2, \in)$	pop 'a'
$\delta(q_2, \in, z_0)$	=	(q_f, z_0)	

Q3 **Construct a PDA for the language : L = {$a^{m+n} b^m c^n$ | m, n ≥ 1}**

Sol:



Consider a string w = aaabbc

We can write L = $\{a^{m+n}b^mc^n \mid m, n \geq 1\}$

as L = $\{a^na^mb^mc^n \mid m, n \geq 1\}$

$\delta(q_0, a, z_0)$	=	(q_0, az_0)	push 'a' onto the stack
$\delta(q_0, a, a)$	=	(q_0, aa)	push 'a' onto the stack
$\delta(q_0, a, a)$	=	(q_0, aa)	push 'a' onto the stack
$\delta(q_0, b, a)$	=	$(q_1, \in)$	pop 'a'
$\delta(q_1, b, a)$	=	$(q_1, \in)$	pop 'a'
$\delta(q_1, c, a)$	=	$(q_2, \in)$	pop 'a'
$\delta(q_2, \in, z_0)$	=	(q_f, z_0)	

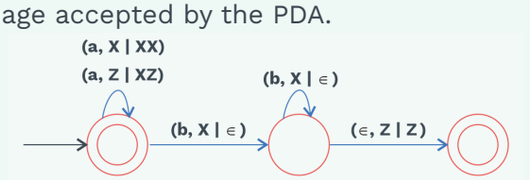
Rack Your Brain

Construct a PDA for the language: L = $\{a^{n+1}\ b^{2n} \mid n \geq O\}$

Previous Years' Question

Consider the transition diagram of a PDA given below with input alphabet ∑ = {a, b} and stack alphabet τ = {X, Z}, Z is the initial stack symbol.

Let L denote the language accepted by the PDA.



Which one of the following is TRUE? **(GATE 2016 (Set-1))**

1) L = $\{a^nb^n \mid n \geq O\}$ and is not accepted by any Finite Automata.

2) L = $\{a^n \mid n \geq O\} \cup \{a^nb^n \mid n \geq O\}$ and is not accepted by any Deterministic PDA.

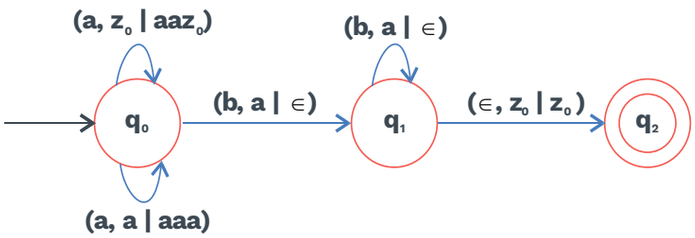
3) L is not accepted by any Turing Machine that halts on every input.

4) L = $\{a^n \mid n \geq O\} \cup \{a^nb^n \mid n \geq O\}$ and is deterministic context free.

Sol: Option 4)

Q4 **Construct a PDA for the language : $L = \{a^n b^{2n} \mid n \geq 1\}$**

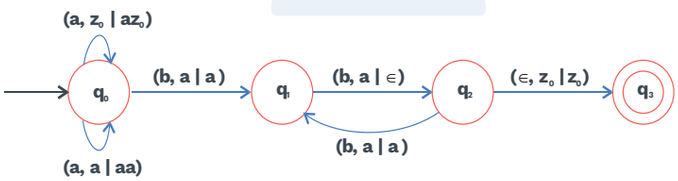
Sol: Push 2 a's onto stack on seeing 1 a in input



Consider a string w = aabbbb

$\delta(q_0, a, z_0)$	=	(q_0, aaz_0)	Push 2 a's on seeing 1 a in input
$\delta(q_0, a, a)$	=	(q_0, aaa)	Push 2 a's onto stack
$\delta(q_0, b, a)$	=	$(q_1, \in)$	Pop One a on input b
$\delta(q_1, b, a)$	=	$(q_1, \in)$	Pop One a on input b
$\delta(q_1, b, a)$	=	$(q_1, \in)$	Pop One a on input b
$\delta(q_1, b, a)$	=	$(q_1, \in)$	Pop One a on input b
$\delta(q_1, \in, z_0)$	=	(q_2, z_0)	

Method: 2



Consider a string w = aabbbb

$\delta(q_0, a, z_0)$	=	(q_0, az_0)	a z_0	Push 'a'
$\delta(q_0, a, a)$	=	(q_0, aa)	a a z_0	Push 'a'
$\delta(q_0, b, a)$	=	(q_1, a)	a a z_0	Do Nothing (Skip)

$\delta(q_1, b, a) = (q_2, \in)$ 

Pop 'a'

$\delta(q_2, b, a) = (q_1, a)$ 

Do Nothing (Skip)

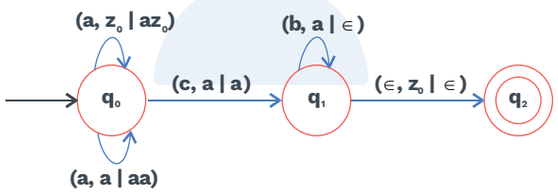
$\delta(q_1, b, a) = (q_2, \in)$ 

Pop 'a'

$\delta(q_2, \in, z_0) = (q_3, z_0)$

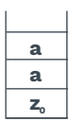
Q5 **Construct a PDA for the language : L = {$a^n cb^n$ | n ≥ 1}**

Sol: On input 'a' perform push operation onto the stack. On input 'c', skip (i.e., do nothing). On input 'b' and 'a' on top of stack, perform pop operations.



Consider a string w = aacbb.

$\delta(q_0, a, z_0) = (q_0, az_0)$ 

$\delta(q_0, a, a) = (q_0, aa)$ 

$\delta(q_0, c, a) = (q_1, a)$ 

Do Nothing

$\delta(q_1, b, a) = (q_1, \in)$ 

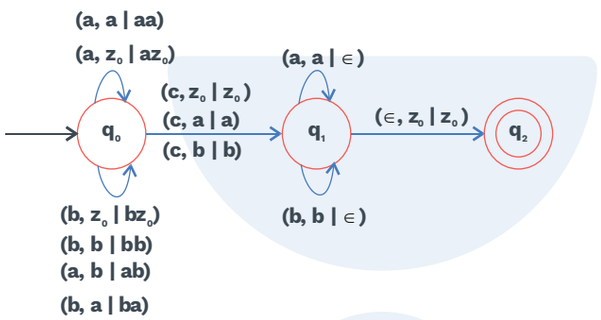
Pop 'a'

$\delta(q_1, b, a) =$ $(q_1, \in)$  Pop 'a'

$\delta(q_1, \in, z_0) = (q_2, z_0)$

Q6 **Construct PDA which accepts the languages $L = \{wcw^R \mid w \in \{a, b\}^*\}$.**

Sol:



Consider a string = $\frac{ab}{w}\frac{c}{c}\frac{ba}{w^R}$

$\delta(q_0, a, z_0) = (q_0, az_0)$  Push 'a'

$\delta(q_0, b, a) = (q_0, ba)$  Push 'b'

$\delta(q_0, c, b) = (q_1, b)$ Do Nothing

$\delta(q_1, b, b) = (q_1, \in)$  Pop 'b'

$\delta(q_1, a, a) = (q_1, \in)$  Pop 'a'

$\delta(q_1, \in, z_0) = (q_2, z_0)$

Previous Years' Question

Consider the Pushdown Automaton (PDA) below which runs over the input alphabet (a, b, c). It has stack alphabet $\{z_0, x\}$ where z_0 is the bottom-of-stack marker. The set of states of the PDA is (s, t, u, f) where s is the start state and f is the final state. The PDA accepts by final states. The transitions of the PDA given below are depicted in a standard manner. For example, the transition $(s, b, x) \rightarrow (t, xz_0)$ means that if the PDA is in state s and the symbol on the top of the stack is x, then it can read b from the input and move to state t after popping top of stack and pushing the symbols z_0 and x (in that order) on the stack. **(GATE IT 2006)**

$(s, a, z_0) \rightarrow (s, xxz_0)$

$(s, \in, z_0) \rightarrow (f, \in)$

$(s, a, x) \rightarrow (s, xxx)$

$(s, b, x) \rightarrow (t, \in)$

$(t, b, x) \rightarrow (t, \in)$

$(t, c, x) \rightarrow (u, \in)$

$(u, c, x) \rightarrow (u, \in)$

$(u, \in, z_0) \rightarrow (f, \in)$

The language accepted by the PDA is:

1) $\{a^l b^m c^n \mid l = m = n\}$

2) $\{a^l b^m c^n \mid l = m\}$

3) $\{a^l b^m c^n \mid 2l = m + n\}$

4) $\{a^l b^m c^n \mid m = n\}$

Sol: Option 3)

Rack Your Brain

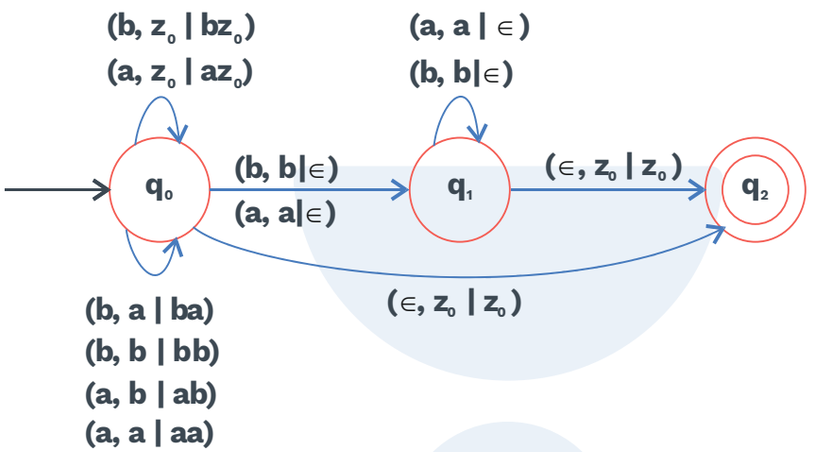
Construct PDA that accepts the following language on $\Sigma = \{a, b, c\}$

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

Q7 **Construct NPDA's that accepts the languages L = {$ww^R | w \in \{a, b\}^*$}.**

Sol: We cannot construct deterministic pushdown Automata for the given language L = {$ww^R | w \in \{a, b\}^*$}.

Here Non-determinism comes into picture since on same state, on same input more than one transition will be defined.



$\delta(q_0, a, z_0) = (q_0, az_0)$

$\delta(q_0, b, z_0) = (q_0, bz_0)$

$\delta(q_0, a, a) = (q_0, aa)$ or $(q_1, \in)$

$\delta(q_0, b, b) = (q_0, bb)$ or $(q_1, \in)$

$\delta(q_0, a, b) = (q_0, ab)$

$\delta(q_0, b, a) = (q_0, ba)$

$\delta(q_1, \in, z_0) = (q_2, z_0)$

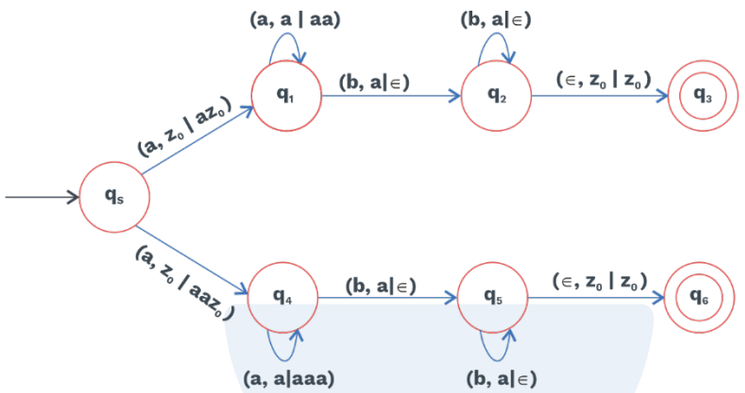
$\delta(q_0, \in, z_0) = (q_2, z_0)$

$\delta(q_1, a, a) = (q_1, \in)$

$\delta(q_1, b, b) = (q_1, \in)$

Q.8 **Construct NPDA for the language : $L = \{a^n b^n \mid n \geq 1\} \cup \{a^n b^{2n} \mid n \geq 1\}$**

Sol:



Difference between DPDA and NPDA:

Deterministic pushdown automata	Non-deterministic pushdown automata
1) Transition function for deterministic pushdown automata: $\delta : Q \times \{\Sigma \cup \in\} \times \tau \rightarrow Q \times \tau^*$	**1)** Transition function for non-deterministic pushdown Automata: $\delta : Q \times \{\Sigma \cup \in\} \times \tau \rightarrow$ set of finite subsets of $Q \times \tau^*$
2) Only one transition is defined from a state on an input symbol and stack symbol.	**2)** More than one transition can be defined from a state on an input symbol and stack symbol.
3) DPDA is less powerful than NPDA.	**3)** NPDA is more powerful than DPDA.

Table 4.1 Differences

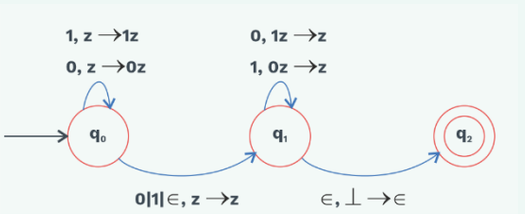
Rack Your Brain

Construct NPDA's that accepts the language $L = \{a^n b^m \mid n \geq 0, n \neq m\}$.

Previous Years' Question

Consider the NPDA < Q = $\{q_0, q_1, q_2\}$, $\Sigma = \{0, 1\}$, $\tau = \{0, 1, \perp\}$, δ, q_0, $\perp$, F = $\{q_2\}$ >
Where (as per usual convention) Q is the set of states, Σ is the input alphabet, τ is the stack alphabet, δ is the state transition function, q_0 is the initial state, $\perp$ is the initial stack symbol, and F is the set of accepting states. The state transition is as follow:

(GATE 2015 (Set-1))



Which one of the following sequences must follow the string 101100 so that the overall string is accepted by the automaton?

1) 10110 **2)** 10010

3) 01010 **4)** 01001

Sol: Option 2)

Rack Your Brain

Construct NPDA on $\Sigma = \{a, b, c\}$ that accepts the language
$L = \{w_1 c w_2 \mid w_1, w_2 \in \{a, b\}^*, w_1 \neq w_2\}$.

Equivalence between CFL and CFG

1) $L = \{a^m b^n \mid m = n\}$
Grammar corresponding to given CFL:
$S \rightarrow aSb|\in$

2) $L = \{a^m b^n \mid m = 2n\}$
Grammar Corresponding to given CFL:
$S \rightarrow aaSb|\in$

3) $L = \{a^m b^n c^n \mid m, n \geq 1\}$
Grammar Corresponding to given CFL:
$S \rightarrow AB$
$A \rightarrow a|aA$
$B \rightarrow bBc|bc$

4) $L = \{a^n b^p c^n \mid p, n \geq 1\}$
Grammar Corresponding to given CFL:
$S \rightarrow aSc|aAc$
$A \rightarrow bB|b$

5) $L = \{a^m b^n \mid m > n;\ m,n \geq 1\}$
Grammar Corresponding to given CFL:
$S \rightarrow aSb|aAb$
$A \rightarrow aA|a$

6) $L = \{a^m b^n \mid m \neq n;\ m,n \geq 1\}$
Cases: $m > n$ or $m < n$
$S \rightarrow S_1 \mid S_2$
$S_1 \rightarrow aS_1b|aAb$
$A \rightarrow aA \mid a$
$S_2 \rightarrow aS_2b|aBb$
$B \rightarrow bB|b$

7) $L = \{w| \ |w_a| = |w_b|, w \in \{a, b\}^*\}$ (number of a's and number of b's should be equal)
Grammar Corresponding to given CFL:
$S \rightarrow SaSbS \mid SbSaS \mid \in$

Closure properties of CFL's

The context-free languages are closed under:

- **Union:**
  Let L_1 and L_2 be CFL's, then $L_1 \cup L_2$ is also context-free language.
- **Concatenation:**
  Let L_1 and L_2 be CFL's, then $L_1.L_2$ is also context-free language.
- **Kleen-closure:**
  If L_1 is a CFL, then L_1^* is also context-free language.
- **Reversal:**
  The CFL's are also closed under reversal.
  If L is a CFL, then so is L^R.
- The family of context-free languages is not closed under intersection, set difference, and complementation.
- If CFL is not closed under intersection means there exist two CFL languages for which $CFL \cap CFL_2 \neq CFL$.

Example: Consider the two languages:

$L_1 = \{a^n b^n c^m \mid n \geq 0,\ m \geq 0\}$ and $L_2 = \{a^n b^m c^m \mid n \geq 0,\ m \geq 0\}$, then
$L_1 \cap L_2 = \{a^n b^n c^n \mid n \geq 0\}$ which is not context-free.

Thus, context-free language is not closed under intersections.

Again, $L_1 \cap L_2 = \overline{\overline{L_1} \cup \overline{L_2}}$

If the family of context-free languages was closed under complementation, then the right side of the above expression would be a context-free language for any context-free L_1 and L_2.

But this contradicts since we have shown, the intersection of two context-free languages is not necessarily context-free. Thus, the family of context-free languages is not closed under complementation.

Note:

If L is a CFL and R is a regular language, then $L \cap R$ is a context free language.

Rack Your Brain

Is the language $L = \{a^n b^n | \ n \geq 0, n \neq 100\}$ is context free?

Previous Years' Question

Let L_1 be a regular language and L_2 be a context free language. Which of the following languages is/are context free? **(GATE 2021 (Set-2))**

1) $L_1 \cap \overline{L}_2$

2) $\overline{\overline{L}_1 \cup \overline{L}_2}$

3) $L_1 \cup (L_2 \cup \overline{L}_2)$

4) $(L_1 \cap L_2) \cup (\overline{L}_1 \cap L_2)$

Sol: Option 2) 3), 4)

Previous Years' Question

Let L_1, L_2 be any two context free languages and R be any regular language, then which of the following is/are **CORRECT**? **(GATE 2017 (Set-2))**

I) $L_1 \cup L_2$ is context free.

II) $\overline{L}_1$ is context free.

III) $L_1 - R$ is context free.

IV) $L_1 \cap L_2$ is context free.

1) I, II and IV only **2)** I and III only

3) II and IV only **4)** I only

Sol: Option 2)

- CFLs are closed with the following operations:
  - **1)** Union
  - **2)** Concatenation
  - **3)** Kleene closure
  - **4)** Substitution
  - **5)** Homomorphism
  - **6)** Inverse Homomorphism
  - **7)** Reversal
  - **8)** Intersection with a regular set
  - **9)** INIT (L)
- CFL are not closed under the following operations:
  - **i)** Intersection
  - **ii)** Complement
  - **iii)** Set difference
  - **iv)** Quotient
  - **v)** Inverse substitution

Deterministic Context-Free Languages (DCFLs)

Definition

The language that are recognized by deterministic pushdown Automata (DPDAs) are called as Deterministic Context free languages (DCFLs).

- This subclass of context-free languages is relevant to practical applications, such as the design of parsers in compilers for programming languages.

- In DPDAs, at each step of its computation, the DPDA has at most one way to proceed according to its transition function.

Example: $L = \{0^n1^n \mid n \geq 0\}$ is DCFL.

Note:

Every language accepted by DPDA is also accepted by NPDA but converse need not to be true.

Closure properties of DCFL

- The class of DCFLs is closed under complementation.
- The class of DCFLs is closed under Intersection with Regular set.
  If L_1 = DCFL and L_2 = Regular language

  Then $L_1 \cap L_2$ is DCFL
- The class of DCFLs is closed under Inverse Homomorphism.
- DCFLs are not closed under the following operations:
  1) Union
  2) Concatenation
  3) Kleene closure
  4) Homomorphism
  5) Intersection
  6) Substitution
  7) Reversal
  8) $\in$-free Homomorphism

Rack Your Brain

Is the language $L = \{a^nb^m | m > n + 2\}$ is deterministic?

Grey Matter Alert!

- If a Grammar G is an LL(K) grammar, then L(G) is a deterministic context free language.
- A grammar is an LL(K) grammar if we can uniquely identify the correct production, given the currently scanned symbol and a 'look-ahead' of the next K-1 symbols.

Previous Years' Question

If L_1 and L_2 are context free languages and R a regular set, one of the languages below is not necessarily a context free language? **(GATE -1996)**

Which one?

1) $L_1.L_2$ 2) $L_1 \cap L_2$ 3) $L_1 \cap R$ 4) $L_1 \cup L_2$

Sol: Option 2)

Previous Years' Question

Let L be a context free language and M a regular language. Then the language $L \cap M$ is: **(GATE-2006)**

1) always regular

2) never regular

3) always a deterministic context free language

4) always a context free language

Sol: Option 4)

Decision properties of CFL's

The following problems are decidable under CFL's:

1) Emptiness problem

2) Finiteness problem

3) Membership problem

Emptiness problem:

Given a context-free grammar G = (V, T, S, P), there exists an algorithm for deciding whether L (G) is empty or not; hence this is a decidable problem.

E_{CFG} = { <G> | G is a CFG and L(G) = ϕ }

Finiteness problem:

Given a context-free grammar G = (V, T, S, P), there exists an algorithm for determining whether L(G) is infinite or not.

Membership problem:

Given a context-free grammar G = (V, T, S, P), there exists an algorithm, for deciding whether a given word w is in the language defined by a context-free grammar.

Proof:

- Start by converting whatever representation of L is given into a CNF grammar for L.
- As the parse trees of a Chomsky-normal form grammar are binary trees, if w is of length n then there will be exactly 2n-1 nodes labelled by variables in the tree.
- The number of possible trees and node-labellings is thus only exponential in n. List them all and check if any of them yields w.

Note:

There is much more efficient technique based on the idea of dynamic programming, known as table filling algorithm or tabulation.
This algorithm is CYK algorithm; used to test membership in a context free language.

Grey Matter Alert!

The Cocke-Younger-Kasami (CYK) algorithm tells whether a given string is in a given context free language. For a fixed CFL, this test takes time $O(n^3)$, if n is the length of the string being tested.

Undecidable CFL problems

1) Is a given CFG G ambiguous?
2) Is a given CFL inherently ambiguous?
3) Is the intersection of two CFL's empty?
4) Are two CFL's the same?
Is a given CFL equal to Σ^*, where Σ is the alphabet of this language?

Rack Your Brain

Does L(G) contain atleast 100 strings, for a given CFG G?
Is the given problem is decidable?

Previous Years' Question

Which of the following problems is undecidable? **(GATE 2014 (Set 3))**

1) Deciding if a given context free grammar is ambiguous.

2) Deciding if a given string is generated by a given context free grammar.

3) Deciding if the language generated by a given context free grammar is empty.

4) Deciding if the language generated by a given context free grammar is finite.

Sol: Option 1)

Rack Your Brain

Which of the following problem is undecidable?

a) Completeness problem for CFGs

b) Equivalence problem for CFGs

Pumping lemma for context-free language:

Let L be an infinite context-free language. Then there exists some positive integer m such that any $w \in L$ with $|w| \geq m$ can be decomposed as

$w = uvxyz,$

with $|vxy| \leq m,$

and $|vy| \geq 1$

Such that

$uv^i xy^i z \in L$

for all i = 0, 1, 2, ...

This is known as the pumping lemma for context-free languages.

Note:

Pumping lemma for context free language are used negatively to show that a given language does not belong to context free language family.

Show that the language:

Example: $L = \{a^n b^n c^n \mid n \geq 0\}$ is not context-free language.

Sol.: Let's assume L is context-free language

$\exists\, m \geq 1$

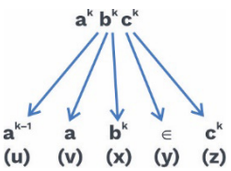
such that

$w \in L$ with $|w| \geq m$

$w = a^k b^k c^k$

Here $3k \geq m$

Now, split $a^k b^k c^k$ in uvxyz.



$|vxy| = k + 1 \leq m$

Hence, $uv^ixy^iz \in L \ \forall \ i = 0, 1, 2, \ldots$

for $i = 2$, $a^{k-1} a^2 b^k \in c^k$

$\Rightarrow$ $a^{k+1} b^k c^k$ which does not belong to given language

$\therefore$ $L = \{a^n b^n c^n \mid n \geq 0\}$ is not context-free language.

Note:

If $\Sigma = \{a\}$ i.e. singleton set and language L is defined over L then Lis context free language if and only if the length of strings in L are in arithmetic progression.

eg. 1: $L = \{a^{3n-1} \mid n \geq 1\}$
$=\{aaaa, aaaaaaa, \ldots\ldots\}$
context free language. **(Following arithmetic progression)**

eg. 2: $L = \{a^p \mid p$ is a prime$\}$
$=\{aa, aaa, aaaaa, \ldots\ldots\}$
Not context free language. **(Not following arithmetic progression)**

eg. 3: $L = \{a^{n^2} \mid n \geq 1\}$
$=\{a, aaaa, \ldots\ldots\}$
Not context free languages **(Not following arithmetic progression)**

eg.4: $L = \{a^{n!} \mid n \geq 1\}$
$=\{a, aa, aaaaaa, \ldots\ldots\}$
Not context free langauge **(Not following arithmetic progression)**

4.3 CSL AND GRAMMAR

Context-sensitive grammar:

Definition

A grammar G = (V, T, S, P) is said to be context sensitive if all productions are of the form,

$$x \rightarrow y,$$

where $x, y \in (VUT)^*$ and $|x| \geq |y|$

Grey Matter Alert!

The definition shows clearly that this grammar is non contracting in the sense that the length of successive sentential forms can never decrease.
All such grammars can be rewritten in a normal form in which all productions are of the form:

$xAy \rightarrow xvy$

This is equivalent to saying that the production

$A \rightarrow v$

can be applied only in the situation where A occurs in a context of the string x on the left and the stringy on the right.

eg. : $S \rightarrow abc/aAbc$

$Ab \rightarrow bA$

$Ac \rightarrow Bbcc$

$bB \rightarrow Bb$

$aB \rightarrow aa/aaA$ is an example of context sensitive grammar.

Context-sensitive languages:

The language generated by the context-sensitive grammar is known as context-sensitive languages.

Definition

A language L is said to be context sensitive if there exists a context sensitive grammar G such that

$L = L(G)$ or, $L = L(G) \cup \{\in\}$

Here, in this definition, we reintroduce the empty string:

- Definition of context-sensitive grammar implies that $x \rightarrow \in$ is not

allowed so that a context-sensitive grammar can never generate a language containing the empty string.

- Every context-free language without $\in$ can be generated by a special case of a context-sensitive grammar, say by one in Chomsky or Greibach's normal form. Both satisfy the condition of context-free grammar defined earlier.

Note:

By including the empty string in the definition of a context sensitive language. We can claim that the family of context free language is a subset of the family of context sensitive languages.

Example 1: The language $L = \{a^n b^n c^n \mid n \geq 1\}$ is a Context-Sensitive language.

Context-Sensitive grammar exists for this language is:

$S \rightarrow abc|aAbc$
$Ab \rightarrow bA$
$Ac \rightarrow Bbcc$
$bB \rightarrow Bb$
$aB \rightarrow aa|aaA$

Look at a derivation of $a^3b^3c^3$.

$S \Rightarrow aAbc$
$\Rightarrow abAc$
$\Rightarrow abBbcc$
$\Rightarrow aBbbcc$
$\Rightarrow aaAbbcc$
$\Rightarrow aabAbcc$
$\Rightarrow aabbAcc$
$\Rightarrow aabbBbccc$
$\Rightarrow aabBbbccc$
$\Rightarrow aaBbbbccc$
$\Rightarrow aaabbbccc$

Note:

The language in the previous example is not context free, we can say that the family of context free languages is a proper subset of the family of context sensitive languages.

Example 2 : The language $L=\{a^n b^m c^n d^m \mid n,m \geq 1\}$ is a Context-Sensitive language. Context-Sensitive grammar exists for this language is:

S → aAB | aB
A → aAX | aX
B → bBd | bYd
Xb → bX
XY → Yc
Y → c

Derivation of string aabbbccddd as follows:

S ⇒ aAB
⇒ aaXB
⇒ aaXbBd
⇒ aaXbbBdd
⇒ aaXbbbYddd
⇒ aabXbbYddd
⇒ aabbXbYddd
⇒ aabbbXYddd
⇒ aabbbYcddd
⇒ aabbbccddd

Example of Context-Sensitive languages

$L_1 = \{a^n b^n c^n \mid n \geq 1\}$
$L_2 = \{a^{n!} \mid n \geq 0\}$
$L_3 = \{a^n \mid n = m^2, m \geq 1\}$
$L_4 = \{a^n \mid n \text{ is a prime number}\}$
$L_5 = \{a^n \mid n \text{ is not a prime number}\}$
$L_6 = \{ww \mid w \in (a,b)^+\}$
$L_7 = \{w^n \mid w \in (a,b)^+, n \geq 1\}$
$L_8 = \{www^R \mid w \in \{a,b\}^+\}$

Rack Your Brain

Find context sensitive grammars for the languages:

a) $L = \{w \mid n_a(w) = n_b(w) = n_c(w)\}$

b) $L = \{w \mid n_a(w) = n_b(w) < n_c(w)\}$

Theorem:
For every Context-Sensitive language L not including $\in$, there exists some linear bounded automaton M such that L = L(M).

Theorem:
If a language L is accepted by some linear bounded automaton M, then there exists a Context-Sensitive grammar that generates L.

Relation between recursively enumerable and context-sensitive languages:
Every Context-Sensitive language is accepted by some Turing machine and is therefore recursively enumerable.

Theorem:
Every Context-Sensitive language is recursive.

Theorem:
There exists a recursive language that is not Context-Sensitive.

Note:

- There exists a recursive language that is not context-sensitive. This theorem indicates that linear bounded automata are indeed less powerful than Turing Machines. Since, they accept only a proper subset of the recursive languages.
- It follows from the same result that linear bounded automata are more powerful than Pushdown Automata.

Note:

- Context free languages, being generated by context free grammars, are a subset of the context-sensitive languages.
- Any language accepted by a Pushdown Automata is also accepted by some linear bounded automaton but there are languages accepted by some linear bounded automata for which there are no Pushdown Automata.

Previous Years' Question

Which one of the following language over $\Sigma = \{a, b\}$ is **NOT** context free?

(GATE (CSE) -2019)

1) $\{ww^R \mid W \in \{a, b\}^*\}$

2) $\{wa^n b^n w^R \mid w \in \{a, b\}^*, n \geq 0\}$

3) $\{wa^n w^R b^n \mid w \in \{a, b\}^*, n \geq 0\}$

4) $\{a^n b^i \mid i \in \{n, 3n, 5n\}, n \geq 0\}$

Sol: Option 3)

Closure properties of context-sensitive language:

The family of Context-Sensitive languages is closed under

1) Union.
2) Intersection.
3) Complementation.
4) Concatenation
5) Kleene star.
6) Reversal.
7) Inverse Homomorphism.

Note:

The family of context sensitive language is not closed under Homomorphism.

Closure properties of DCFL, CFL and CSL:

Property	DCFL	CFL	CSL
Union	No	Yes	Yes
Intersection	No	No	Yes
Set-difference	No	No	Yes
Complement	Yes	No	Yes
Concatenation	No	Yes	Yes

Kleene closure	No	Yes	Yes
Homomorphism	No	Yes	No
Reversal	No	Yes	Yes
Substitution (∈-free)	No	Yes	Yes
Inverse homomorphism	Yes	Yes	Yes
Intersection with a regular language	Yes	Yes	Yes

Table 4.2 Closure Properties

- **Context-free grammar:** A grammar G = (V, T, S, P) is said to be context-free if all productions in P have the form:

$$A \rightarrow x$$

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

- **Derivation of string:** Left Most derivation and Right Most Derivation.
- A derivation is said to be left most if in each step, the leftmost variable in the sentential form is replaced.
- A derivation is said to be right most if in each step, the rightmost variable in the sentential form is replaced.
- **Ambiguity in grammar:** A Context-Free Grammar G is said to be ambiguous if there exists some $w \in L(G)$ that has at least two distinct derivative trees (parse tree).

Conversion of CFG into simplified context-free grammar:

A safe order is:

i) Eliminate ∈ - productions
ii) Eliminate Unit - productions
iii) Eliminate useless symbol

- **Chomsky normal form:** A Context-Free Grammar is in Chomsky Normal Form if every rule is of the form: $A \rightarrow BC$ (OR) $A \rightarrow a$
where A, B, C are in variables and a is any terminal.
- We can convert Context-Free Grammar to Chomsky Normal Form as well as to Greibach Normal Form.
- **Pushdown automata:** The automaton which defines the Context-Free languages is known as Pushdown Automata.

Chapter Summary

- **Deterministic and non-deterministic pushdown automata:** Non-Deterministic Pushdown Automata is more powerful than Deterministic Pushdown Automata.
- **Closure properties of CFLs:** The Context-Free Languages are closed under: **1)** Union **2)** Concatenation **3)** Kleene-Closure **4)** Reversal.
- **Decision properties of CFLs:** Following problems are decidable under CFLs: Emptiness problem, Finiteness problem, Membership problem
- **Context-sensitive languages:** Language generated by Context-Sensitive Grammar is known as Context-Sensitive Languages.

5 Turing Machine

5.1 BASICS OF TURING MACHINE

Introduction:

- A Turing machine is a control unit with a finite number of states and unbounded tape(s).
- Although, the Turing machine seems to be very simple from the structural point of view, it is as powerful as simple it looks.
- There are so many problems that are unsolvable by push down automaton. Such problems are solved using a Turing Machine.
- According to the Turing thesis, the only general type of automaton whose power almost matches with a computer is nothing but the Turing machine.

Standard turing machine:

- Turing machine uses the tape as a buffer, in other words, we can say that it is a temporary storage area.
- There is a read-write head which is a part of the Turing machine, and it is associated with the tape to read and write one symbol at a time onto the tape at the time of traversing the tape in the right or left direction.

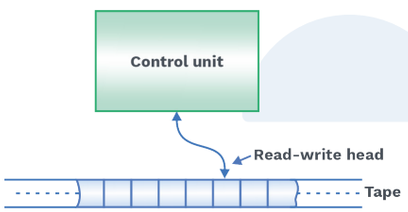


Fig. 5.1 Diagram of a Turing Machine

If M is a Turing machine, then the definition of M should be
$M = (Q, \Sigma, \Gamma, \delta, q_0, B, F)$

Where
Q is the set of finite states,
Σ is the set of input alphabets,
Γ is a finite set of symbols called the tape alphabet,
δ is the transition function, defined as
$\delta: Q \times \Gamma \rightarrow Q \times \Gamma \times \{L, R\}$
where L = left movement & R = right movement,
$B \in \Gamma$ is a special symbol called the blank,
$q_0 \in Q$ is the initial state,
$F \subseteq Q$ is the set of final states.

Note: In the definition of a Turing machine, we assume that E $\Sigma \subseteq \Gamma$- {B}, that the input alphabet is a subset of the tape alphabet, not including the blank.

- The interpretation of the transition function δ on Q × Γ states the working principle of the Turing machine.
- The present state and the recent tape symbol, which is being read, are passed as the arguments of the transition function.
- As a result, the control of the Turing machine will move to a new state, the previous tape symbol may be changed with a new one, and the read write head will be shifted to the left or to the right.
- The shift or move symbol will decide about the movement of the read write head after the write of the new symbol in the place of the old one, whether to shift left or right by one cell.

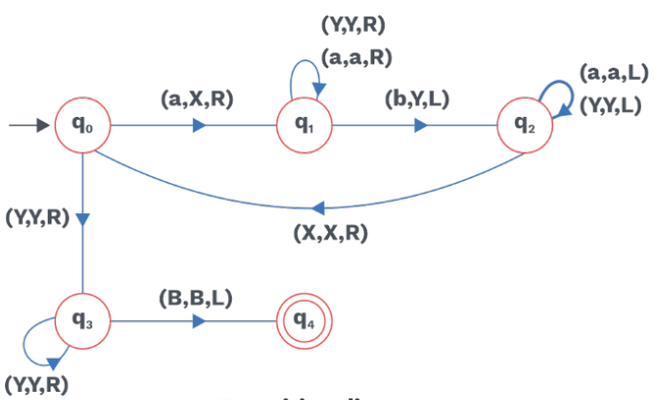
Note: Turing machine can't accept epsilon.

Turing machine as acceptor:

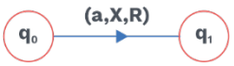
SOLVED EXAMPLES

Q1 **Design a TM for L = $a^n b^n$ / n ≥ 1.**

Sol:



Transition diagram

The transition 

means q_0 by getting 'a'; it is replacing 'a' with 'X' & going to state q_1 and read-write head move one cell right.

We can even give transition table for $L = a^n b^n / n \geq 1$ as shows below:

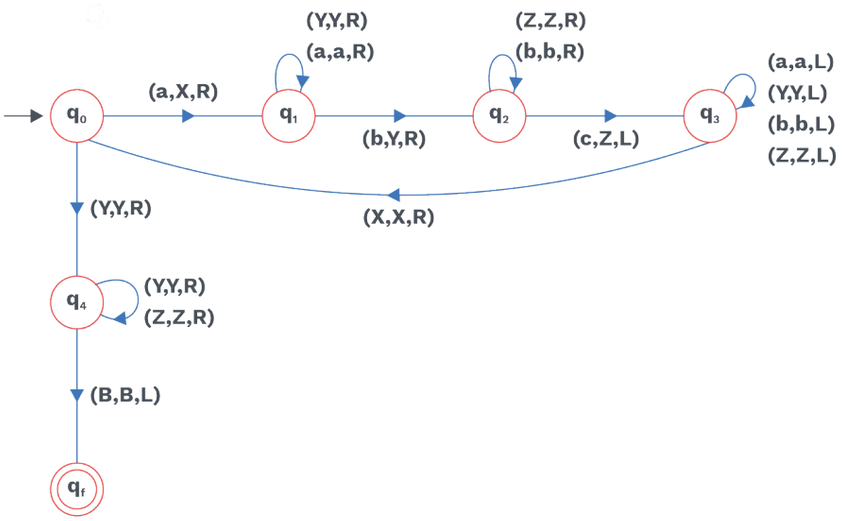
$\delta(q_0, a) \rightarrow (q_1, X, R)$
$\delta(q_0, Y) \rightarrow (q_3, Y, R)$
$\delta(q_1, a) \rightarrow (q_1, a, R)$
$\delta(q_1, Y) \rightarrow (q_1, Y, R)$
$\delta(q_1, b) \rightarrow (q_2, Y, L)$
$\delta(q_2, a) \rightarrow (q_2, a, L)$
$\delta(q_2, Y) \rightarrow (q_2, Y, L)$
$\delta(q_2, X) \rightarrow (q_0, X, R)$
$\delta(q_3, Y) \rightarrow (q_3, Y, R)$
$\delta(q_3, B) \rightarrow (q_4, B, L)$

If the string will be in language then it will get accepted and halt at state q_4 otherwise it will either go dead state or dead configuration.

Note: The transition table and transition diagram both are equivalent in power.

Q2 Design a Turing machine for $L = \{a^n b^n c^n / n \geq 1\}$.

Sol:



Transition diagram

The transition table for $L = \{a^n b^n c^n \mid n \geq 1\}$ is given below:

$\delta(q_0, a) \rightarrow (q_1, X, R)$

$\delta(q_0, Y) \rightarrow (q_4, Y, R)$
$\delta(q_4, Y) \rightarrow (q_4, Y, R)$
$\delta(q_4, Z) \rightarrow (q_4, Z, R)$
$\delta(q_4, B) \rightarrow (q_f, B, L)$
$\delta(q_1, b) \rightarrow (q_2, Y, R)$
$\delta(q_1, a) \rightarrow (q_1, a, R)$
$\delta(q_1, Y) \rightarrow (q_1, Y, R)$
$\delta(q_2, b) \rightarrow (q_2, b, R)$
$\delta(q_2, Z) \rightarrow (q_2, Z, R)$
$\delta(q_2, c) \rightarrow (q_3, Z, L)$
$\delta(q_3, b) \rightarrow (q_3, b, L)$
$\delta(q_3, Z) \rightarrow (q_3, Z, L)$
$\delta(q_3, Y) \rightarrow (q_3, Y, L)$
$\delta(q_3, a) \rightarrow (q_3, a, L)$
$\delta(q_3, X) \rightarrow (q_0, X, R)$

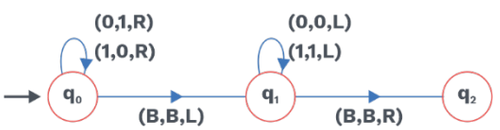
Transducer:
This is a more general automaton,which is able to produce strings of symbols as output and it is called a transducer.

Turing machine as transducer:

SOLVED EXAMPLES

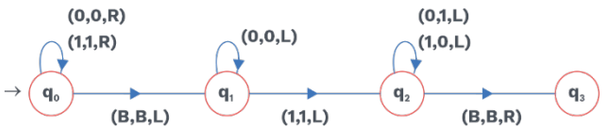
Q1 **Design a TM to find 1's complement of a binary number.**

Sol:



Q2 **Design a TM to find 2's complement of a binary number.**

Sol:

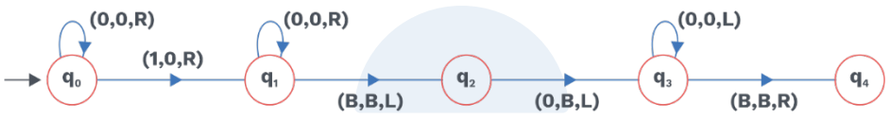


Note: Always, the read-write head of Turing machine will point to starting of the input.

Q3 **Design a turing machine to add two number and the numbers are represented as unary number and use '1' as a separator between two number (for ex. 3 will be represented as '000' in unary).**

Sol: **Ex.:** 3 + 4 can be represented as input tape as BB00010000BB and the output will be BB0000000BB.

So, it is the same as a concatenation of two numbers.



Q4 **Design a Turing machine to compare two number and two numbers are like a (3) b (5) 111 0 11111 . Here, '0' act as separator.**

Sol: Here, we have three cases:

Case 1:

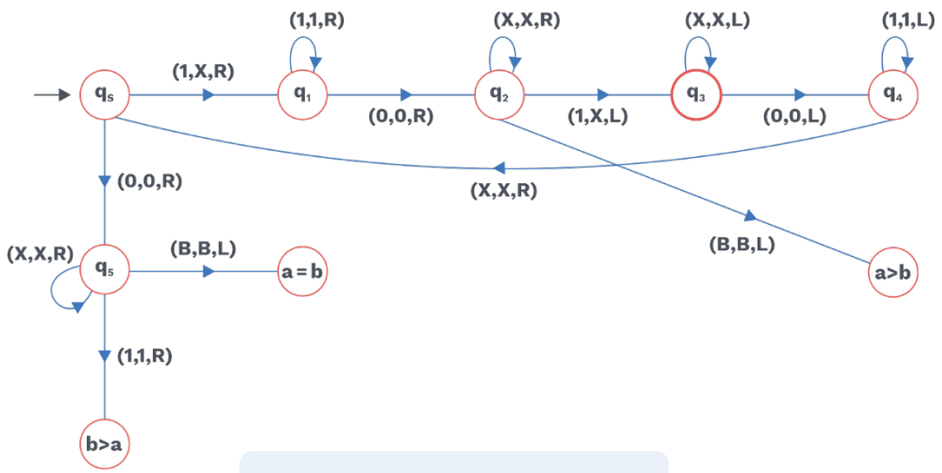
When a = b, it means that it should halt in a state which will indicate both are equal.

Case 2:

When a > b, it means that it should halt in a state which will indicate 'a' is greater than b.

Case 3:

When a < b, it means that it should halt in a state which will indicate 'b' is greater than 'a'.

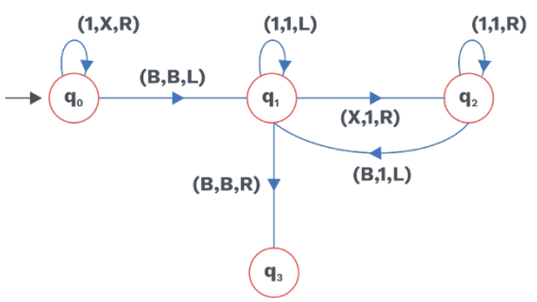


Note:

- Turing machine is capable of doing addition and comparison.
- So, if we could do addition and comparison we could do any other mathematical operation. So, every mathematical function either multiplication, division, logarithm or exponential can be written in terms of comparison and addition.
- So, Turing machine can implement any mathematical function. Therefore, Turing machine is mathematical complete.
- Turing machine can compute mathematical function even using any other number system.

Q5 **Design a turing machine which will do copies of the input for example if the input is B11B then output will be BB1111BB.**

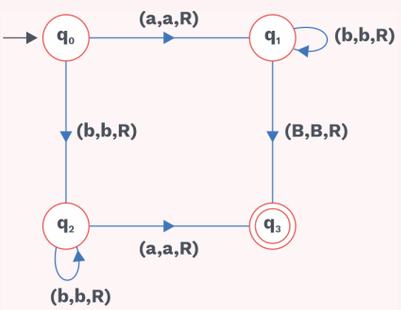
Sol:



Rack Your Brain

1) For $\sum = \{0, 1\}$, design a Turing machine that accepts the language denoted by the regular expression 00^*.

2) What language is accepted by the Turing machine whose transition graph is in the figure below?



Main features of standard turing machine:

Apart from all the different descriptions of a Truing machine, the standard one can be defined as

- As δ defines at most one move always for each configuration, the Turing machine becomes deterministic by default.
- The definition doesn't incorporate any specific input or output file. There is an assumption that- initially, the tape holds some characters. A specific number of these characters can be treated as an input. In a similar manner, a definite series of characters belonging to the tape maybe visualized as the output.

Turing machine as language accepters:

The Turing machine M $(Q, \sum, \Gamma, \delta, q_0, B, F)$ will accept the language called $L(M) = \{w \in \sum^+ : q_0 w \vdash^* x_1 q_f x_2$ for some $q_f \in F, x_1, x_2 \in \Gamma^*\}$

Computable:

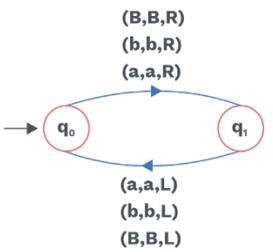
The necessary condition of Turing computable or computable for a function with a given domain is, there should be some Turing machine $M(Q, \sum, \Gamma, \delta, q_0, B, F)$ such that,

$q_0 w \vdash^*_M q_f f(w)$, $q_f \in F$, for all $w \in D$.

Halting problem:

- There are some Turing machines which never halt. It may go into infinite loop.

Example:



- In a Turing machine, the not halting issue i.e whether the TM is ever halted or not, which is commonly known as the halting problem.
- This type of problem does not occur either in Finite Automata and Pushdown Automata because there we have only one move in a forward direction.
- So, when input gets over, either the FA or PDA will definitely halt, but in the case of TM it can go either forward or backwards.

Variations of turing machines:

Equivalence of classes of automata:

If two automatons have acceptance for the same language, then they are called equivalent. For example, suppose there are two automata classes, C_1 and C_2 and two automatons M_1 and M_2. If for every M_1 in C_1 there is an M_2 in C_2 such that $L(M_1) = L(M_2)$ then we can equalize the power of both the classes. And, if the vice versa is also hold then the automata classes, i.e. C_1 and C_2 are equivalent.

Turing machines with a stay-option:

- The standard Turing machine definition says that the movement of the read-write head must be to the left or to the right.
- Another most convenient option provided, is to do nothing or to stay at the current place even after updating the symbol.
- Hence, the Turing machine with a do-nothing or stay option can be defined by modifying δ

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

S represents do-nothing or no movement of the R/W head.

- Anyway, this option will not add extra power to the automaton.

Turing machines with semi-infinite tape:

- The tape is bounded only in one direction.
- Let's visualize a tape with a left boundary.

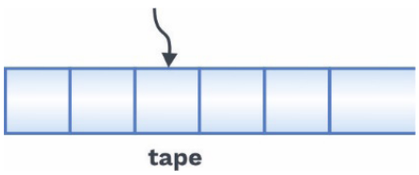


Fig. 5.2 Diagram of Turing machine with Semi-Infinite Tape

The off-line turing machine:

- Here, input is given in a separate file, and we are not allowed to modify the input.
- In case, we have to do some modification on input, then we have to 1st copy entire input into the tape, and then we have to do modifications to the tape.
- Even after computation is over, the input is going to remain intact on the input file.

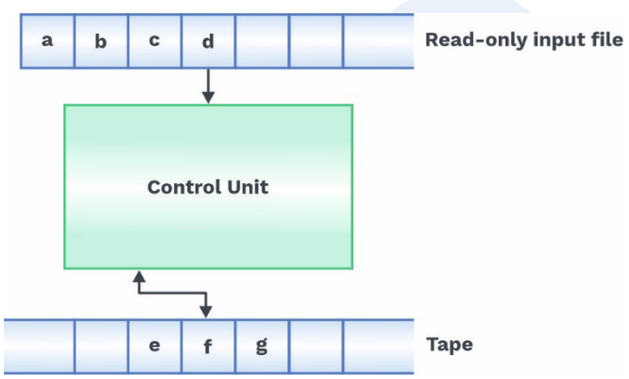


Fig. 5.3 Diagram of The off-Line Turing Machine

- This modified TM is also equivalent in power to standard TM.

Multitape turing machines:

- More than one tape is there in the case of a multitape Turing machine, and the most important part is every tape has its dedicated R/W head.
- We define an n-tape machine by $M=(Q,\Sigma,\Gamma,\delta,q_0,B,F)$ where Q, Σ, Γ, q_0, F are the same as standard TM, but where

$$\delta : Q \times \Gamma^n \rightarrow Q \times \Gamma^n \times \{L,R\}^n$$

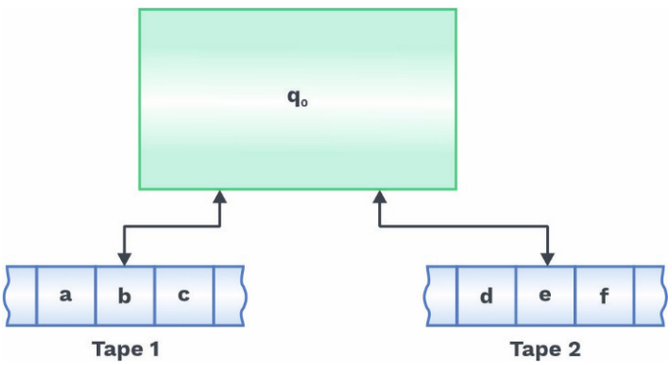


Fig. 5.4 Diagram of Multitape Turing Machines

- Multitape turing machine is also equivalent to standard turing machine.

Multidimensional turing machines:

- In such Turning architecture, infinite extension of the tape is allowed in any direction.

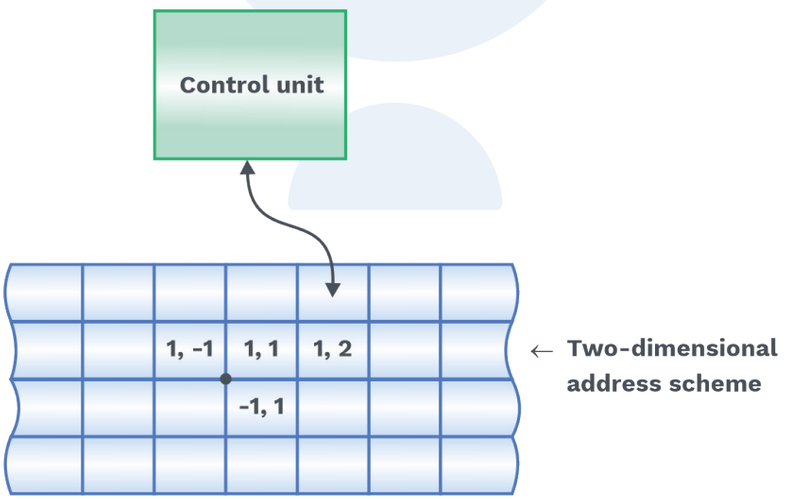


Fig. 5.5 A Diagram Of A Two-Dimensional Turing Machine Is Shown Above.

- The transition function δ of a 2D Turing machine can be defined as

$$\delta : Q \times \Gamma \rightarrow Q \times \Gamma \times \{L, R, U, D\}$$

where, U represents the up movement of the read-write head and D represents the down movement.

- Multidimensional Turing machine is equivalent to the standard Turing machine.

Jumping turing machine:

- In this jumping TM, instead of moving just one step ahead either to the left or to the right, it can move many steps in one go.
- Here, the δ is given below

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

$$\uparrow$$

Here, 'n' number of steps.

- This jumping TM is also equivalent in power to standard TM.

Non erasing turing machine:

- In non-erasing TM, on the input symbol, we are not allowed to write Blank(B) but we can write any other symbol.
- This non-erasing TM is also equivalent in power to standard TM.

Always writing TM:

- In always writing Turing machine, it is always needed to modify the input with different symbols whenever the Turing Machine sees any input. Turing Machine cannot leave it as it is. Example: if TM get 'a' as input, then it will write something which is not 'a'.
- This Turing machine is also equivalent to standard TM.

Multihead TM:

- This modified TM have more than 1 read-write head.
- Multiple heads simultaneously look after the scanned symbol and make the needful, i.e. move or write onto the tape independently.
- A Single head Turing machine can simulate a multi head Turing machine.

Automata with a queue:

- If there are finite Automata, and if we are adding a queue, then it is equivalent to standard TM.

Turing machine with only 3 states:

- Any Turing machine can be minimized to a Turing machine which has only 3 states.
- If a Turing machine has only three states, then it is also equivalent to a standard Turing machine.

Multitape TM with stay option and atmost 2 states:

- Any Turing machine can be as multitape TM with stay option and almost 2 states.
- This modified TM also have the same power as standard TM.

Non deterministic turing machine:

- A non deterministic Turing machine is an automaton is defined by

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

Where
Q = finite set of internal states
Σ = finite set of input alphabets
Γ = Finite set of symbols or tape alphabets
δ = the transition function

$$\delta : Q \times \Gamma \rightarrow 2^{Q \times \Gamma \times \{L,R\}}$$

B = blank space (special symbol) from tape symbol.
q_0 = initial state $\in$ Q
F = Set of final states $\subseteq$ Q

- In a non-deterministic machine, the range of δ is a set of all possible paths, so the machine can take any path out of all possible paths.
- The deterministic Turing machines classes are equivalent to the non deterministic Turing machines classes.
- If at least one configuration of a non deterministic Turing machine accepts a string w belongs to L then that language L is said to be accepted by the same Turing Machine.
- "A non deterministic Turing machine M is said to decide a language L if, for all w $\in$ Σ*, There is a path that leads either to the acceptance or rejection."

Note: A NPDA with one extra independent stack. The δ is given below:-
δ: $Q \times (U\{\in\}) \times \Gamma \times \Gamma \;\; 2^{Q \times \Gamma^* \times \Gamma^*}$
This is also equivalent to standard TM.

Multistack machines:

- The tapes of a multi tape Turing machine in such a way, that it starts behaving like a stack.
- A one-stack machine is a DPDA, while a machine with two stacks is a Turing Machine.

Grey Matter Alert!

Simulating a Turing Machine by a real computer

- It is possible, in principle, to simulate a TM by a real computer if we accept that there is a potentially infinite supply of a removable storage device such as a disk, to simulate the non-blank will be better portion of the TM tape.
- Since the physical resources to make disks are not infinite, this argument is questionable.
- However, since the limits on how much storage exists in the universe are unknown and undoubtedly vast, the assumption of an infinite resource, as in the TM tape, is realistic in practice and generally accepted.

Simulating a computer by a turing machine:

- A Turing Machine can simulate the storage and control of a real computer if it uses one tape to store all the locations and their contents registers, main memory, disk and other storage devices.
- Thus, we can surely say that something that cannot be done by a TM also cannot be done by a real computer.

Rack Your Brain

Consider the nondeterministic Turing machine
$M = (\{q_o, q_1, q_2, q_f\}, \{O, 1\}, \{O, 1, B\}, \delta, q_o, B, \{q_f\})$
Informally but clearly describe the language L(M) if 8 consists of the following sets of rules:

$\delta(q_0, O) = \{(q_0, 1, R), (q_1, 1, R)\}$
$\delta(q_1, 1) = \{(q_2, 0, L)\}$
$\delta(q_2, 1) = \{(q_0, 1, R)\}$
$\delta(q_1, B) = \{(q_f, B, R)\}$

Universal turing machine:

- In a universal Turing machine M_u, a standard Turing machine(M) will be given as input along with an input string w. We can simulate the computation of M on w.
- To give a standard Turing machine as an input, first, a standard representation of a Turing machine is required.
- Assume $Q = \{q_1, q_2, \ldots\ldots, q_n\}$
- where, q_1 = initial state, q_2 = single final state, and

$$\Gamma = \{a_1, a_2, \ldots\ldots, a_m\},$$

Where, a_1 represents the blank symbol.

- Suppose, there is an encoding where q_1 can be represented as 1, q_2 can be represented as 11 and so on.
- Similarly, the tape input symbols can be encoded as $a_1 = 1$, $a_2 = 11$, and so on.
- To separate the 1's or the strings of 1's can be distinguished using 0 as a symbol.
- Any Turing machine can be defined by the transition function δ along with the initial, final state and the blank symbol defined.
- "The transition function is encoded according to this scheme, with the arguments and result in some prescribed sequence". For example, $\delta(q_1,a_2) = (q_2,a_3,L)$ might appear as

...10110110111010.....

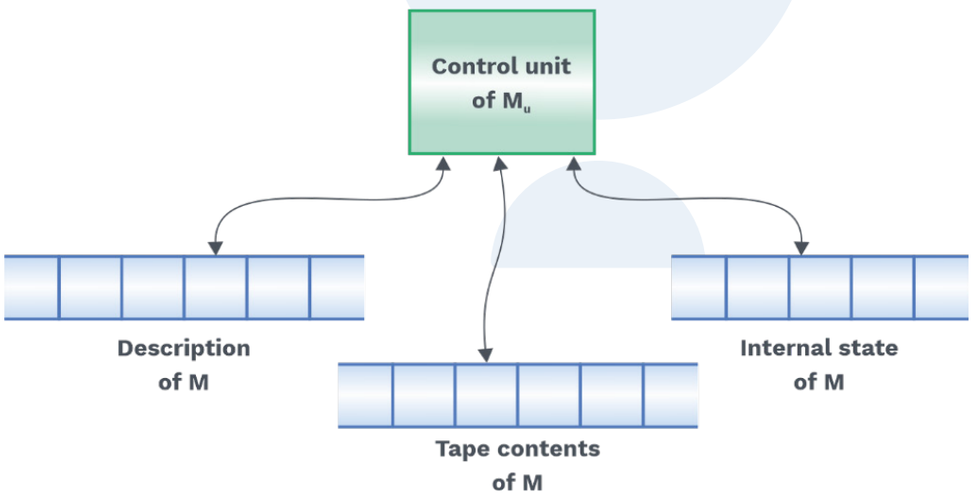


Fig. 5.6

Note:

- It follows from this that any turing machine has a finite encoding as a string on $\{0,1\}^+$ and that, given any encoding of M, we can decode it uniquely.
- But any combination of strings will not represent any Turing machine (e.g. the string 00011).

Working procedure of universal turing machine:

- The encoded definition of M will be kept in the tape 1, given an input M and w.
- The tape symbols of M will be placed in the tape 2 and tape 3 will contain the states of M.
- The universal Turing machine identifies the configuration of M by looking at the tape 2 and tape 3.
- The tape 1 will help for the transition by seeing tape 2 and tape 3.
- Finally, the changes will be reflected on the tape 2 and tape 3, the results will be reflected in tape 2.
- As it can be implemented using any programming language, we can implement the same using any standard Turing machine also.

Previous Years' Question

A single tape Turing Machine M has two states q_0 and q_1, of which q_0 is the starting state. The tape alphabet of M is {0, 1, B} and its input alphabet is {0, 1}.
The symbol B is the blank symbol used to indicate end of an input string. The transition function of M is described in the following table. *(GATE-2003)*

	0	1	B
q0	q1, 1, R	q1, 1, R	Halt
q1	q1, 1, R	q0, 1, R	q_0, B, L

The table is interpreted as illustrated below.
The entry (q1, 1, R) in row q0 and column 1 signifies that if M is in state q0 and reads 1 on the current page square, then it writes 1 on the same tape square, moves its tape head one position to the right and transitions to state q1.
Which of the following statements is true about M?
1) M does not halt on any string in $(0 + 1)^+$
2) M does not halt on any string in $(00 + 1)^*$
3) M halts on all strings ending in a 0.
4) M halts on all strings ending in a 1.
Sol: Option 1)

Note:

- FA + tape = FA + 2stack = FA + queue = TM
- TM = FA+ 2 stack
  = FA +3 stack
  = FA + 4 stack
  .
  .
  .
  = FA+ n stack (n ≥ 2)

Recursively enumerable:

- To ensure a language L is recursively enumerable, we have to find a Turing machine for it.
- Recursive Enumerable can be defined as a Turing machine M, where, for every w ∈ L,

$$q_0 w \vdash^*_M x_1 q_f x_2,$$

with q_f a final state.

- If string w is not in the language, then the definition doesn't say whether it will go in a non-final state and halt or it may go into an infinite loop.

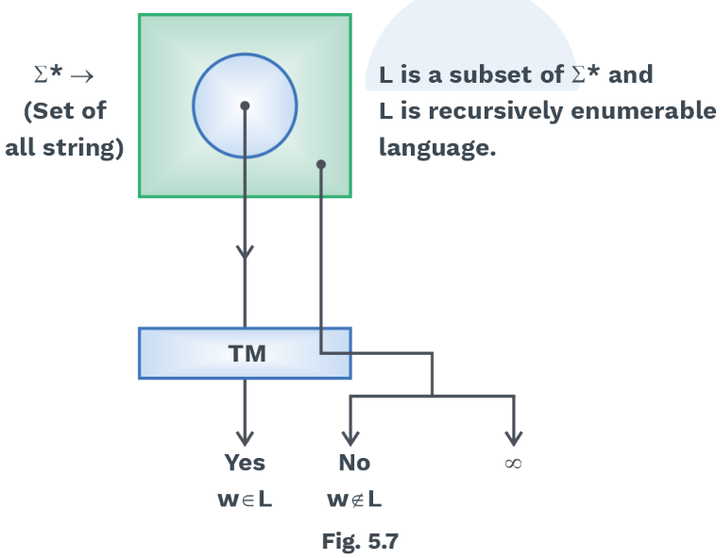


Fig. 5.7

Recursive language:

1) A formal language for which a standard Turing machine exists, i.e., any strings from that language will be accepted by the Turing machine and the machine halts as well is called recursive language.

2) "The Turing machine that always halt is called Halting TM/decider/Total Turing machine."

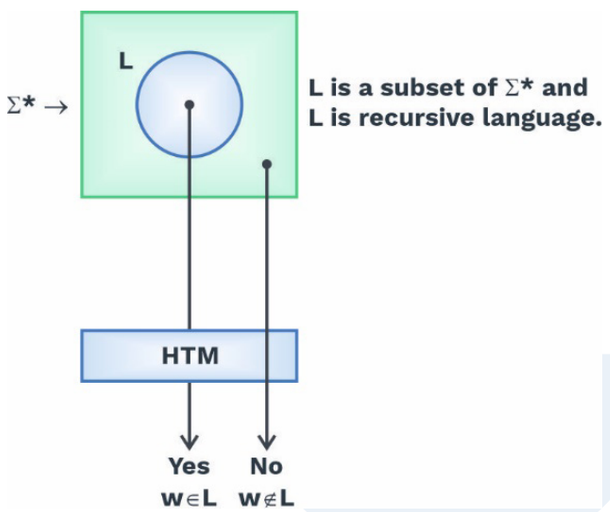


Fig. 5.8

Conclusion

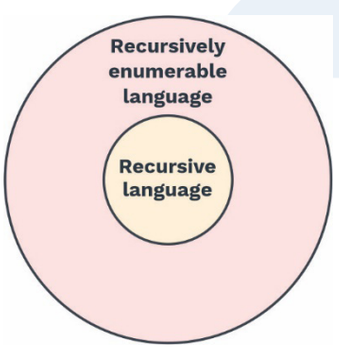


Fig. 5.9

"Recursive Language is a subset of recursively enumerable language".

Chomsky hierarchy:

1) All formal languages are divided into 4 classes by Chomsky and those class hierarchy known as “Chomsky Hierarchy”.

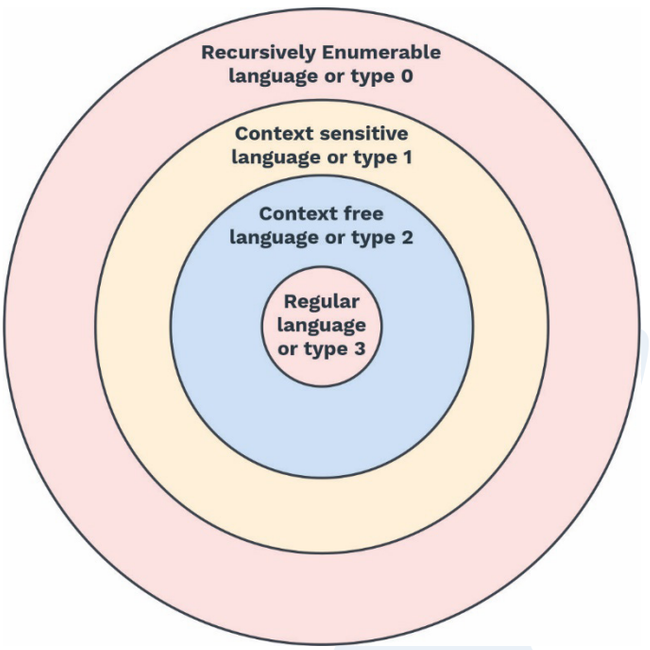


Fig. 5.10

2) Actually Type 0 represents recursively enumerable language, type 1 represents context-sensitive language, type 2 represents context free language and type 3 represents regular language.

Note:

Recursive language is not type (0) language. Only recursively enumerable language is type (0) language.

Extended Version of Chomsky Hierarchy:

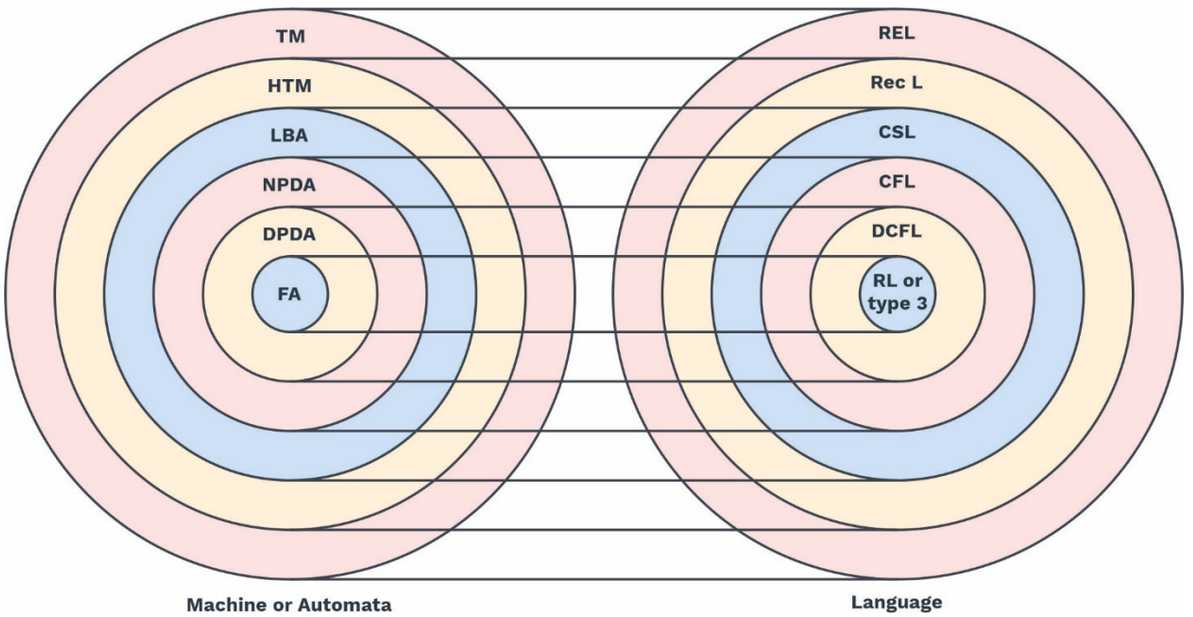


Fig. 5.11

Language and their corresponding grammar:

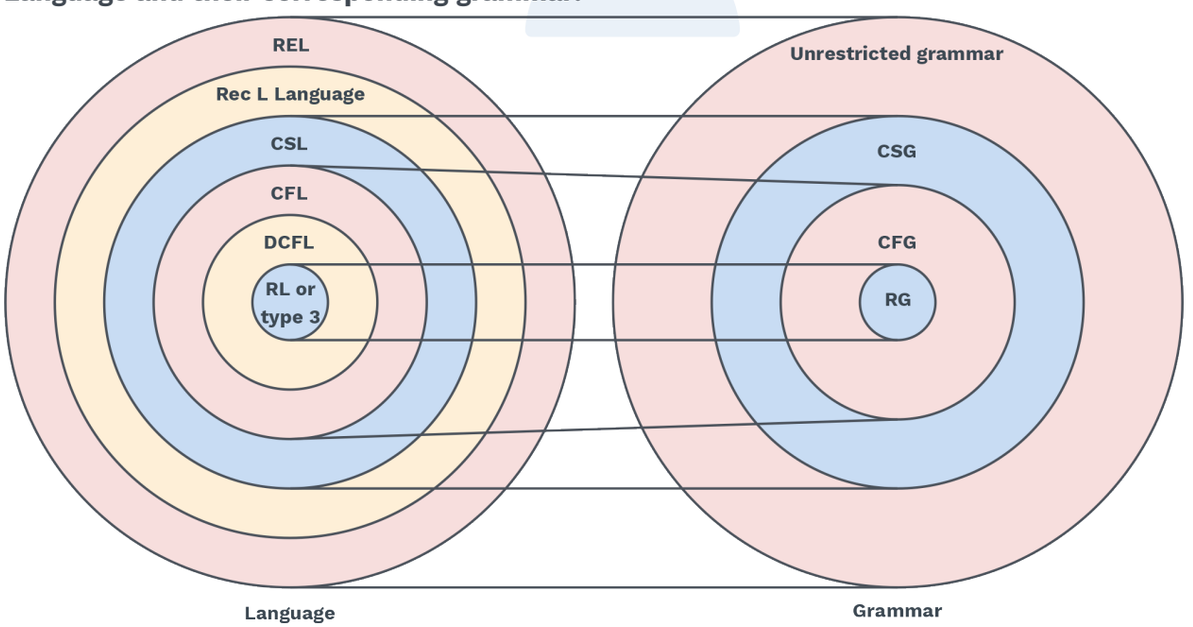


Fig. 5.12

Theorem 1:
"If a language L and its complement $\bar{L}$ are both recursively enumerable then both languages are recursive."

Theorem 2:
"If L is recursive, then $\bar{L}$ is also recursive and consequently both are recursively enumerable."

Unrestricted grammar/type 0 grammar:
When all the productions in a grammar are of the form $X \rightarrow Y$
where $X \in (V + T)^+$ and $Y \in (V + T)^*$
[Here, V indicate set of variables and 'T' indicate set of terminals]

- Unrestricted grammar can be defined '$\in$' but TM is not configured in order to accept a '$\in$' string. So, we assume that we don't consider language deriving '$\in$'.

Ex.: $S \rightarrow S_1B$

$S_1 \rightarrow aS_1b$

$bB \rightarrow bbB$

$aS_1B \rightarrow aa$

$B \rightarrow \in$

Closure properties of Recursive and Recursive Enumerable Language

Property	Recursive	RE language
Union	Yes	Yes
Intersection	Yes	Yes
Set Difference	Yes	No
Complementation	Yes	No
Intersection with regular language	Yes	Yes
Union with regular language	Yes	Yes
Concatenation	Yes	Yes
Kleen Closure	Yes	Yes
Kleen Plus	Yes	Yes
Reversal	Yes	Yes
Homomorphism	No	Yes
$\in$-Free Homomorphism	Yes	Yes
Inverse Homomorphism	Yes	Yes
Substitution	No	Yes

Table 5.1 Cosure Properties

Theorem 1:

"For all recursive language 'L' there is an enumeration technique for it."

Proof:

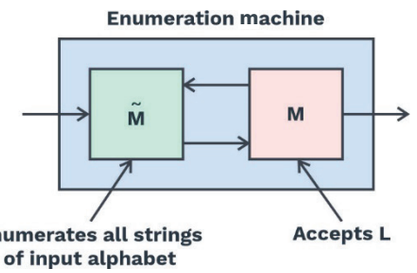


Fig. 5.13

If the alphabet is {a, b} then $\tilde{M}$ enumerate strings as follows:

'a'
'b'
'aa'
'ab'
'ba'
'bb'
'aaa'
'aab'
...

Enumeration Procedure
Repeat:
$\tilde{M}$ generates a string w
M checks if w ∈ L
Yes: Print w to output
No: Ignore w
End of proof.

Example: L = {b, ab, bb, aaa, ...}

$\tilde{M}$	L(M)	Enumeration Output
a		
b	b	b
aa		
ab	ab	ab
ba		
bb	bb	bb
aaa	aaa	aaa
aab		
...		

Note:

Theorem 2:

"A language is recursively enumerable if and only if there is an enumeration procedure for it."

Previous Years' Question

Define languages L_0 and L_1 as follows: **(GATE-2003)**

L_0 = {<M, w, 0> | M halts on w}

L_1 = {<M, w, 1> | M does not halts on w}

Here <M, w, i> is a triplet, whose first component M is an encoding of a Turing Machine, second component w is a string, and third component i is a bit.

Let L = L_0 U L_1. Which of the following is true?

1) L is recursively enumerable, but L′ is not.

2) L′ is recursively enumerable, but L is not.

3) Both L and L′ are recursive.

4) Neither L and L′ is recursively enumerable.

Sol: Option 4)

Previous Years' Question

Consider the following languages. **(GATE-2016 (set 2))**

- L_1 = {<M> | M takes at least 2016 steps on some input},
- L_2 = {<M> | M takes at least 2016 steps on all inputs} and
- L_3 = {<M> | M accepts $\in$},

where for each Turing machine M, <M> denotes a specific encoding of M.

Which one of the following is TRUE?

1) L_1 is recursive and L_2, L_3 are not recursive

2) L_2 is recursive and L_1, L_3 are not recursive

3) L_1, L_2 are recursive and L_3 is not recursive

4) L_1, L_2, L_3 are recursive

Sol: Option 3)

Linear bound automata:

- We can limit the power of a Turing machine by restricting the way in which we are going to use tape.
- If we limit the Non-deterministic TM in such a way that it can use the tape like a stack then it converges to a PDA.

- If we limit the Turing machine to use only finite amount of cell and only unidirectional tape movement is allowed then it can converges into finite automata.
- If we limit tape of Turing machine to use only that part of the tape where the input is present and not beyond it, then it converge to LBA (Linear Bounded Automata).
- A linear bounded automata is designed with an unbounded tape. It extents up to which the tape needs to be made use of purposely, with an instant of the input. Practically, the unbounded tape is restricted by the input string.
- To enforce this, we can envision the input as bracketed by the two special symbols, the left-end marker and the right-end marker.
- LBA is less powerful than TM and it is more powerful than PDA. Ex.: $a^nb^nc^n \mid n \geq 1$ is accepted by LBA but not by PDA.
- Here, we don't know whether the deterministic LBA are equivalent in power to non-deterministic LBA. It is undecidable.
- Complement of the CFL is CSL.
- Language accepted by LBA is called context sensitive language.
- LBA is halting, Turing machine as context sensitive language is also a recursive language.

Countability

- **Countable set definition:** "A set 's' is said to be countable, if the elements of the set can be put in one to one correspondence with the set of natural numbers."
- By this we mean that the elements of the set can be written in some order, say, x_1, x_2, x_3, ..., so that every element of the set has some finite index.

for ex.: E = set of even number = {0, 2, 4, 6, 8...}

For any element in E of form 2i (i starts with 0) we give a corresponding element (i + 1) in N (Natural Number). So, for every element in even number set we can associate a natural number as a correspondence. Therefore, we can say that set of even number is countable.

eg.: O = set of odd number

= {1, 3, 5, 7, 9, 11, ...}

Here, we can represent set of odd number in terms of (2i + 1) ($i \geq 0$). We can map every odd number of the form (2i + 1) with (i + 1) (Natural number).

$$O = \{1, 3, 5, 7, 9, 11, \ldots\}$$
$$\downarrow \downarrow \downarrow \downarrow \downarrow \downarrow$$
$$N = \{1, 2, 3, 4, 5, 6, \ldots\}$$

Hence, the set of odd number is countable.

Uncountable set definition: "A set is uncountable if it is infinite and not countable."

Rack Your Brain

Given R = set of real number. Is R countable set?

Alternative definition of countability:

- A set is said to be countable if there exists an enumeration method using which all the elements of the set can be generated and for any particular element, it takes only a finite number of steps to generate it.
- The finite number of steps taken to generate an element can be used as its index and hence a mapping into a natural number set.

for ex.:

i) Set of even number

Enumeration procedure: for (q = 0 to ∞)

printf (2 × q);

Output :	0	2	4	6	8	...
Steps :	1	2	3	4	5	...

So, the index for (0, 2, 4, 6, 8, ...) is (1, 2, 3, 4, 5, ...).

Hence, set of even number is countable.

Similarly;

ii) Set of odd number

for (q = 0 to ∞)

printf (2 × q + 1)

Output :	1	3	5	7	9	11	...
Steps :	1	2	3	4	5	6	

SOLVED EXAMPLES

Q1 **Let S = set of all quotients in the form of p/q, where p and q are positive integers and q ≠ 0. Check whether S is countable or not?**

Sol: The set S is countable since we are having an enumeration procedure for set S.
Here, we don't go either in increasing order of numerator or denominator, we just add numerator and denominator and we just go in the increasing order of sum.
Here, the smallest number is 1/1.
So, the 1st sum is 2.
Within the sum we can enumerate in increasing order.
So, the enumeration order is $\frac{1}{1}, \frac{1}{2}, \frac{2}{1}, \frac{1}{3}, \frac{2}{2}, \frac{3}{1}, \ldots$
Hence, it is countable.

Lexicographic ordering:

- Lexicographic ordering is nothing but the alphabetical order, i.e., the sequence of letters present in the dictionary.
- Suppose we have a set Σ = {a, b} and Σ* in lexicographical order will be {∈, a, b, aa, ab, ba, bb, aaa, aab, ...}

Rack Your Brain

Given a set S = set of all string over {a, b} is
1) Countable
2) Uncountable
3) Finite
4) Neither countable nor uncountable

Note:

- Set of all string possible over any alphabet is countable.
- So, Σ* = set of all string over Σ is countable.
- Every subset of countable set is either finite or countable.
- Since every language is a subset of Σ*. Therefore, every language is countable.
- It means given any language it is definitely either countable language or finite language.

Q2 **Check whether T = set of all Turing machine is countable or not?**

Sol: Since every Turing machine can be encoded as a string of 0's and 1's.
Let $\Sigma = \{0, 1\}$ then Σ^* = set of all strings over {0, 1}.
Since, Σ^* is countable and T is subset of Σ^*.
Hence, T = set of all Turing machines.

Implication of the fact that the set of all turing machines are countable:

- Since we know that the set of Turing machines are countable, then the set of recursively enumerable language are also countable.
- Since the set of recursive language is subset of the set of recursively enumerable language. Hence, a set of recursive languages is countable.
- Similarly, a set of context free language, a set of context sensitive language, and the set of regular language are also countable.
- We can also modify a turing machine in such a way that it can act as PDA. Therefore, if set of TM is countable it implies that the set of PDA is also countable.
- Since, all machines are subset of set of Turing machine. So, the set of LBA, and set of FA are also countable.

Theorem: Set of all languages possible are uncountable.

Implication from theorem:

- Set of TM is countable, and the set of all languages possible is uncountable.
- Therefore, there are some languages for which there are no machines yet. It means there are some languages which are not even recursively enumerable.

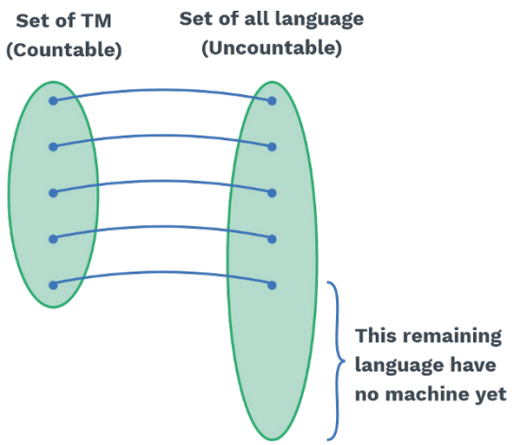


Fig. 5.14

Theorem:
"set of all languages are uncountable".

Proof:
Suppose there is a set $\Sigma = \{a, b\}$.
We need to prove that, 2^{Σ^*} is uncountable.
Let us assume, 2^{Σ^*} is countable.
As, $\Sigma = \{a, b\}$
$\Sigma^* = \{\in, a, b, aa, ab, ba, bb, aaa, aab,....\}$

Let us represent each string using 0 or 1, if the string is present in the language $\in 2^{\Sigma^*}$ then mark that string using '1' otherwise '0'.

Σ^*	$\in$	a	b	aa	ab	ba	bb	aaa	aab	.....
Language 1	(0)	1	1	0	1	0	1	0	0	.....
Language 2	1	(0)	1	1	1	1	1	0	0	.....
Language 3	0	1	(1)	0	0	1	1	0	0	.....
Language 4	1	1	1	(1)	1	0	1	0	1	.....
Language 5	1	1	1	1	(1)	1	1	1	1	.....
.	.	.	.	.	.	.	.	.	.	.....
.	.	.	.	.	.	.	.	.	.	.....
.	.	.	.	.	.	.	.	.	.	.....

Now, if we consider the diagonal, then we will find a string S = 0 0 1 1 1 ..... Complement of the diagonal 00111... will be S′ = 1 1 0 0 0..... which is not in the language.

This particular language $\overline{S}$ is not included in that list as it is a different form all the languages by at least one string.
So, our assumption was incorrect.
Hence, 2^{Σ^*} is uncountable. (Proved)

Note:

This kind of argument, because it involves a manipulation of the diagonal elements of a table is called diagonalization.

Note:

Theorem:

"There exists a recursively enumerable language whose complement is not recursively enumerable."

Theorem:

"There exists a recursively enumerable language that is not recursive i.e., the family of recursive languages is a proper subset of the family of recursively enumerable languages."

- Diagonalization method is used to prove that a language is not recursively enumerable because if for a language, we proved that it is not countable then it means there is no enumeration procedure for that language.

  And theorem says that a language is Recursive Enumerable Language if and only if there is enumeration procedure for it. Hence, that language is not recursively enumerable.

Theorem:

"If S_1 and S_2 are countable sets, then $S_1 \cup S_2$ and $S_1 \times S_2$ are countable and union and cross-product can be extended to any number of finite sets."

Theorem:

"The set of all languages that are not recursively enumerable is uncountable."

Proof:

The set of all languages is 2^{Σ^*} (set of all subsets of Σ^*), and out of that some languages are already recursive enumerable, because if there is TM for a language then that language is recursively enumerable and we already know that the number of TM is countable. Therefore, the number of languages in the total language which is recursively enumerable are countable and that will be equal to a number of Turing machines.

And in the remaining set, we don't know anything, so we are just assuming that it is countable. So, the resulting set is the union of a countable set; therefore overall, that should be countable. And if that is true, then our initial proof that 2^{Σ^*} is uncountable is a failure as we know that set of all languages 2^{Σ^*} is uncountable. Therefore, our assumption that, the remaining language i.e., the set of all languages that are not- recursively enumerable is countable, is false. So, set of all languages which are not recursively enumerable, must be uncountable.

Previous Years' Question

Consider the following sets:
S1: Set of all recursively enumerable languages over the alphabet {O, 1}
S2: Set of all syntactically valid C programs
S3: Set of all language over the alphabet {O. 1}
S4: Set of all non-regular languages over the alphabet {O, 1)}
Which of the above sets are uncountable? **(GATE-2019)**
1) S1 and S2
2) S3 and S4
3) S2 and S3
4) S1 and S4

Previous Years' Question

Let N be the set of natural numbers. Consider the following sets,
P: Set of Rational numbers (positive and negative) **(GATE-2018)**
Q: Set of functions from {O, 1} to N
R: Set of functions from N to {O, 1}
S: Set of finite subset of N
Which of the above set are countable?
1) Q and S only
2) P and S only
3) P and R only
4) P, Q and S only

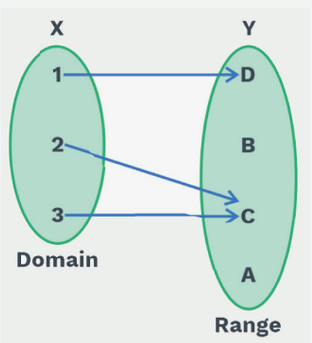
Grey Matter Alert!

Function

- A function is a rule that assigns to elements of one set a unique element of another set.

  OR

  In mathematics, a function is a relation between sets, that associates every element of a first set with exactly one element of the second set.
  Ex.: X = {1, 2, 3} & Y = {A, B, C, D}
  t = {(1, D), (2, C), (3, C)}



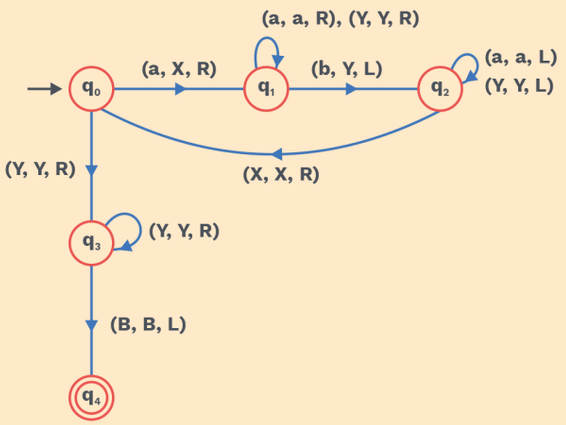
- If f denotes a function, then the first set is called the domain of f and the second set is its range. We write

  $$f : S_1 \rightarrow S_2$$

  to indicate that the domain of f is a subset of S_1 and that the range of f is a subset of S_2.
- If the domain of f is all of S1, we say that f is a total function on S_1, otherwise f is said to be a partial function.

SOLVED EXAMPLES

Q1 **The transition diagram for Turing machine is given below**



Which one of the following strings is accepted by the above TM?

1) ab **2) aab**

3) abb **4) None of the above**

Sol: **1) Explanation:** It is accepting L = $\{a^n b^n \mid n \geq 1\}$. So, (1) is the correct option.

Q2 **Let L = $\{X^{n+m} Y^{n+m} X^m \mid n, m \geq 0\}$. The above language L is**

1) CSL but not CFL **2) Rec L but not CSL**

3) REL but not Rec **4) CFL but not regular**

Sol: **1) Explanation:** It is a CSL because we can give a linear bound automata. We cannot compare number of X's and number of Y's using one stack here.

Q3 **Consider the given statements:**
S1: Set of all irrational number are countable.
S2:If L be a finite language then L^* is recursively enumerable language.
Which of the following statements are correct?

1) S1 **2) S2**
3) S1 & S2 **4) None of the above**

Sol: **2) Explanation:** Statement S1 is incorrect because set of all irrational number are uncountable.

Statement S2: Since L is finite language. So, it is regular. All regular language are recursively enumerable language and recursively enumerable language are closed under Kleene closure. Hence, L^* is recursively enumerable language.

Q4 **Consider the given statements:**
S1: Let L_1 be recursive and L_2 be recursively enumerable, then $L_2 - L_1$ is necessarily recursively enumerable.
S2:$L = \{www^k \mid w \in (a, b)^+\}$ is CSL.
Which of the above statement(s) is/are correct(s)?

1) S1 **2) S2**
3) S1 & S2 **4) None of the above**

Sol: **3) Explanation:**

S1 : $L_2 - L_1 = L_2 \cap \overline{L}_1$

$= REL \cap \overline{Rec\ L}$

$= REL \cap Rec\ L$

$= RE\ L.$

So, statement S1 is correct.

S2 : $L = \{www^k \mid w \in (a, b)^+\}$ is CSL because for this we can give a LBA.

It needs more than one stack to accept this language L, so it is CSL.

Q5 **Consider the given statements:**
S1: Let L_1 be recursive language then its complement $\overline{L}_1$ is also recursive language.
S2: Let L_2 be recursively enumerable language then its complement $\overline{L}_2$ is also recursively enumerable language.
Which of the above statement(s) is/are correct(s)?

1) S1 **2) S2**
3) S1 & S2 **4) None of the above**

Sol: **1) Explanation:** Recursive language is closed under complementation. Hence, $\overline{L}_1$ is also recursive language. Recursively enumerable language is not closed under complementation. Hence, $\overline{L}_2$ may or may not recursively enumerable language.

So, we can't say $\overline{L}_2$ is recursively enumerable language.

Q6 **Consider the following language,**
L_1 = Recursive enumerable language,
L_2 = Recursive language
L_3 = Context free language
Now,
Let $L = (\overline{L}_2 \cap L_1) - L_3$
L is

1) L is CFL **2) L is recursive**
3) L is REL **4) L is not REL**

Sol: **3) Explanation:**

$$L = (\overline{L}_2 \cap L_1) - L_3$$

$$= (\overline{\text{Rec } L} \cap \text{REL}) - \text{CFL}$$

$$= (\text{Rec } L \cap \text{REL}) - \text{CFL}$$

$$= \text{REL} \cap (\overline{\text{CFL}})$$

$$= \text{REL} \cap \text{CSL} = \text{REL}.$$

[Here, Rec L means recursive language,
REL means recursively enumerable language,
CFL means context free language,
CSL means context sensitive language]

Q7 **What is the highest type number that can be assigned to the following grammar?**

$S \rightarrow S_1B$

$S_1 \rightarrow aS_1b$

$bB \rightarrow bbbB$

$aS_1b \rightarrow aa$

$B \rightarrow \in$

1) Type 0 **2) Type 1**

3) Type 2 **4) Type 3**

Sol: **1) Explanation:** The given grammar is in the form of $u \rightarrow v$ where u belongs to $(V+T)^+$ v belongs to $(V+T)^*$ here, V is set of non-terminals and T is set of terminals Hence, it is unrestricted grammar or type-0 grammar.

Q8 **If L_1 and L_2 are two recursively enumerable languages, then they are not closed under**

1) Complementation **2) Union**

3) Intersection **4) Concatenation**

Sol: **1) Explanation:** Recursively enumerable language is not closed under "complementation" and it is closed under "union", "intersection", "concatenation", etc.

Q9 **A Turing machine has _____ number of states.**

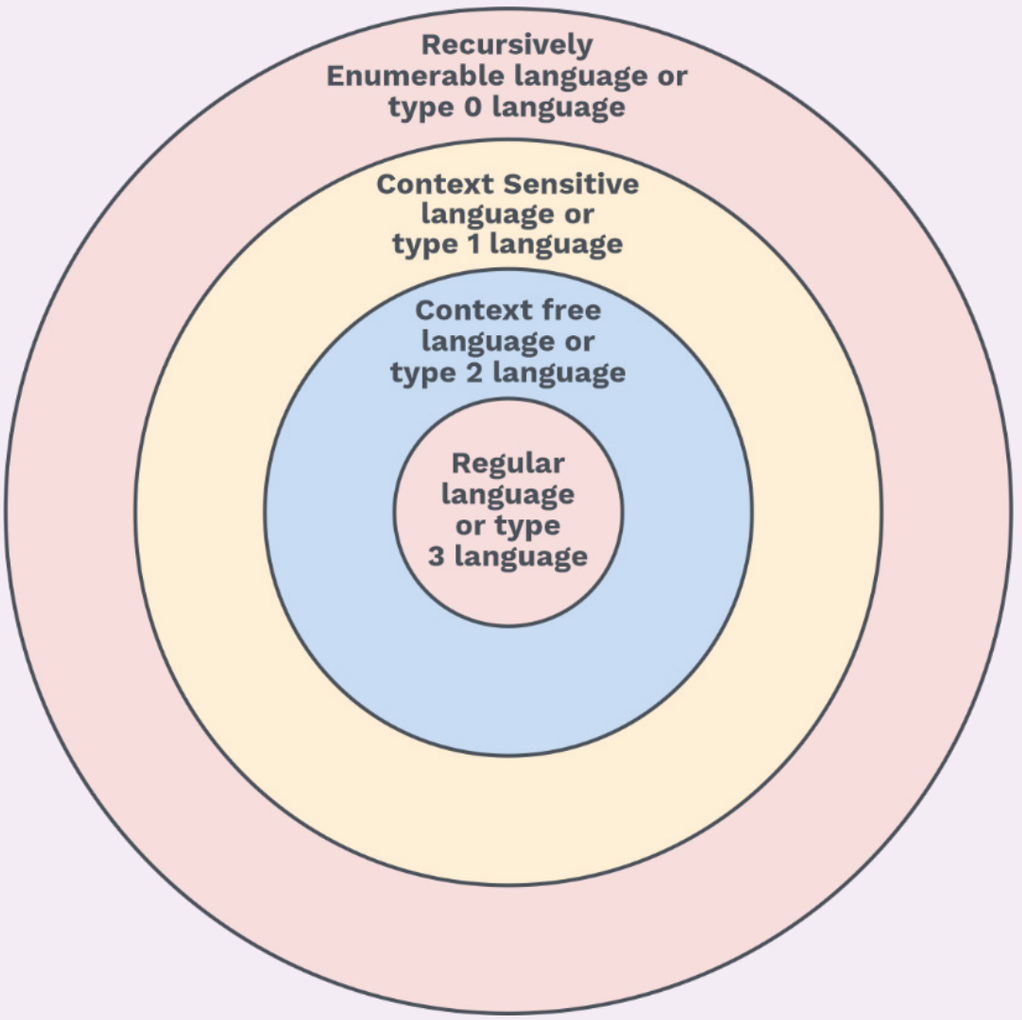
1) Finite **2) Infinite**

3) May be finite **4) None of the mentioned**

Sol: **1) Explanation:** In the definition of Turing machine, 'Q' represents the "Set of finite number of states".

Chapter Summary

- A Turing machine M is defined by M = (Q, Σ, Γ, δ, q_0, B, F) where Q is the set of internal states, Σ is the input alphabets, Γ is a finite set of symbols called the tape alphabet, δ is the transition function, defined as
  δ: {Q × Γ → Q × Γ × {L, R}} where L = left and R = right, B ∈ Γ is a special symbol called the blank, q_0 ∈ Q is the initial state, F ⊆ Q is the set of final states.
- Turing machine can't accept epsilon.
- Turing machine can act both as a transducer as well as acceptor.
- Turing machine is mathematically complete. It can do any mathematical function.
- **Halting Problem:** Some times Turing machine goes into an infinite loop. This problem of not halting of Turing machine is called the halting problem in the Turing machine.
- **There are different variations of Turing machines as shown below:**
  i) Turing machine with a stay option.
  ii) Turing machine with semi-infinite tape.
  iii) The off-line Turing machines.
  iv) Multitape Turing machines.
  v) Multidimensional Turing machines
  vi) Jumping Turing machine
  vii) Non Erasing Turing machine
  viii) Always writing Turing Machine
  ix) Multi head Turing Machine
  x) Turing machine with only 3 states.
  xi) Multitape Turing Machine with stay option and at most 2 states.
  xii) Non-deterministic Turing machine
  All these variations of the Turing machines are equivalent in power to the standard Turing machines.
- We can represent any Turing machine in terms of 0's and 1's but any combination of 0's and 1's is not a Turing machine.
- Finite Automata + n stack (n ≥ 2) = TM
- Finite automata + queue = TM.
- **Recursively Enumerable Language:** For a language L, if there is a Turing machine that can accept L, then the language L is known as recursively enumerable.
- **Halting TM:** Turing machine that always halts is called Halting TM/decider/Total Turing machine.
- **Chomsky Hierarchy:** All formal languages are divided into 4 classes by Chomsky and those class hierarchy known as "Chomsky Hierarchy".



- **Unrestricted Grammar/Type 0 Grammar:** A grammar is called unrestricted if all the productions are of the form $X \rightarrow Y$
- where X is in $(V + T)^+$ and Y is in $(V + T)^*$
  [Here, V indicates set of variables and T indicates set of terminals]
- **Closure Property of Recursive and Recursively Enumerable Language**

Property	Recursive	RE Language
Union	Yes	Yes
Intersection	Yes	Yes
Set Difference	Yes	No
Complementation	Yes	No
Intersection with regular language	Yes	Yes
Union with regular language	Yes	Yes
Concatenation	Yes	Yes
Kleen Closure	Yes	Yes
Kleen Plus	Yes	Yes
Reversal	Yes	Yes
Homomorphism	No	Yes
$\in$-Free Homomorphism	Yes	Yes
Inverse Homomorphism	Yes	Yes
Substitution	No	Yes

- **Linear Bound Automata:** If we limit the tape of the Turing machine to use only where the input is present and not beyond it, then it converges to LBA.
- In LBA, whether deterministic LBA is equivalent to non-deterministic LBA is not known.
- Language accepted by LBA is called context-sensitive language.
- **Countable Set:** If there persists a one-to-one mapping from the elements of a set to the set of positive integers, such a set is classified as countable.
- **Uncountable Set:** A set S is uncountable if it is infinite and not countable.
- Set of all strings possible over any alphabet is countable.

- Set of all languages possible is uncountable.
- There exists a recursively enumerable language whose complement is not recursively enumerable.
- There exists a recursively enumerable language that is not recursive i.e., the family of recursive language is a proper subset of the family of recursively enumerable language.
- If S_1 and S_2 are countable sets, then $S_1 \cup S_2$ and $S_1 \times S_2$ are countable, and union and cross-product can be extended to any number of finite sets.
- The set of all languages that are not recursively enumerable is uncountable.

6 Decidability

What is a decision problem:
A decision problem is a set of related statements, each of which must be either true or false.
For example:
"For context-free grammar (G), the language L(G) is ambiguous or not?
For some context-free grammar (G), this is true, and for some, it is false.
But we can clearly say that the statement is either true or false.
Basically, we have two kinds of problems:

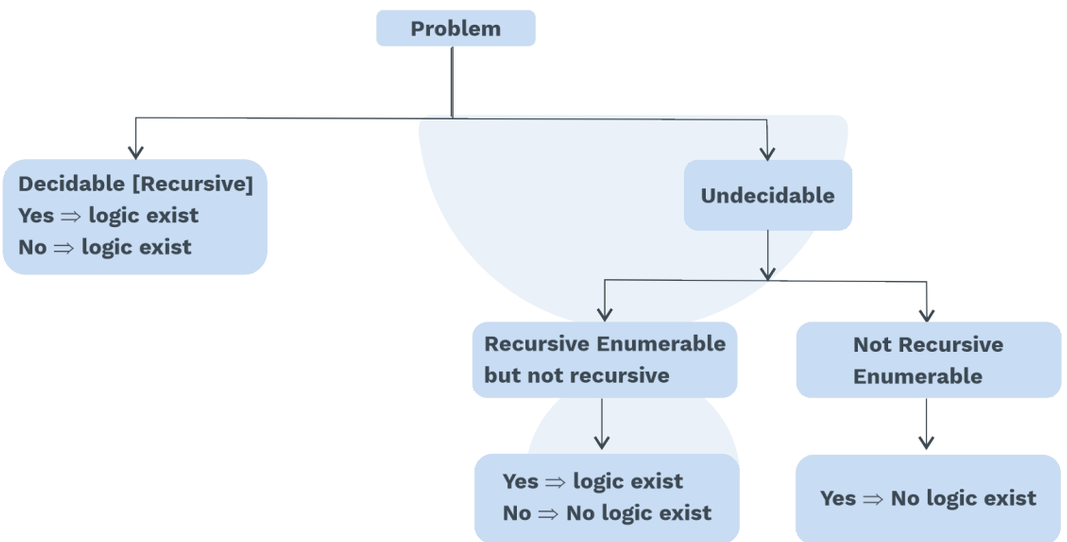


Fig. 6.1 Decidability

Note:

- For decidable problem, the Halting Turing Machine (HTM) exist.
- For undecidable problem\s, the Halting Turing Machine (HTM) does not exist, but Turing Machine (TM) may or may not exist.
- If a problem is not recursive enumerable (RE), then no Turing Machine exists for that problem.

Language:
The set of all strings that can be generated from the grammar.
Language can be finite or infinite. If a language consists of a finite number

of strings, then it is called a Finite Language; otherwise, we can say it is an infinite language.

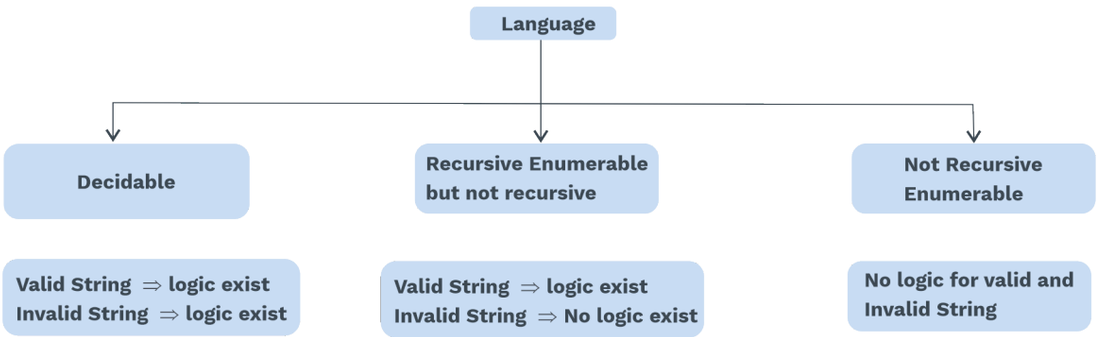


Fig. 6.2

Example:

Consider a String w and a language L

1) If $w \in L$ or $w \notin L$

logic exist logic exist

For both the cases, we have logic, then the language is decidable.

2) If $w \in L$ or $w \notin L$

logic exist No logic exist

Logic exists for valid strings. So, the language is Recursively Enumerable but not Recursive.

3) If $w \in L$ or $w \notin L$

No logic exist No logic exist

For both the cases, we do not have logic, then the language is not Recursively Enumerable.

Encoding a turing machine (TM):

Each Turing Machine with input alphabet {0, 1} may be thought of as a binary string.

To represent a Turing Machine $(M) = (Q, \Sigma, \Gamma, \delta, q, B, F)$ as a binary string where $\Sigma = \{0, 1\}$.

First, assign integer to the tape symbols, states and directions Left (L) and Right (R).
Let the states be: q_1, q_2, q_3, q_4 ...q_k.
Here q_1 is a Starting state, q_2 is a Final State and q_3, q_4 ... q_k are other states.
Let tape symbols are: X_1, X_2, X_3, ...X_m.

X_1 = 0
X_2 = 1
X_3 = B (Blank)

Let the direction left be D_1 and the direction Right be D_2.

Encoding of transition function (δ):
one transition rule is,

$\delta(q_i, X_j) = (q_k, X_l, D_m)$ for some integers i, j, k, l and m.
Now, code this rule by the string- $0^i10^j10^k10^l10^m$
where i, j, k, l, m should be greater than 0 (i, j, k, l, m > 0).
There should not be two consecutive 1's within the code for a single transition.
The code for the entire Turing Machine will be:

$$Q_1\ 11\ Q_2\ 11\ Q_3\ 11\ \ldots\ Q_n$$

Each transition is separated by two consecutive 1's and $Q_1, Q_2 \ldots Q_n$ are the transition of Turing Machine.

Example:
Consider the Turing Machine (TM)

$M = (\{q_1, q_2, q_3, q_4, q_5\}, \{0, 1\}, \{0, 1, B\}, \delta, q_0, B, \{q_2\})$

where δ consists of the rules:

$\delta(q_1, B) = (q_5, B, L)$

$\delta(q_4, 1) = (q_3, B, R)$

$\delta(q_5, B) = (q_4, 1, R)$

$\delta(q_3, 1) = (q_2, 1, L)$

Codes for each transition:

$(q_i, X_j, q_k, X_i, D_m)$

1) $\delta(q_1, B) = (q_5, B, L)$
$(q_1, X_3, q_5, X_3, D_1)$
So,

$0^1 10^3 10^5 10^3 10^1$
01000100000100010

2) $\delta(q_4, 1) = (q_3, B, R)$
$(q_4, X_2, q_3, X_3, D_2)$
So,
$0^4 10^2 10^3 10^3 10^2$
000010010001000100

3) $\delta(q_5, B) = (q_4, 1, R)$
$(q_5, X_3, q_4, X_2, D_2)$
So,
$0^5 10^3 10^4 10^2 10^2$
00000100010000100100

4) $\delta(q_3, 1) = (q_2, 1, L)$
$(q_3, X_2, q_2, X_2, D_1)$
So,
$0^3 10^2 10^2 10^2 10^1$
00010010010010
A code for Turing Machine (M):
0100010000010001011000010010001000100**11**0000100010000100100**11**0
0010010010010
Here, two consecutive 1's are used as Separators.

The diagonalization language (l_d):
It is the set of strings (w_i) such that w_i is not in $L(M_i)$.

$$L_d = \{w_i \in (0+1)^+ \mid w_i \notin L(M_i)\}$$

Let,

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

And we know, Σ^* is countable; now we must prove 2^{Σ^*} is uncountable.

Prove by contradiction:

Let 2^{Σ^*} is countable and it has one-one mapping.
We know,

$$\Sigma^* = \in, a, b, aa, ab, ba, bb, aaa \ldots$$

Let,

$$L_1 = \{\in, a\}$$

$$L_2 = \{a, b\}$$

$$L_3 = \{\in, a, aa\}$$

$L_4 = \{b, ab, ba\}$

$L_5 = \{aa, ba, bb\}$

$L_6 = \{\in, aaa\}$ and so on.

Now, mapping L_1, L_2, L_3, L_4, L_5 and L_6 with Σ^*.

	∈	a	b	aa	ab	ba	bb	aaa	...
L_1 :	1	1	0	0	0	0	0	0	...
L_2 :	0	1	1	0	0	0	0	0	...
L_3 :	1	1	0	1	0	0	0	0	...
L_4 :	0	0	1	0	1	1	0	0	...
L_5 :	0	0	0	1	0	1	1	0	...
L_6 :	1	0	0	1	0	0	0	1	...
; ;	;	;	;	;	;	;	;	;	...
; ;	;	;	;	;	;	;	;	;	...
; ;	;	;	;	;	;	;	;	;	...

Diagonal

Fig. 6.3 The Table That Represents M Accepts W

To construct Diagonal Language (L_d), we complement the diagonal.
Complement of diagonal would begin:

0 0 1 1 1 1...

∈ a b aa ab ba...

Thus, L_d would contain w_i = {b, aa, ab, ba...} except '∈' and 'a'.
Since L_d is not present in Diagonal Matrix or given set of languages. So, the initial assumption was wrong.

By contradiction, 2^{Σ^*} is uncountable.

Proof that L_d is not Recursively Enumerable (RE):
By the above intuition, there is no Turing Machine that accepts the language L_d. So, L_d is not Recursive Enumerable (RE).

An undecidable problem that is recursively enumerable (RE):
It has been already established that diagonal languages are not recognized by Turing machines.
Let's have insight into the area of recursively enumerable languages.

Case 1:
A Turing machine will necessarily halt for strings that are not inclusive to a given language. However, it may or may not enter into an accepting state.

Case 2:
Let us consider a scenario where recursively enumerable languages are involved. The Turing machine will necessarily halt if the input string is inclusive to the language. However, for excluded input strings, the Turing machine may get inter-wined into an infinite loop or halt.

Recursive languages:
A recursive language L If L = L(M) for some Turing Machine (M) such that:
1) $w \in L \Rightarrow$ Halts and accepts
2) $w \notin L \Rightarrow$ Halts and rejects

Note:

Recursive language is always decidable. If language is not recursive, then it is undecidable/semi decidable.

Compliment of recursive and recursive enumerable language (RE):
If a language L is Recursive Enumerable but the complement of recursive enumerable language $(\overline{L})$ is not Recursive Enumerable (RE), then we can say L can not be Recursive.
If L is Recursive, then the complement of $L(\overline{L})$ would also be recursive, which implies L is Recursive Enumerable(RE).

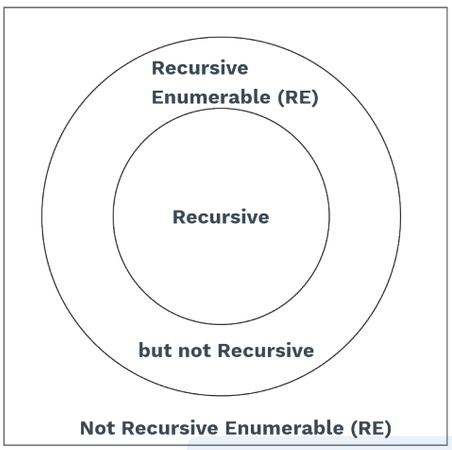


Fig. 6.4 Relationship between recursive and RE and not RE languages

Note:

The Recursive languages are closed under complement.

If L is language that is Recursive, so complement of L ($\overline{L}$) is also recursive:
Proof:

Let,

L = L(M) for some Turing Machine (M) that always halts. Now, we will construct another Turing Machine (M_1) such that:

$$\overline{L} = L(M_1)$$

where M_1 behaves just like M, we will have to modify M to create M_1:

1) All accepting states of M are changed as non-accepting states of M_1 with no transition. It means M_1 will halt without accepting.
2) M_1 has a new accepting state R; there is no transition from R.
3) For each combination of a non-accepting state of the Turing Machine (M) and a tape symbol of M such that M has no transition, add transitions to R.

M is guaranteed to halt. So, we can say M_1 will also halt. M_1 accept only those strings that M will never accept.

So, M_1 accepts the

complement of $L(\overline{L})$.

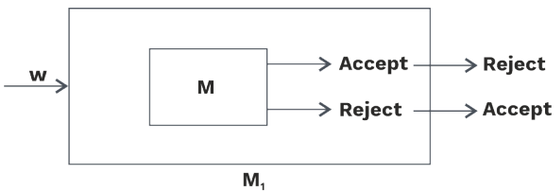


Fig. 6.5 Construction Of A Tm Accepting The Compliment Of Recursive Language.

If both the language L and its complement are Recursive Enumerable (RE), then language L is Recursive.

Proof as consider:

M_1 and M_2 are Turing Machines which are simulated in parallel by a Turing Machine (M).
Let,

$$L = L(M_1) \text{ and } \overline{L} = L(M_2)$$

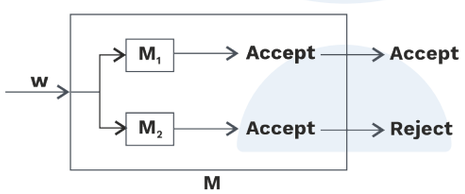


Fig. 6.6 Construction Of A Turing Maschine Accepting A Compliment Of A Recurisve Language

We can make Turing Machine (M) a two-tape Turing Machine and then it is converted into a single tape Turing Machine for simplicity.
Tape one of the Turing machines (M) simulate the tape of M_1, while tape two of the Turing machine (M) simulate the tape of M_2.
If

$$\text{Input } w \in L(M) \rightarrow M_1 \text{ will accept} \rightarrow M \text{ accepts and halts}$$

$$\text{Input } w \notin L(M) \rightarrow \overline{L} \rightarrow M_2 \text{ will accept} \rightarrow M \text{ halts without accepting}$$

Thus, for all inputs, M halts, and L(M) is the same as L. Since M always halts, and L(M) = L. So, we can say that L is Recursive.

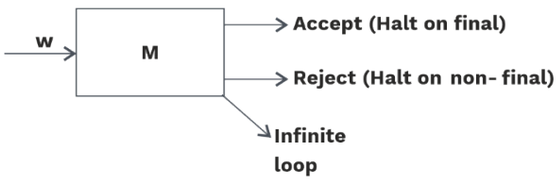
Turing machine halting problem:

"Does a Turing Machine (M) halt on input w?"

OR

"Is $w \in L(M)$?"

We know that Turing Machine has three possibilities.



If $w \in L(M)$. So, it will halt on accepting states.

If $w \notin L(M)$. So, it may halt on the non-final state, or it can get into an infinite loop.

Halt: The program/algorithm on certain input will halt on accepting state or halt on the non-final state, but it would never go into an infinite loop.

"Can we design an algorithm that tells whether the given program will halt or not?"

"The Answer will be NO". Because we cannot design any generalized algorithm which can surely say that a given program will halt or not."

Only a single way we have, just run the program and check whether it will halt or not.

So,

"Halting problem is an undecidable problem because we cannot have any algorithm that tells whether the given program will halt or not in a generalized way."

Table for decidable and undecidable problems:

Problems	FA	DPDA	PDA	LBA/HTM	TM
Halting	D	D	D	D	UD
Membership	D	D	D	D	UD
Emptiness	D	D	D	UD	UD
Finiteness	D	D	D	UD	UD
Totality	D	D	UD	UD	UD
Equivalence	D	D	UD	UD	UD
Disjoint	D	UD	UD	UD	UD
Set containment	D	UD	UD	UD	UD

Table 6.1

D $\Rightarrow$ Decidable

UD $\Rightarrow$ Undecidable

Note:

- For Regular Languages, all the given problems are decidable.
- For Recursive Enumerable Language (REL), all given problems are undecidable.

Rice theorem part-I:

- Any non-trivial property of Recursively Enumerable Language (REL) is undecidable.
- It is a negativity test. It does not prove if the language is Recursive or Recursive Enumerable.
- It Proves If a language is not recursive, it may or may not be recursively enumerable.

Definitions

Property : A property of language is simply a set of languages. We say 'L' satisfies the property 'P' is $L \in P$.

Examples:

- {<M>|M has 15 states}

This is a Property of Recursive Enumerable Language (REL) but not a language.

- {<M>|L(M) is infinite}
- {<M>|L(M) is regular.}

Trivial and non-trivial properties:

We know that set of all Turing Machines a (string of 0's and 1's) are countably infinite.

∴ Recursive Enumerable (RE) set is countably infinite like Recursive Enumerable Language (REL) set (RE_1, RE_2, RE_3, ...}

Trivial property:

A property is trivial if it is satisfied by all Recursive Enumerable Languages or it is not satisfied by any Recursive Enumerable Language (REL).

- If all the elements of the Recursive Enumerable Language (REL) set have the property → Trivial property

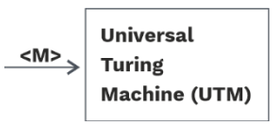
- If all the elements of the Recursive Enumerable Language (REL) set do not have the property → Trivial property

Non-trivial property:

A property is non-trivial if it is satisfied by some Recursive Enumerable Languages (REL) and are not satisfied by others.

- If at least one Recursive enumerable language (RE_i) has the property and one Recursive enumerable language (RE_i) does not have the property, then the property is said to be non-trivial.

Example:



$L = \{<M> \mid L(M) \text{ is infinite}\}$

Here, 'M' is the arbitrary Turing Machine Code which acts as an input to the Universal Turing Machine (UTM).

Let's find a Recursive Enumerable Language (RE) that has the property:

$\{L(RE_i) \text{ is infinite}\}$

RE_i_Yes = {1, 11, 111, ...}.

Turing Machine accepts languages:

$$L = \{W \mid W \in 1^*\}$$

Let's find a RE_j that does not have the property:

$\{L(RE_j) \text{ is finite}\}$

RE_j_NO = {1},

Turing Machine accepts language,

$$L = \{W \mid W \in 1\}$$

∴ Hence, we can find RE_i_YES (RE_i that has the property) and RE_i_NO(RE_i that does not have the property).

Hence, it is a Non-Trivial property of Recursively Enumerable Language. So, by Rice Theorem, we can say that the language is undecidable.

Rice theorem part-II:

Any Non-Monotonic property of Recursively Enumerable Language is not even semi-decidable.

Note:

- Recursive $\Rightarrow$ Decidable
- Semi decidable $\Rightarrow$ Recursive Enumerable but not REC.

This theorem is used for proving if the language is not semi-decidable (Not Recursive Enumerable).

Let's discuss the non-monotonic properties.

A property of recursively enumerable language is non-monotonic if any set (RE_i) having the property is PROPER SUBSET of any set (RE_j) that does not have the property.

Example:

where <M> is the arbitrary Turing Machine code.

Language of the Universal Turing Machine (UTM) can be defined as:

L(Universal Turing Machine) = {M | L(M) has at most 10 strings}

Let's see

If we can find Recursive Enumerable Language (REL_i); from REL_Set that has the property.

REL_i_YES = $\{\phi\}$ (Turing Machine accepts language $\Rightarrow$ L = $\{\phi\}$)

Can we find REL_j from REL_Set that does not have the property?

"Answer is YES"

REL_j_NO = {0, 1, 00, 000, 1111, ...0011,...}

(Turing Machine accepts language L $\Rightarrow$ $L = \left\{\Sigma^+\right\}, \Sigma = \{0,1\}$)

We can observe now,

$$REL_i \_ YES \subset REL_j \_ NO$$

So, it is a non-monotonic property.

Hence, By Rice Theorem, the language is not even semi-decidable.

Example:

L = {<M>|L(M) is infinite}

This is monotonic property since an infinite set can never be a subset of a finite set. So, we cannot apply Rice Theorem Part-II here.

We can use reduction to prove language is not Recursive Enumerable (RE).

It proves RICE Theorem Part-II is not the necessary condition for language to be NOT Recursive Enumerable (RE). It is a sufficient condition.

Reducibility:

Let us have two problems, A and B. If we have an algorithm to convert instance of problem A to instance of problem B that have the same answer, then we say

A reduces to B

or

A is polynomially reduced to B

or

A is many to one reducible to B

or

$A \leq B$

or

$A \propto_p B$

or

$A \leq_m B$

or

B is at least as hard as A.

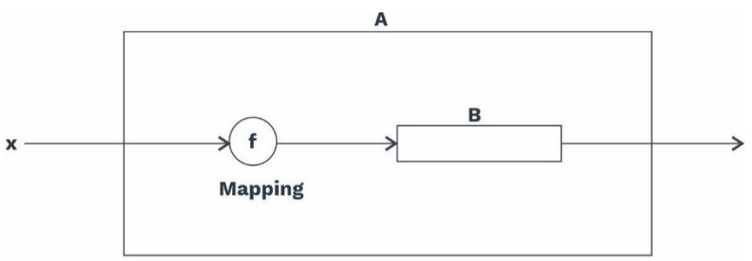


Fig. 6.7 A Is Reducible To B With The Help Of Mapping

Let, $L_1 \leq L_2$

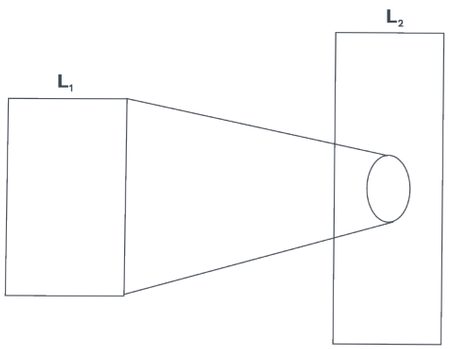


Fig. 6.8 Every Member Of L_1 Is Also Mapped To Some Member Of L_2

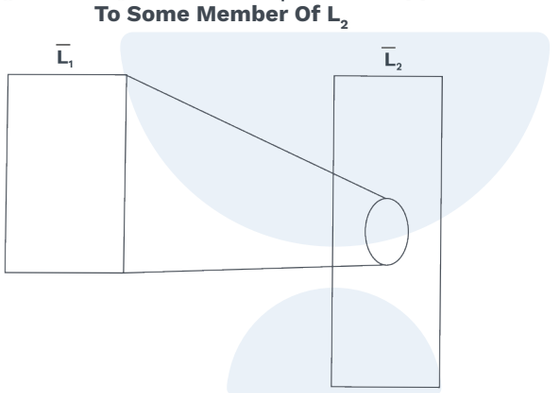


Fig. 6.9 Every Instance Of L_1 Is Mapped To Some Instance Of L,

Grey Matter Alert!

Let L_1 reducible to L_2 ($L_1 \leq L_2$)

- If L_2 is decidable, then L_1 is decidable.
- If L_1 is undecidable, then L_2 is undecidable.
- If L_2 is Recursive Enumerable (RE), then L_1 is Recursive Enumerable (RE).
- If L_1 is not Recursive Enumerable (RE), then L_2 is not Recursive Enumerable (RE).
- If $L_1 = \theta(n^2)$, then $L_2 = \Omega(n^2)$.
- If $L_2 = \theta(n^2)$, then $L_1 = O(n^2)$.

Note:

- If $A \leq B$ and $B \leq C$, then A is also reducible to C $(A \leq C)$.
- If $A \leq B$ and $C \leq B$, then
  - If B is decidable, then A and C both are decidable.
  - If A is decidable, then we cannot answer about B and C.
- If $A \leq B$ and $B \leq C$, then
  - If C is decidable, then B is decidable as well as A is decidable.
  - If B is decidable, then A is also decidable but we cannot answer about C.
  - If A is undecidable, then B and C are also undecidable.
  - If B is undecidable, then C is also undecidable.

Previous Years' Question

Choose the correct alternatives (More than one may be correct).
It is undecidable whether:

a) An arbitrary Turing machine halts after 100 steps.
b) A Turing machine prints a specific letter.
c) A Turing machine computes the products of two number.
d) None of the above.

Sol: b)

Previous Years' Question

Which of the following statements is false?

a) The Halting problem of Turing machines is undecidable
b) Determining whether a context-free grammar is ambiguous is undecidable.
c) Given two arbitrary context-free grammars G_1 and G_2 it is undecidable whether $L(G_1) = L(G_2)$
d) Given two regular grammar G_1 and G_2 it is undecidable whether $L(G_1) = L(G_2)$

Sol: d)

Previous Years' Question

Which one of the following is not decidable?

a) Given a Turing machine M, a string s and an integer k, M accepts s within k steps.

b) Equivalence of two given Turing machines.

c) Language accepted by a given finite state machine is not empty.

d) Language generated by context free grammar is non-empty.

Sol: b) **(GATE - 1997)**

Previous Years' Question

Consider the following decision problems:

(P_1) : Does a given finite state machine accept a given string?

(P_2) : Does a given context free grammar generate an infinite number of strings?

Which of the following statements is true?

a) Both (P_1) and (P_2) are decidable.

b) Neither (P_1) nor (P_2) are decidable.

c) Only (P_1) is decidable.

d) Only (P_2) is decidable.

Sol: a) **(GATE - 2000)**

Previous Years' Question

Consider the following problem X.

Given a Turing machine M over the input alphabet Σ, any state q of M and a word $w \in \Sigma$, does the computation of M on w visit the state of q?

Which of the following statements about X is correct?

a) X is decidable

b) X is undecidable but partially decidable.

c) is undecidable and not even partially decidable.

d) X is not a decision problem.

Sol: b) **(GATE - 2001)**

SOLVED EXAMPLES

Q1 **Membership problem is decidable on (More than one maybe correct option):**
a) Regular language
b) Context-free language
c) Context-sensitive language
d) Deterministic context-free language.

Sol: **a), b), c) & d)**

Explanation:
Membership problem is decidable for all languages except recursively enumerable language because all the machine always halts except Turing machine.

Q2 **Consider the following statement:**
S1: Recursive language is also called decidable language.
S2: Recursively Enumerable languages are also called as semi-decidable language.
S3: Partially decidable and semi-decidable both are the same.
Number of correct statement(s) is/are_____.

Sol: **3 to 3**

Explanation:
All the statements is correct.
Recursive language is called decidable language because, for the recursive language, we have halting Turing Machines.
For recursively enumerable language, we have a Turing machine and Turing machine may or may not halt, but TM always halts for string belongs to the language.
Hence, Recursively enumerable languages are semi-decidable.
Partially decidable and semi decidable means the same only.

Q3 **CFL are decidable on:**
a) Membership problem only **b) Emptiness problem only**
c) Finiteness Problem only **d) All the above**

Sol: **d)**

CFL are decidable on membership problem, emptiness problem, finiteness problem because we have analgorithm for all the problems.
Push down automata or CYK is analgorithm for membership problems.
For Emptiness problem algorithm is the simplification of CFG.
The Dependency tree lets us know whether the grammar is finiteness or not.
Hence, d) is the correct option.

Chapter Summary

- **Decision problem:**
  A decision is a set of related statements, each of which must be either true or false.
  i) For decidable problems, Halting Turing Machine (HTM) exists.
  ii) For undecidable problem, Halting Turing Machine (HTM) does not exist, but Turing Machine (TM) may exist.
  iii) If a problem is not recursive enumerable (RE), then no Turing Machine exists for that problem.

- **Language:**
  The set of all strings that can be generated from the grammar.

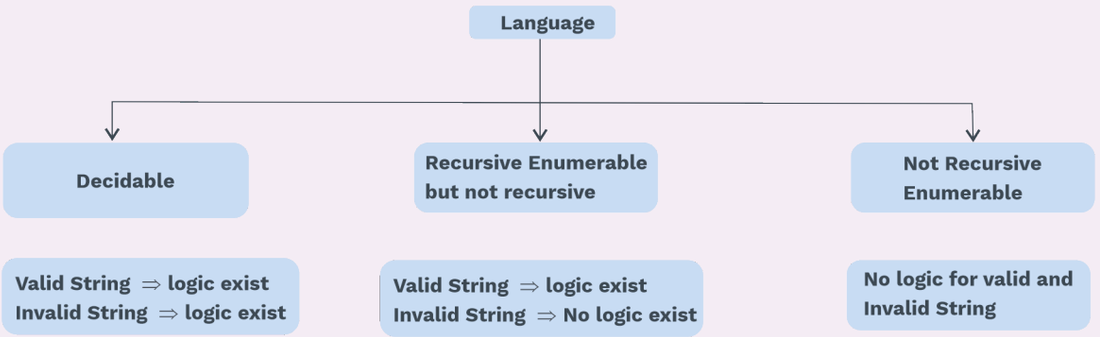


Fig. 6.10

- **Encoding a Turing Machine (TM)**
  Each Turing Machine with input alphabet {0, 1} may be thought of as a binary string.

- **The diagonalization language (L_d):**
  It is the set of strings (w_i) such that w_i is not in $L(M_i)$.

$$L_d = \{w_i \in (0+1)^+ \mid w_i \notin L(M_i)\}$$

- **An undecidable problem that is recursive enumerable (RE):**
  We have already seen that there is no Turing machine for Diagonal Language (L_d) to accept it.

- **Recursive languages:**
  A recursive language L If L = L(M) for some Turing Machine (M) such that:
  1) $w \in L \Rightarrow$ Halt and accept
  2) $w \notin L \Rightarrow$ Halt and reject

- **Compliment of recursive and recursive enumerable language (RE):**
  If a language L is Recursive Enumerable but the complement of recursive enumerable language $(\overline{L})$ is not Recursive Enumerable (RE), then we can say L can not be Recursive.

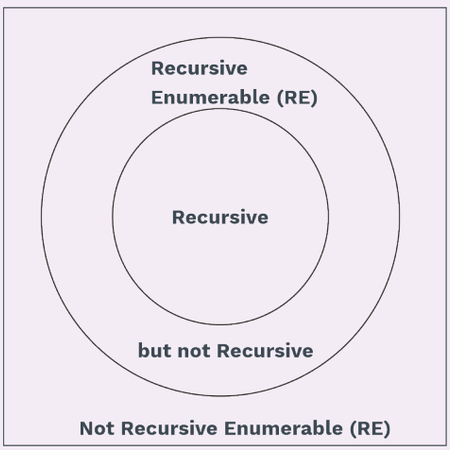
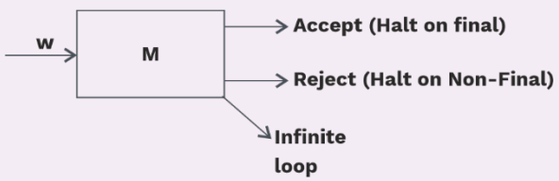


Fig. 6.11 Relationship between Recursive and Recursive Enumerable (RE)and not Recursive Enumerable (RE) language.

- **T uring machine halting problem:**
  "Does a Turing Machine (M) halt on input w?"
  OR
  "Is $w \in L(M)$?"
  We know that Turing Machine has three possibilities.



  If $w \in L(M)$. So, it will halt on accepting states.
  If $w \notin L(M)$. So, it may halt on the reject state, or it can get into an infinite loop.

- **Rice theorem part-I:**
  Any non-trivial property of Recursively Enumerable Language (REL) is undecidable.

- **Trivial property:**
  A property is trivial if either it is satisfied by all Recursive Enumerable Languages or it is not satisfied by any Recursive Enumerable Language (REL).

- **Non-trivial property:**
  A property is non-trivial if it is satisfied by some Recursive Enumerable Languages (REL) and are not satisfied by others.
- **Rice theorem part-II:**
  Any Non-Monotonic property of Recursively Enumerable Language is not even Semi-decidable.
- **Reducibility:**
  Let us have two problems, A and B. If we have an algorithm to convert an instance of problem A to instance of problem B that has the same answer, then we say A reduce to B