Let, the strings be accepted by finite state machine

$S = 0^n10^n1$

$= 0^7107^1$

$= 00000001000000001$

Dividing S into x, y and z we get,

$S = \underbrace{00}_{x} \quad \underbrace{00000}_{y} \quad \underbrace{100000001}_{z}$

Here, the above condition (ii) and (iii) are satisfied as

$|y| \geq 0$

$|xy| = 7 \leq 7$

Changing for condition 1

$S = xy^iz$ let, $i = 2$

$= xy^2z$

$= 00\ 0000000\ 100000001$

Here, , on pumping i.e. (xy^iz) the language doesn't follow patter (0^n10^n1) (i.e. first half is equal to the second half). So none of th cases follow all 3 conditions of regular language as stated b pumping lemma. Therefore language A is not regular language.

♦♦♦

Exam Solution

1. List closure properties of regular expression. If M and N are any two regular languages then show that L = (M ∪ N) is also regular language **[2075 Ashwin, Back]**

The closure properties of regular language are as follows:

(a) Union

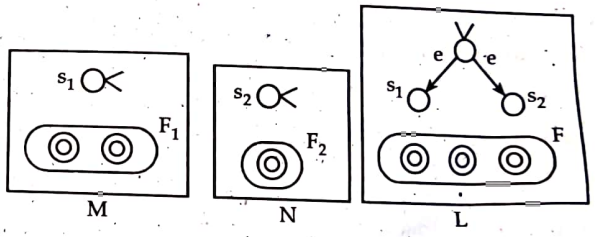
(b) Concatenation

(c) Kleene star

(d) Complementation

(e) Intersection

Let $M = (\theta_1, \Sigma, \Delta_1, S_1, F_1)$ and $N = (\theta_2, \Sigma, \Delta_2, S_2, F_2)$ be non-deterministic finite automata, we shall construct a non-deterministic finite automata such that $L = M \cup N$. The construction of L is rather simple and intuitively clear, illustrated in fig. (i). Basically, L uses non-determinism to guess whether the input is in M or in N, and then processes the string exactly as the corresponding automaton would, it follows that $L = M \cup N$. But let us give the formal details and proof for this case.



Without loss of generally, we may assume that θ_1 and θ_2 are disjoint sets. Then the finite automaton L accepts that accepts $M \cup N$ is defined as follows:

$L = (\theta, \Sigma, \Delta, S, F)$, where S is a new state not in θ_1 or θ_2

$\theta = \theta_1 \cup \theta_2 \cup \{S\}$

$F = F_1 \cup F_2$

$\Delta = \Delta_1 \cup \Delta_2 \cup \{(S, e, S_1), (S, e, S_2)\}$

That is, L beings any computation by non-deterministically choosing to enter either the initial state of M or the initial state of N and thereafter, L imitates either M or N. Hence, L accepts w if and only if M accepts w or N accepts w and $L = M \cup N$.

2. **Write statement of pumping lemma for regular languages. Show that L = {$a^n b^n$, n ≥ 0} is not a regular language by using pumping lemma.** **[2075 Ashwin, Back]**

Please refer to the theory part

3. **What is the significance of finite automata? Design a DFA that accepts the strings over an alphabet Σ = {0, 1} that either start with 01 or end with 01. Hence test your design for any two strings.** **[2075 Ashin, Back]**

The significance of finite automata are as follows:

(a) It is very useful in processing strings i.e. it can accept certain input strings and reject others.

(b) It is useful for communication protocols.

(c) It is used to parse formal languages.

(d) It is useful in the creation of compiler and interpreter techniques.

Let L(M) be the required DFA

L(M) = (θ, Σ, δ, q_0, F)

where

θ = {q_0, q_1, q_2, q_3, q_4, q_5}

Σ = {0, 1}

q_0 = {q_0}

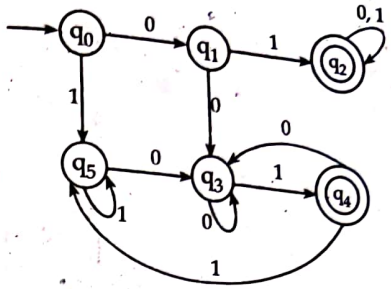
F = {q_2, q_4}

and δ is defined as follows:

Transition able

States/Σ	0	1
→ q_0	q_1	q_5
q_1	q_3	q_2
*q_2	q_2	q_2
q_3	q_3	q_4
*q_4	q_3	q_5
q_5	q_3	q_5

State diagram



Test for input string '0110'

(q_0, 0010)) ⊢M (q_1, 110)

⊢M (q_2, 10)

⊢M (q_2, 0)

⊢M (q_2, ∈) Accepted

Since, q_2 is final state, string '0110' is accepted

Test for input string '11100'

q_0, 11100) ⊢M (q_5, 1100)

⊢M (q_5, 100)

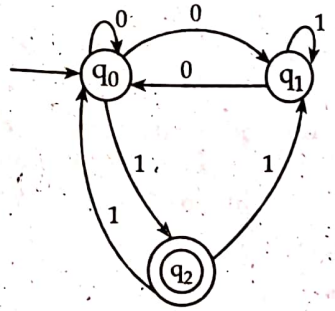
⊢M (q_5, 00)

⊢M (q_3, 0)

⊢M (q_3, ∈) Rejected

Since, q_3 is not final state, string '11100' is rejected.

4. **Differentiate between DFA and NDFA. Convert the following NDFA to its DFA.** **[2074 Ashwin, Back]**



For the differences between DFA and NDFA, see theory part.

Here, transition table of given NDFA.

States/Σ	0	1
→ q_0	$\{q_0, q_1\}$	$\{q_2\}$
q_1	q_0	q_1
* q_2	ϕ	$\{q_0, q_1\}$

Transition table of required DFA

States/Σ	0	1
→ q_0	q_A	q_2
q_A	q_A	q_B
* q_B	q_0	q_A
* q_2	q_d	q_A
q_d	q_d	q_d

Note: readers are requested to derive steps on their own using previous theory part.

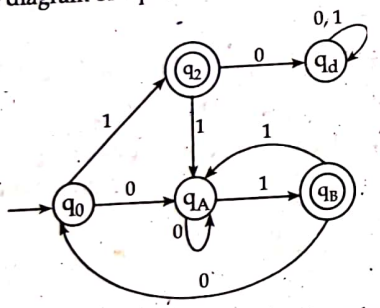
where, $q_A = \{q_0, q_1\}$

$q_B = \{q_1, q_2\}$

q_d = dead state

Here, aestrick is used to denoted final states in transition table.

State diagram of equivalent DFA,



5. What are the components of finite automata? Design a DFA that accepts the strings given by L = {w ∈ {a, b}* : w has number of a divisible by 3 and number of b divisible by 2. [2075 Ashwin Back]

The components of finite automata are:

(a) Input tape: It contains single string.

(b) Reading head: It reads input string one symbol at a time.

(c) Memory: It is in one of a finite number of states.

Let, L(M) be the required DFA

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

$\theta = \{q_0, q_1, q_2, q_3, q_4, q_5\}$

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

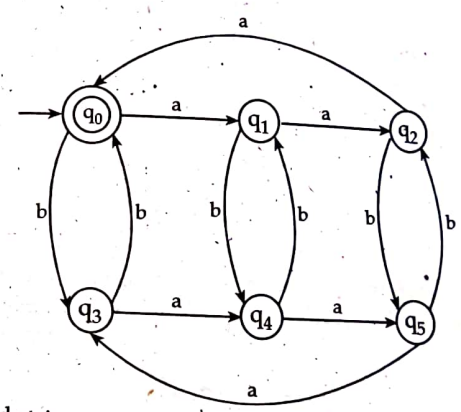
$q_0 = \{q_0\}$

$F = \{q_0\}$

and transition δ is defined as

States/Inputs	a	b
→ *q_0	q_1	q_3
q_1	q_2	q_4
q_2	q_0	q_5
q_3	q_4	q_0
q_4	q_5	q_1
q_5	q_3	q_2

States diagram



Accepted strings

$(q_0, ababa)$ $\vdash M(q_1, baba)$

$\vdash M(q_4, aba)$

$\vdash M(q_5, ba)$

$\vdash^{*} M(q_2, a)$

$\vdash M(q_0, \epsilon)$ Accepted

Since, q_0 is final state, string 'ababa' is accepted.

Test for input string 'ababbba'

$(q_0, ababbba) \vdash M(q_1, babbba)$
$\vdash M(q_4, abbba)$
$\vdash M(q_5, bbba)$
$\vdash M(q_2, bba)$
$\vdash M(q_5, ba)$
$\vdash M(q_2, a)$
$\vdash M(q_0, \in)$ Accepted

Since, q_0 is final state, string 'ababbba' is accepted.

Rejected strings

Test for input string 'bbabb'

$(q_0, bbabb) \vdash M(q_3, babb)$
$\vdash M(q_0, abb)$
$\vdash M(q_1, bb)$
$\vdash M(q_4, b)$
$\vdash M(q_1, \in)$ Rejected

Since, q_1 is not final state, so string 'bbabb' is rejected

Test for input string 'bba'

$(q_0, bba) \vdash M(q_3, ba)$
$\vdash M(q_0, a)$
$\vdash M(q_1, \in)$

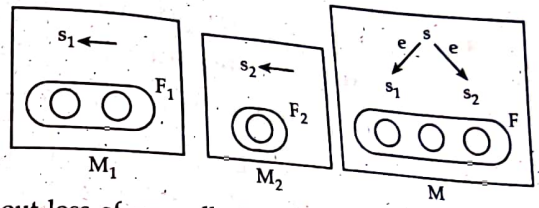
6. Define closure properties of regular language prove that regular are closed under union, intersection and complementation operation. [2074 Ashwin, Back]

The closure properties of regular language are

(a) Union
(b) Concatenation,
(c) Kleene star
(d) Complementation
(e) Intersection

To prove: Regular languages are closed under union i.e. union of two regular languages is also regular.

(a) Union: Let $M_1 = (0_1, \Sigma, \delta_1, S_1, F_1)$ and $M_2 = \{\theta_2, \Sigma, \delta_2, S_2, F_2\}$ be non-determinist finite automata. We shall construct a non-deterministic finite automata such that $L(M) = L(M_1) \cup L(M_2)$. The construction of M is rather simple and intuitively clear illustrated in fig below. Basically, M uses non-determinism to guess whether input is in $L(M_1)$ or in $L(M_2)$ and then processes the string exactly as the corresponding automata would it follows that $L(M) = L(M_1) \cup L(M_2)$. But let us give the formal details and proof for this case.



Without loss of generally, we may assume that K_1 and K_2 are disjoint sets. Thus, the finite automata M that accepts $L(M_1) \cup L(M_2)$ is defined as shown in figure above $M = \{\theta, \Sigma, \delta, S, F\}$ where S is a new state not in θ_1 and θ_2.

$$\theta = \theta_1 \cup \theta_2 \cup \{s\}$$

$$F = F_1 \cup F_2$$

$$\delta = \delta_1 \cup \delta_2 \cup \{s, e, s_1), (s, e, s_2)$$

That is M beings any computation by non deterministically choosing to enter either initial sate of M_1 or the initial state of M_2 and therefore M imitates either M_1 or M_2 thereafter. Hence M accepts w if and only if M_1 accepts w or M_2accepts w and $L(M) = L(M_1) \cup L(M_2)$. Hence, regular properties are closed under union.

(b) Complementation: Let $M = (\theta, \Sigma, \delta, q_0, F)$ be a DFA that accepts L, then DFA

$\overline{M} = (\theta, \Sigma, \delta, q_0, \theta, F)$ accepts $\overline{L}$.

We can justify it as if M accepts the language L then some of the states of this FA are final states and must likely, some are not. Let us reverse the final state of each state, that is if it was a final state, make it a non-final state and vice-versa. If an input string formally ended in a non-final state, it now ends in a final

state and vice-versa. This new machine we have built accepts all input strings that weren't accepted by the original FA (all the words in $\overline{L}$) and rejects all input strings that were accepted by original FA. Therefore, in machine accepts exactly the language $\overline{L}$. So L is also regular language.

(c) **Intersection:** From De-Margin's law, we can write

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

Because L_1 and L_2 are regular so $\overline{L_1}$ and $\overline{L_2}$ are regular and so is $\overline{L_1} + \overline{L_2}$ as we have proved that regular language is closed under union and complementation. And then because $\overline{L_1} + \overline{L_2}$ is regular then so is $\overline{\overline{L_1} + \overline{L_2}}$ which means $L_1 \cap L_2$ is regular. i.e. regular languages are closed under intersection.

7. Explain finite automata with their application. Design a DFA that accepts the language L = {w ∈ {a, b} : w must have either aaa or bbb as substring} [2074 Chaitra, Regular]

Finite automata is a abstract model of digital computer. Its mechanisms to read input strings on a input tape and either accepts it or rejects it. Some of the application of finite automata are discussed below:

(a) It is used in parsing formal languages.

(b) It is useful in creation of compiler and interpreter techniques.

(c) It is useful for communication protocols.

(d) It is used in processing input strings.

Let, the required DFA be

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

where,

$\theta = \{q_0, q_1, q_2, q_3, q_4, q_4, q_5, q_6\}$

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

$q_0 = \{q_0\}$

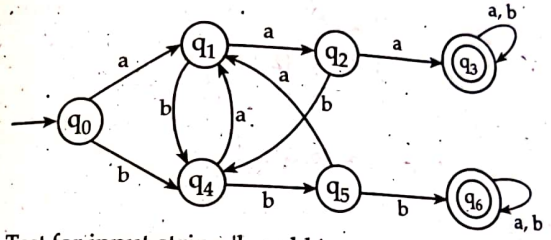
$F = \{q_3, q_6\}$

and transition δ is defined as follows:

Transition table

States/Σ	a	b
→ q_0	q_1	q_4
q_1	q_2	q_4
q_2	q_3	q_4
* q_3	q_3	q_3
q_4	q_1	q_5
q_5	q_1	q_6
* q_6	q_6	q_6

State diagram



Test for input string 'baaabb'

$(q_0, baaabb) \vdash M(q_4, aaabb)$

$\vdash M(q_1, aabb)$

$\vdash M(q_2, abb)$

$\vdash M(q_3, bb)$

$\vdash M(q_3, b)$

$\vdash M(q_3, \in)$ Accepted

Since, q_3 is final state, string 'baaabb' is accepted.

Test for input string 'bbaab'

$(q_0, bbaa) \vdash M(q_4, baab)$

$\vdash M(q_5, aab)$

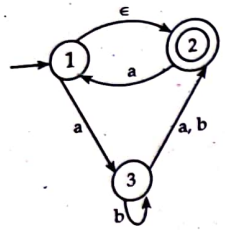
$\vdash M(q_1, ab)$

$\vdash M(q_2, b)$

$\vdash M(q_4, \in)$ Rejected

Since, q_4 is not final state, string 'bbaab' is not accepted.

8. Convert the following NFA into its equivalent DFA. [2074 Chaitra]



Here, this is ϵ-NFA

and e-closure (1) = {1, 2} = A (say)

Transition states

Step I

$\delta(A, a)$ = e-closure $(\delta (1, 2), a)$
= e-closure $(\delta (1, a) \cup \delta (2, a))$
= e-closure $(3 \cup 1)$
= e-closure (3) ∪ e-closure (1)
= {3} ∪ {1, 2}
= {1, 2, 3}
= B (say)

$\delta(A, b)$ = e-closure $(\delta (1, 2), b)$
= e-closure $(\delta (1, b) \cup \delta (2, b))$
= e-closure $(\phi \cup \phi)$
= e-closure (ϕ)
= C (say) (dead state)

Step II

$\delta(B, a)$ = e-closure $(\delta (1, 2, 3), a)$
= e-closure $(\delta (1, a) \cup \delta (2, a) \cup \delta(3, a))$
= e-closure $(3 \cup 1 \cup 2)$
= e-closure (3) ∪ e-closure (1) ∪ e-closure (2)
= {3} ∪ {1, 2} ∪ {2}
= {1, 2, 3}
= B

$\delta(B, b)$ = e-closure $(\delta (1, 2, 3), b)$
= e-closure $(\delta (1, b) \cup \delta (2, b) \cup (3, b))$
= e-closure $(\phi \cup \phi \cup \{3, 2\})$
= e-closure (3) ∪ e-closure (2)
= {2} ∪ {3} = {2, 3}
= D (say)

$\delta(D, a)$ = e-closure $(\delta (2, 3), a)$
= e-closure $(\delta (2, a) \cup \delta (3, a))$
= e-closure $(1 \cup 2)$
= e-closure (1) ∪ e-closure (2)
= {1, 2} ∪ {2}
= {1, 2}
= A

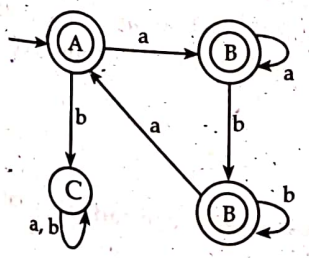
$\delta(D, b)$ = e-closure $(\delta (2, 3), b)$
= e-closure $(\delta (2, b) \cup \delta (3, b))$
= e-closure $(\phi \cup \{3, 2\})$
= e-closure (3) ∪ e-closure (2)
= {3} ∪ {2}
= {2} ∪ {3}
= D

e-closure $\delta(c, a)$ = e-closure $\delta(c, b)$ = c

Transition table

States/Σ	a	b
→ * A	B	C
C	C	C
* B	B	D
* D	A	D

State diagram



Note: Final states of DFA are all those new states which contains final states of NFA with ϵ-transitions.

9. State the pumping lemma for the regular languages. Show that the languages

$L = \{0^{n^2} \mid n >= 1\}$ not regular example,

if x = 1, w = 0, n = 2, w = 0000, n = 3, w = 000000000 [2074, Chaitra]

Pumping lemma states that if a language A is regular then it must have string $|s| \geq P$ where P is pumping length and can be divided into xyz such that

(a) $xy^iz \in A$ for some i

(b) $|y| > 0$

(c) $|xy| \leq P$

Here, given language is $L = \{0^{n^2}\} / n >= 1\}$

To prove: L is not regular

Proof: Using pumping lemma

Let's assume language L is regular.

Then, if must have pumping length P.

Let P = 3 (say)

Then, let s be the string accepted by finite automaton.

i.e. $s = o^{P^2}$

$= 0^{3^2}$

$= 0^9$

$= 000000000$

Dividing s into x, y and z as

$\underbrace{0}_{x}\ \underbrace{00}_{y}\ \underbrace{000000}_{z}$

Checking for condition 1

$s = xy^iz = xy^2z$ (e = 2 (say))

$= 0\ 0000\ 000000$

Now, $|s| = 11$

Here, no value of P(or n) would give (s) = 1 in other words, length of string s on being pumped is not perfect square which means it doesn't belong to language L so, condition 1 fails. Though, condition 2 and 3 are satisfied.

$|y| = 2 > 0$

$|xy|\ 3 \geq P$

Since, condition 1 is failed, our assumption contradicts. That's why language L is not regular.

10. Differentiate between NFA and DFA. Design a DFA that accepts the language $L = \{x : x \in \{0, 1\}^* \mid 10110$ doesn't occur as substring in x}. Verify your design with supporting examples. [2073 Chaitra]

For the difference between NFA and DFA, see theory part.

Let, L(M) be the required DFA

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

where,

$\theta = \{q_0, q_1, q_2, q_3, q_4, q_5\}$

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

$q_0 = \{q_0\}$

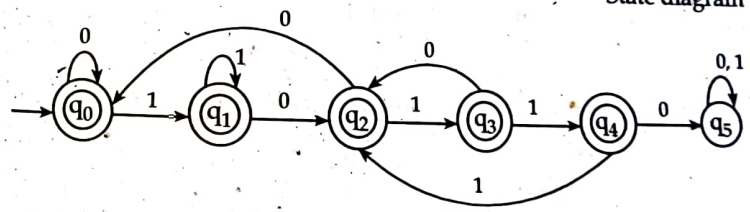
$F = \{q_0, q_1, q_2, q_3, q_4\}$

and δ is defined as

Transition table

States/Σ	0	1
→ * q_0	q_0	q_1
* q_1	q_2	q_1
* q_2	q_0	q_3
* q_3	q_2	q_4
* q_4	q_5	q_1
q_5	q_5	q_5

State diagram



Test for input string '1101100'

$(q_0, 1101100) \vdash M(q_1, 101100)$

$\vdash M(q_1, 01100)$

$\vdash M(q_2, 1100)$

$\vdash M(q_3, 100)$

$\vdash M(q_4, 00)$

$\vdash M(q_5, 0)$

$\vdash M(q_5, \in)$ Rejected

Sine, q_5 is not final state and string contains '10110' as substring. So, it is rejected.

Test for input string '11100'

$(q_0, 11100) \vdash M(q_1, 1100)$
$\vdash M(q_1, 100)$
$\vdash M(q_1, 00)$
$\vdash M(q_2, 0)$
$\vdash M(q_0, \epsilon)$

Since, q_0 is one of the final states string '11100' is accepted.

11. State pumping lemma for regular language. Use pumping lemma and prove that language L = {w | w ∈ {0, 1}* and w has an equal number of 0's and 1's} is not regular. [2073 Chaitra]

For statement of pumping lemma for regular language, please see theory part.

Proof: Let, assume that language L is regular. Now, since it is regular, it must follow these conditions:

(a) $xy^iz \in L$ for every $i \geq 0$

(b) $|y| > 0$

(c) $|xy| \leq P$ where P is pumping length

Lent, $S \in L = 01010101$ and $P = 3$

Now, dividing S into x, y and z, we get

$\underbrace{0}_{x}\ \underbrace{10}_{y}\ \underbrace{10101}_{z}$

Here, the above condition (ii) and (iii) are satisfied as

$|y| = 2 > 0$

$|xy| = 2 \leq 3$

Now, checking for condition 1,

xy^iz

Let, $l = 2$

xy^2z

$= 01010\ 10101$

Then, no of 0's is not equal to number of 1's so, it doesn't belong to L and condition 1 fails. If any of the above condition fails, then our assumption is also false which means language L is not regular.

12. Why is NDFA important although it is equivalent to a DFA? Design NDFA which accepts L {w | w ∈ {a, b}* such that w contains either two consecutive a's or two consecutive b's} [2073 Chaitra]

NDFA (non-deterministic finite automata) is important although it's equivalent to DFA (Deterministic Finite Automata) because of following reasons:

(a) NDFA's are easy to implement and to construct than DFA's.

(b) For choosing the right path in implementation of a particular algorithm, usually NFAs to DFAs conversion works.

(c) NDFAs can be used to reduce the complexity.

(d) NDFAs can be used to reduce the complexity of the mathematical work.

(e) It is much easier to prove closure properties of regular language using NDFAs than DFAs.

Let, the required NDFA be

$L(M) = (\theta, \Sigma, \delta, q_0, E)$

where,

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

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

$q_0 = \{A\}$

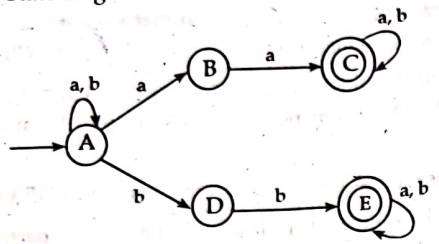
$F = \{C, E\}$

and δ is defined as follows:

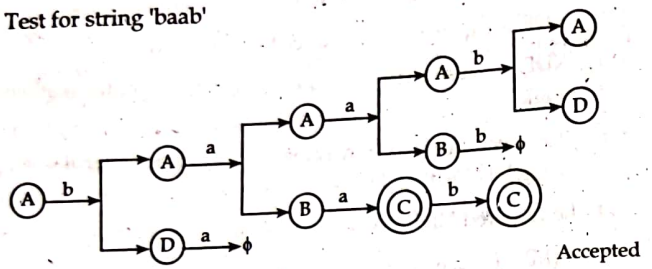
States/Σ	a	b
A	{A, B}	{A, D}
B	C	ϕ
C	C	C
D	ϕ	E
E	E	E

In transition table, some states have multiple states to go example A and some states have nowhere to go on input like (B and D). Therefore, this represents NDFA.

State diagram



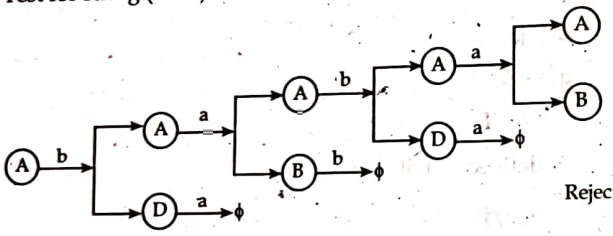
Test for string 'baab'



Accepted

Since, one of the final state is acceptor state i.e. C it is accepted.

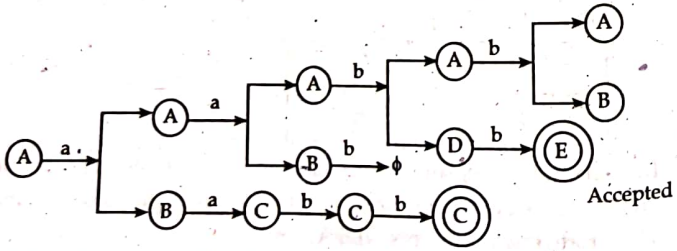
Test for string (baba)



Rejected

Hence, none of final state is acceptor state, so string 'baba' is rejected.

Test for string 'aabb'



Accepted

This is the required NDFA

13. For every NDFA, there exists a DFA which simulates the behaviour of NDFA. Alternative if L is the set accepted by NDFA. Then exists a DFA which also accepts L.

Let $M = \{Q, \Sigma, \delta, q_0, F\}$ be q NDFA accepting language L. We construct a DFA M' as follows.

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

where,

(a) $Q' = 2^\theta$ (any state in Q is denoted by $\{q_1, q_2, \ldots q_i\}$ where $q_1, q_2, \ldots q_j \in Q$)

(b) $q'_0 = [q_0]$

(c) F' is the set of all subsets of Q containing an element of F.

(d) $(\delta' (q_1, q_2, \ldots, q_i), a) = \delta (q_1, a) \cup \delta(q_2, a) \ldots \cup \delta(q_i, a)$ equivalently $\delta' (q_1, q_2, q, j, q) = [p_1, p_2, p_j]$ if and only if $\delta[\{q_1, q_2, \ldots q_i\}, q] = \{p_1, p_2, \ldots p_j\}$ if and only if $\delta(q_0, x) = \{q_1, \ldots q_i\}$ for all x in Σ

First we are going to prove by induction that $\delta^* (q_0, w) = \delta'^* (q'_0, w)$ for any string w, when it is proven, it obviously implies that NDA M and DFA M' accept the same strings.

Now, we have q task to prove $\delta^* (q_0, w) = \delta'^* (q'_0, w)$. This is going to be proven by induction on w.

Note: δ^* represents the transition function for processing strings.

Basic step: For w = t

$\delta'^* = (q'_0, t) = q'_0$ by the definition d'*.

$= q_0$ by construction of DFA M'.

$= \delta^* (q_0, t)$ by the definition of δ^*.

Inductive step:

Assume that $\delta^* (q_0, w) = \delta'^* (q'_0, w)$ for an arbitrary string w ... Induction hypothesis.

For string w and arbitrary symbol a in Σ

$\delta^* (q_0, wa) = U_{PE}\, \delta^* (q_0, w)\, \delta(p, a)$

$= \delta' (\delta^* (q_0, w), a)$

$= \delta' (\delta'^* (q'_0, w), a)$

$= \delta^* (q'_0, wa)$

Thus, for any string w $\delta^* (q_0, w) = \delta'^* (q'_0, w)$ holds. So, we can say that L(M) = L(M'). i.e. To every NDFA, there exists an equivalent DFA.

14. Construct a NFA for the language (ab* a ∪ b*aa) provide any two accepted string and two rejected strings. [2073 Shrawan]

Let, the required NFA be

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

where,

$\theta = \{q_0, q_1, q_2, q_3, q_4, q_5\}$

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

$q_0 = \{q_0\}$

$F = \{q_2, q_5\}$

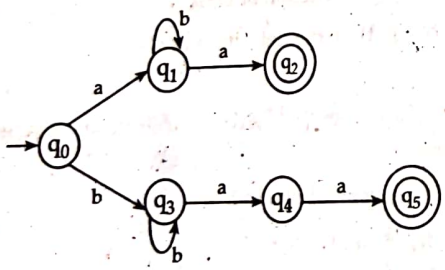
and transition δ is defined as

Transition table (δ)

States/Σ	a	b
→ q_0	q_1	q_3
q_1	q_2	q_1
* q_2	ϕ	ϕ
q_3	q_4	q_3
q_4	q_5	ϕ
* q_5	ϕ	ϕ

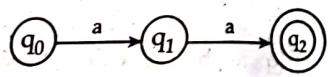
As we can see there are no transition state for some inputs on some states. So, it is NFA.

State diagram



Accepted strings

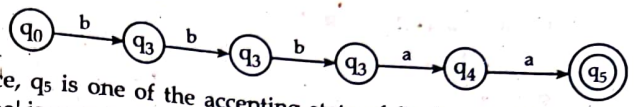
Test for string 'qq'



Accepted

Here, b* means b can be empty (∈). So, string aa can be generated from (ab* a ∪ b*aa). And since, q_2 is accepting state of final state, string 'aa' is accepted.

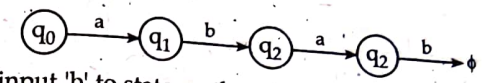
Test f. string 'bbbaa'



Hence, q_5 is one of the accepting state of final state. Hence, string 'bbbaa' is accepted.

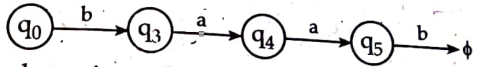
Rejected strings

Test for string 'abab'



On input 'b' to state q_2, there is no transition state defined. So, string 'abab' is rejected.

Test for string 'baab'



Similarly, on input 'b' to state q_2, there is no transition state defined. So, string 'baab' is rejected.

15. Define configuration of DFA design a deterministic. Finite automata (DFA) for language L = {w ∈ {0, 1} : w has both 01 and 10 as substring} verify your design by taking one accepted and one rejected strings. [2073 Shrawan]

The general configuration of DFA is

$(q, w) \in \theta \times \Sigma^*$

where,

q ⟶ initial state

w ⟶ string

θ ⟶ set of states of finite automata

The derivable configuration of DFA is

$(q, w) \vdash M(q', w')$

where

q ⊢ initial state

w ⊢ initial string

q' ⊢ transition state

w' ⊢ new string

M ⊢ finite state machine

Let, the required DFA be

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

where

$\theta = \{q_0, q_1, q_2, q_3, q_4, q_5, q_6\}$

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

$q_0 = \{q_0\}$

$F = \{q_3, q_6\}$

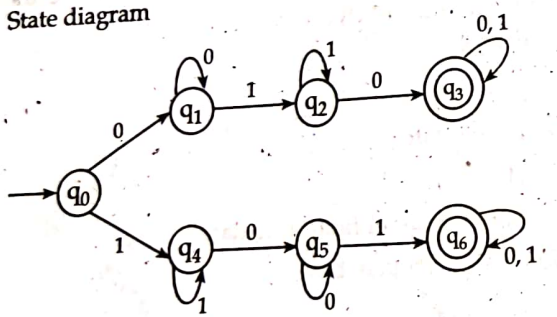
and transition δ is defined as follows:

Transition table

States/Σ	0	1
→ q_0	q_1	q_4
q_1	q_1	q_2
q_2	q_3	q_2
* q_3	q_3	q_3
q_4	q_5	q_4
q_5	q_5	q_6
* q_6	q_6	q_6

Let us analyze the language of given DFA it generates following strings {010, 101, 01011, 00110, ...}

State diagram



Test for input string '01100'

$(q_0, 01100) \vdash M(q_1, 1100)$

$\vdash M(q_1, 100)$

$\vdash M(q_2, 00)$

$\vdash M(q_3, 0)$

$\vdash M(q_3, \epsilon)$ Accepted

Since, string '01100' contains both '01' and '10' as substring, it is accepted.

Test for input string '0011'

$(q_0, 0011) \vdash M(q_1, 011)$

$\vdash M(q_1, 11)$

$\vdash M(q_2, 1)$

$\vdash M(q_2, \epsilon)$ Rejected

Since, string '0011' contains '01' as substring but doesn't contain '10' as substring, so it is rejected.

16. State pumping lemma, for regular language and use the theorem to prove that $L = \{a^n b^{2n} : n \geq 1\}$ is not regular. [2073, Shrawan]

If a language L is a regular language, then it must have a pumping length 'P' such that any string $S \in L$ having length $|S| \geq P$ can be divided into 3 parts such that following conditions are true.

(a) xy^iz for every $i \geq 0$

(b) $|y| > 0$

(c) $|xy| \leq p$

Given, $L = \{a^n b^{2n} : n \geq 1\}$

To prove: L is not regular

Proof: We prove it by contradiction using pumping lemma.

Let's assume language L is regular having pumping length 'P'.

Let, $S \in L = a^p b^{2p}$

Let, P = 4 (say)

Then, $S = a^4 b^8$

$= aaaabbbbbbbb$

Dividing S into x, y and z, we get

Case – I: When y contains m' y 'a'

$S = \underbrace{aa}_{x}\ \underbrace{aa}_{y}\ \underbrace{bbbbbbbb}_{z}$

Checking for condition I

Let, $i = 2$

$S = xy^2z$

$= aa\ aaaa\ bbbbbbbb$

$= a^6b^8 \notin L$

Here, condition 1 fails as 'b' is not as twice as 'a'.

Case – II: Why y contains both kind of symbols 'a' and 'b' i.e.

$S = \underbrace{aa}_{x}\ \underbrace{aaa}_{y}\ \underbrace{bbbbbbb}_{z}$

Checking for condition I

Let, $i = 2$

$S = xy^2z$

$= aa\ aab\ aab\ bbbbbbb \notin L$

Again, condition 1 fails as it doesn't follow pattern

Case – III: When y contains only b's i.e.

$S = \underbrace{aaaa}_{x}\ \underbrace{bbbb}_{y}\ \underbrace{bbbb}_{z}$

Checking for condition I

Let, $i = 2$

$S = xy^2z$

$= aaaa\ bbbbbbbbbbbb$

$= a^4 b^{12} \notin L$

Again, condition 1 fails

Here, in pumping (i.e. xy^iz) doesn't follow condition 1. So, none of the cases follow all three condition of regular language as stated by pumping lemma. Therefore, language L is not regular language.

17. Define DFA formally sate and prove closure properties of regular languages. **[2072, Chaitra]**

For formula definition of DFA see theory part.

for statement and proof of closure properties of regular language see 2074, Ashwin, Back part.

18. Define pumping lemma for regular languages. Use pumping lemma for regular language to show L [a^nba^n for n = 0, 1, 2, ...] is not regular. **[2072, Chaitra]**

Pumping lemma states that if a language L is regular then it must have $|s| \geq p$ where p is pumping length and s is the string, $S \in L$ which can be divided into x, y and z parts such that following conditions are true.

(1) $xy^iz \in L$ for every $i \geq 0$

(2) $|y| > 0$

(3) $|xy| \leq p$

To prove: L = {a^nba^n for n = 0, 1, 2, ...} is not regular.

Proof: Let's assume language L is regular. Then it must have pumping length 'p'.

Let, $S \in L = q^pba^p$

Let, $p = 3$

$S = a^3ba^3$

$aaabaaa$

Dividing S into x, y and z, we get

Case – I: When 'y' contains only 'a'

$S = \underbrace{a}_{x}\ \underbrace{aa}_{y}\ \underbrace{baaa}_{z}$

Checking for condition I

Let, $i = 2$

$S = xy^2z$

$= a\ aaaa\ bbaaa$

$= a^5ba^3 \notin L$

So, condition 1 fails as the power of a's at the start and end of the string are not equal.

Case – II: When y contains both symbol 'a' and 'b'

$S = \underbrace{aa}_{x}\ \underbrace{ab}_{y}\ \underbrace{aaa}_{z}$

Checking for condition I

Let, $i = 2$

$S = xy^2z$

$= aa\ abab\ aaa \notin L$

So, condition 1 fails as it doesn't follow pattern $a^n\ ba^n$.

Hence, on pumping (i.e. xy^iz), none of the cases follow all three conditions of regular language as stated by pumping lemma. Therefore, language L is not regular.

19. Construct a DFA over {a, b} accepting strings having even number of 'a' and odd number of 'b'. **[2072 Chaitra]**

Let, the required DFA be

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

where,

$\theta = \{q_0, q_1, q_2, q_3\}$

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

$q_0 = \{q_0\}$

$F = \{q_3\}$

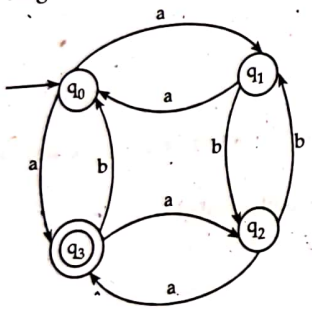
and transition δ is defined as

Transition table (δ)

States/Σ	a	b
$\rightarrow q_0$	q_1	q_3
q_1	q_0	q_2
q_2	q_3	q_1
$*\ q_3$	q_2	q_0

Let us analyse the language. The given language L generates to strings like {b, bbb, aab, aabbb, ...}

State diagram



Test for string 'bbbaa'

$(q_0, bbbaa) \vdash M(q_3, bbaa)$

$\vdash M(q_0, baa)$

$\vdash M(q_3, aa)$

$\vdash M(q_2, a)$

$\vdash M(q_3, \epsilon)$ Accepted

Since, q_3 is final state, string 'bbbaa' is accepted.

Test for input string' aabb'

$(q_0, aabb) \vdash M(q_1, abb)$

$\vdash M(q_0, bb)$

$\vdash M(q_3, b)$

$\vdash M(q_0, \epsilon)$ Rejected

Since, q_0 is not final state, string 'aabb' is rejected.

20. Design a deterministic finite automata, (DFA) for the regular expression (a (ab)* b)* verify your design by taking one accepted and one rejected strings. [2071 Chaitra]

Let, the required DFA be

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

where,

$\theta = \{q_0, q_1, q_2, q_d\}$

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

$q_0 = \{q_0\}$

$F = \{q_0\}$

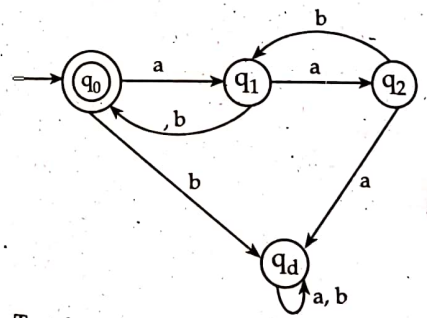
and transition δ is defined as follows

Transition table (δ)

States/Σ	a	b
$\rightarrow *\ q_0$	q_1	q_d
q_1	q_2	q_0
q_2	q_d	q_1
q_d	q_d	q_d

Let us analyse the language. The given language L generates to strings like {ϵ, ab, abab, aabb, aababb, aabbaabb, ...}

State diagram



Test for string 'aababb'

$(q_0, aababb) \vdash M(q_1, ababb)$

$\vdash M(q_2, babb)$

$\vdash M(q_1, abb)$

$\vdash M(q_2, bb)$

$\vdash M(q_1, b)$

$\vdash M(q_0, \epsilon)$ Accepted

Since, q_0 is final state, string 'aababb' is accepted.

Test for input string 'aabbb'

$(q_0, aabbb) \vdash M(q_1, abbb)$
$\vdash M(q_2, bbb)$
$\vdash M(q_1, bb)$
$\vdash M(q_0, b)$
$\vdash M(q_d, \in)$ Rejected

Since, q_d is dead state or not final state, string 'aabbb' is rejected.

21. State pumping lemma for regular language. Use this lemma to prove language L : $\{a^{n^2} : n \geq 0\}$ is not regular. **[2071 Chaitra]**

For statement of pumping lemma, see theory part.

Here, given language $L = \{a^{n^2}, n \geq 0\}$

To prove: L is not regular

Proof: Let's assume language L is regular.

Then, it must have pumping length P.

Let, P = 3 (say)

Then, let S be the string accepted by finite automata.

i.e. $S = a^{P^2}$

$= a^{3^2}$

$= a^9$

Dividing S into x, y and z, we get

$S = \underbrace{a}_{x}\ \underbrace{aa}_{y}\ \underbrace{aaaaaa}_{z}$

Now, checking for condition I

$S = xy^iz$

Let, i = 2

$S = xy^2z$

= a aaaa aaaaaa

Now, |S| = 11

Here, no value of P (or n) would give |S| = 11 in other words, length of string S an being pumped is not perfect square which means it doesn't belong to language L.

So, condition 1 fails.

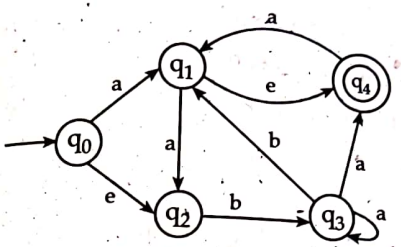
But, condition (2) and (3) are stratified.

i.e. $|y| = 2 > 0$

$|xy| = 3 \geq P$

Since, condition 1 fails, our assumption contradicts. That's why language L is not regular.

22. What are the differences between a DFA and a NFA? convert the following NFA into its equivalent DFA. **[2071 Chiaitra]**



For the differences between a DFA and q NFA se3e theory part.

Here, e-closure $(q_0) = \{q_0, q_2\} = A$ (say)

e-closure $(q_1) = \{q_1, q_4\} = B$ (say)

Transition states

$\delta(A, a)$ = e-closure $(\delta(q_0, q_2), a)$
= e-closure $(\delta(q_0, a) \cup \delta(q_2, a))$
= e-closure $(q_1 \cup \phi)$
= e-closure (q_1)
= $\{q_1, q_4\}$
= B

$\delta(A, b)$ = e-closure $(\delta(q_0, q_2), b)$
= e-closure $(\delta(q_0, b) \cup \delta(q_2, b))$
= e-closure $(\phi \cup q_3)$
= q_3

$\delta(B, a)$ = e-closure $(\delta(q_1, q_0), a)$
= e-closure $(\delta(q_1, a) \cup \delta(q_4, a))$
= e-closure $(q_2 \cup \phi)$
= e-closure (q_2)
= $\{q_2\}$

$\delta(B, b)$ = e-closure $(\delta(q_1, q_4), b)$
= e-closure $(\delta(q_1, b) \cup \delta(q_4, b))$
= e-closure $(\phi \cup q_1)$
= e-closure (q_1)
= $\{q_1, q_4\}$
= B

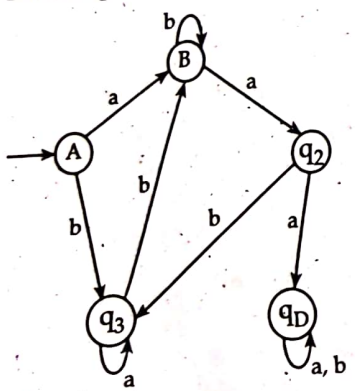
$\delta(q_2, a) = \phi = q_D$ (dead state)
$\delta(q_2, b)$ = e-closure $(\delta(q_2, b))$ = e-closure $(q_3) = q_3$
$\delta(q_3, a)$ = e-closure $(\delta(q_3, a))$ = e-closure $(q_3) = q_3$
$\delta(q_3, b)$ = e-closure $(\delta(q_3, b))$ = e-closure $(q_1) = B$
$\delta(q_D, a) = \delta(q_D, b) = q_D$

Now, transition states are

States/Σ	a	b
→ A	B	q_3
* B	q_2	B
q_2	q_D	q_3
q_3	q_3	B
q_D	q_D	q_D

Note: The final states of equivalent DFA are all those new states which contain final state of NFA with ∈-transition as member.

State diagram



This is the required equivalent DFA.

Determine a Deterministic Finite Automata (FA).

[2072, Kartik, Back]

See formal definition of DFA in theory part.

24. Design a DFA accepting strings over the alphabet {0, 1} defined by {0 0}* {1 1}*.

[2072 Kartik, Back]

Let, L(M) be the required DFA.

Let us anlayse the language. This language generates strings like {00, 11, 0011, 1111, 0000, 0000111, ...}.

i.e. even number of 0's followed by even number of b's.

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

where, $\theta = \{q_0, q_1, q_2, q_3, q_D\}$

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

$q_0 = \{q_0\}$

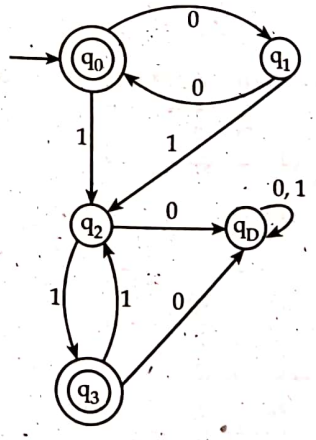
$F = \{q_0, q_3\}$

and transition δ is defined as follows

Transition table

States/Σ	0	1
→ * q_0	q_1	q_2
q_1	q_0	q_2
q_2	q_D	q_3
* q_3	q_D	q_2
q_D	q_D	q_D

State diagram



Test for input string '0011'

$(q_0, 0011) \vdash M(q_1, 011)$
$\vdash M(q_0, 11)$
$\vdash M(q_2, 1)$
$\vdash M(q_3, \in)$ Accepted

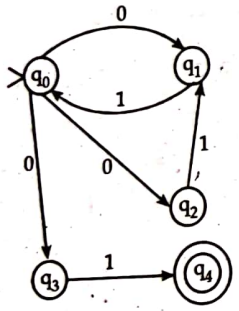
Since, q_3 is final state, string '0011' is accepted.

Test for input string '001100'

$(q_0, 001100) \vdash M(q_1, 01100)$
$\vdash M(q_0, 1100)$
$\vdash M(q_2, 100)$
$\vdash M(q_3, 00)$
$\vdash M(q_D, 0)$
$\vdash M(q_D, \in)$ Rejected

Since, q_D is dead state, string '001100' is not accepted i.e. rejected.

25. Convert the following non-deterministic finite automaton to DFA. **[2072 Kartik, Back]**



Transition table of given NDFA (non-deterministic finite automata)

States/Σ	0	1
q_0	$\{q_1, q_2, q_3\}$	ϕ
q_1	ϕ	q_0
q_2	ϕ	q_1
q_3	ϕ	q_4
q_4	ϕ	ϕ

Transition state

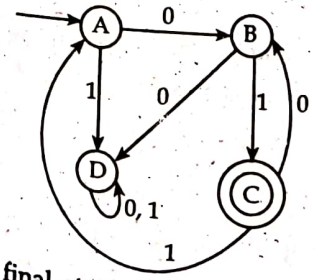
$\delta(q_0, 0) = \{q_1, q_2, q_3\}$
$\delta(q_0, 1) = \phi$
$\delta(\{q_1, q_2, q_3\}, 0) = \delta(q_1, 0) \cup \delta(q_2, 0) \cup \delta(q_3, 0)$
$= \phi \cup \phi \cup \phi$
$= \phi$
$\delta(\{q_1, q_2, q_3\}, 1) = \delta(q_1, 1) \cup \delta(q_2, 1) \cup \delta(q_3, 1)$
$= \{q_0\} \cup \{q_1\} \cup \{q_4\}$
$= \{q_0, q_1, q_4\}$
$\delta(\{q_0, q_1, q_4\}, 0) = \delta(q_0, 0) \cup \delta(q_1, 0) \cup \delta(q_4, 0)$
$= \{q_1, q_2, q_3\} \cup \phi \cup \phi$
$= \{q_1, q_2, q_3\}$
$\delta(\{q_0, q_1, q_4\}, 1) = \delta(q_0, 1) \cup \delta(q_1, 1) \cup \delta(q_4, 1)$
$= \phi \cup \{q_0\} \cup \phi$
$= \{q_0\}$

Let, $\{q_0\} = A$
$\{q_1, q_2, q_3\} = B$
$\{q_0, q_1, q_4\} = C$
$\phi = D$ (dead state)

Transition table of equivalent DFA.

States/Σ	0	1
→ A	B	D
B	D	C
* C	B	A
D	D	D

State diagram



The final states of equivalent DFA are all those new states which contains final states of NDFA.

26. State the pumping lemma for regular language show that the language L = {a^n : n is prime} is not regular using the pumping lemma. [2072 Kartik, Back]

For statement of pumping lemma, see theory part.

To prove: L = {a^n : x is a prime} is not a regular

Proof: Let's assume that language L is regular. Then, according to pumping lemma, it must follow these conditions.

(1) $xy^iz \in L$ for every $i \geq 0$

(2) $|y| > 0$

(3) $|xy| \leq P$

Since, language L is regular, it must have pumping length P.

Let P = 7

Let, $S \in L = a^P$

$= a^7$

= aaaaaaa

Dividing S into x, y and z, we get

S = a | aa | aaaa

x y z

Checking for condition I

Let i = 2

Then, $S = xy^2z$

= a aaaa aaaa

$= a^9 \notin L$

Since, 9 is not a prime number. Therefore, on pumping i.e. xy^iz, condition 1 fails. Though condition (2) and (3) are satisfied.

$|y| = 2 > 0$

$|xy| = 3 < 7$

Here, assuming language L as regular language doesn't follow all three conditions of pumping lemma to be regular language. So, it contradicts our assumption. Therefore, language L is not regular.

27. Design a DFA that accepts the language given by L = {w ∈ {0, 1}* : w beings with 0 and ends with 10}. Your design should accept strings like 010, 011110, 000010, 01011010, and should not accept strings like 1010, 0011, 01011. [2071 Shrawan, Back]

Let, the required DFA be

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

where,

$\theta = \{q_0, q_1, q_2, q_3, q_D\}$

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

$q_0 = \{q_0\}$

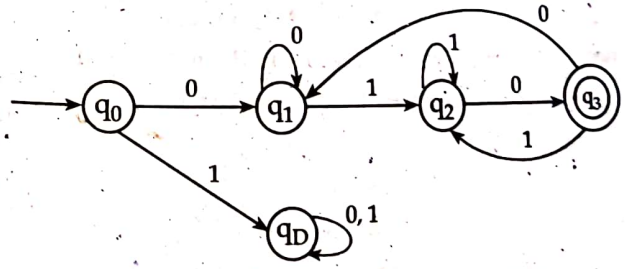
$F = \{q_3\}$

and transition δ is defined as follows

Transition table

States/Σ	0	1
→ q_0	q_1	q_D
q_1	q_1	q_2
q_2	q_3	q_2
* q_3	q_1	q_2
q_D	q_D	q_D

State diagram



Test for accepted string 010, 011110, 000010, 01011010

$(q_0, 010) \vdash M(q_1, 10)$

$\vdash M(q_2, 0)$

$\vdash M(q_3, \epsilon)$ Accepted

Since, q_3 is final state, so, string '010' is accepted.

$(q_0, 011110) \vdash M(q_1, 11110)$
$\vdash M(q_2, 1110)$
$\vdash M(q_2, 110)$
$\vdash M(q_2, 10)$
$\vdash M(q_2, 0)$
$\vdash M(q_3, \in)$ Accepted

Since, q_3 is final state, so string '011110' is accepted

$(q_0, 000010) \vdash M(q_1, 00010)$
$\vdash M(q_1, 0010)$
$\vdash M(q_1, 010)$
$\vdash M(q_1, 10)$
$\vdash M(q_2, 0)$
$\vdash M(q_3, \in)$ Accepted

Since, q_3 is final state, so string '000010' is accepted

$(q_0, 01011010) \vdash M(q_1, 1011010)$
$\vdash M(q_2, 011010)$
$\vdash M(q_3, 11010)$
$\vdash M(q_2, 1010)$
$\vdash M(q_2, 010)$
$\vdash M(q_3, 10)$
$\vdash M(q_2, 0)$
$\vdash M(q_3, \in)$ Accepted

Since, q_3 is final state, so string '01011010' is accepted

Test for rejected strings 1010, 0011, 01011

$(q_0, 1010) \vdash M(q_D, 010)$
$\vdash M(q_D, 10)$
$\vdash M(q_D, 0)$
$\vdash M(q_D, \in)$ Rejected

Since, q_3 is dead state, so string '1010' is rejected.

$(q_0, 0011) \vdash M(q_1, 011)$
$\vdash M(q_1, 11)$
$\vdash M(q_2, 1)$
$\vdash M(q_2, \in)$ Rejected

Since, q_2 is not final state, so string '0011' is rejected.

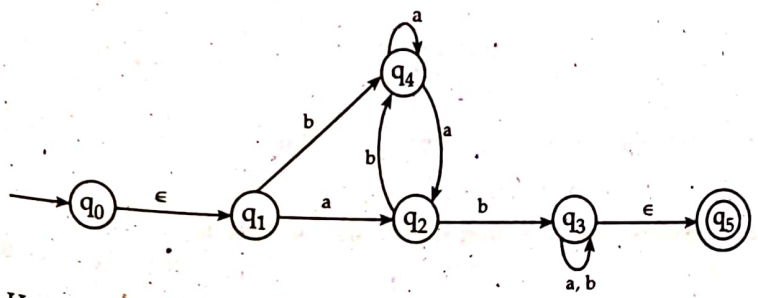
$(q_0, 01011) \vdash M(q_1, 1011)$
$\vdash M(q_2, 011)$
$\vdash M(q_3, 11)$
$\vdash M(q_2, 1)$
$\vdash M(q_2, \in)$ Rejected

Since, q_2 is not final state, so string '01011' is rejected.

28. Find the regular expression represented by NFA M = (k, Σ, Δ, S, F), where K = {q_0, q_1, q_2, q_3, q_4, q_5}, S = q_0, F = {q_5} and Δ is given as follows: **[2071 Shrawan, back]**

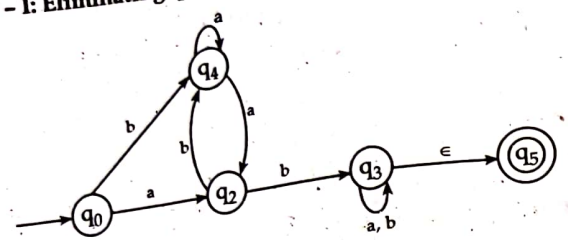
δ/Σ	a	b	∈
→ q_0	–	–	q_1
q_1	q_2	q_4	–
q_2	–	q_3, q_4	–
q_3	q_3	q_3	q_5
q_4	q_2, q_4	–	–
*q_5	–	–	–

Drawing equivalent states diagram of the given Δ,

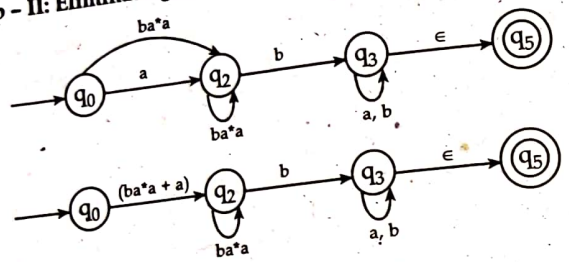


Here, q_0 is initial state and q_5 is final state. Eliminating intermediate states like q_1, q_2, q_3 and q_4 with their equivalent regular expression give regular expression on which we reach from initial state to final state and that will be our required regular expression.

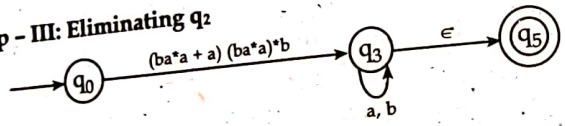
Step – I: Eliminating q_1



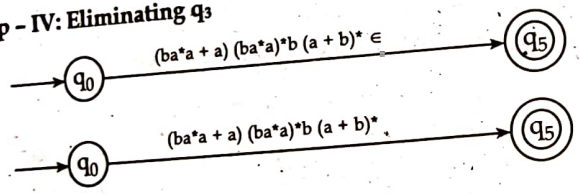
Step – II: Eliminating q_4



Step – III: Eliminating q_2



Step – IV: Eliminating q_3



Therefore, the required expression (R) is

$$R = (ba^*a + a)\,(ba^*a)^*b\,(a + b)^*$$

29. Explain about decision algorithms for regular language. **[2071 Shrawan, Back]**

The decision algorithms / properties of regular language are membership problem, emptiness problem and equivalence problems. And they are discussed below:

Algorithm 1: Given a regular language L over T and $w \in T^*$, there exists an algorithm for determining whether or not w is in L.

Let L be accepted by a DFA M(say). Then, for input w one can see whether w is accepted by M or not. The complexity of this algorithm is 0(n) where $|w| = n$.

Hence, membership problem for regular sets can be solved in linear time.

Algorithm – 2: There exists an algorithm for determining whether a regular language L is empty, finite or infinite. (Emptiness algorithm).

Let M be a DFA accepting L. In the state diagram representation of M with inaccessible states from the initial state removed, one has to check whether there is a simple directed path from initial state of M to a final state. If so, L is not empty. Consider a DFA, M' accepting L, where inaccessible state from the initial states are removed and also states from which a final state can't be reached are removed.

If in the graph of the state diagram of DFA, there are no cycles, then L is finite. Otherwise L is infinite.

One can see that the automaton accepts sentence of length less than n (where n is the number of states of the DFA) if and only if L(M) is non-empty. One can prove this statement using pumping lemma. That is $|w| < n$ for if w where the shortest and $|w| > n$ then $w = xyz$ and xz is shorter than w that belongs to L.

Also, L is infinite if and only if the automaton M accepts at least one word of length l where $n \leq l < 2n$. One can prove one can prove this by using pumping lemma. If $w \in L(M)$, $|w| \geq n$ and $|w| < 2n$, directing from pumping lemma, L is infinite.

Conversely, if L is infinite, we show that there should be a word in L whose length is l where $n \leq l < 2n$. If there is no word whose length is l where $n \leq l$ 2n let w be the word whose length is l where $n \leq l <$ 2n. Let w be the word whose length at least 2n, but as short as any word in L(M) whose length is greater than or equal to 2n. Then by, pumping lemma, $w = w_1 w_2 w_3$ where $1 \leq |w_2| \leq n$ and $w_1 w_2 \in$ L(M). Hence, either w was not the shortest word of length 2n or more or $|w_1 w_3|$ is between n and 2n – 1, which is a contradiction.

Algorithm 3: For any two regular languages L_1 and L_2, there exists an algorithm to determine whether or not $L_1 = L_2$ (Equivalence algorithm)

Consider $L = (L_1 \cap \overline{L_2}) \cup (\overline{L_1} \cap L_2)$. Clearly, L is regular by closure properties of regular languages. Hence, there exists a DFA M which accepts L. Now, by the previous algorithm, one can determine whether L is empty or not L is empty if and only if $L_1 = L_2$. Hence, the theorem.

30. Prove that language which contains set of strings of balanced parenthesis is not regular. **[2071 Shrawan, Back]**

Let us suppose language is regular, and n be a constant according to pumping lemma. We can divide each string in three parts xyz as $|xy| \leq n$ and $y \neq \epsilon$. So, xy^iz must be in L for $i \geq 0$.

Observation it is clear that language will contains the strings like (( ) ( )), ( ) ( ) ( ), (((( )))), ...

Let us select a string w = (((.. ( ) ...))), where there are n left parenthesis by n right parenthesis. this string is in L, and is of length at least n. So, we can divide w in three parts $w \simeq xyz$ as $|xy| \leq x$ and $y \neq \epsilon$.

Conclusion: Not let us choose i = 0. The resulting xz is not in L reason is very clear. Since $w \simeq xyz\,|xy| < n$.

It means y will be made of only left parenthesis. So xz is not in L since it has fewer left parenthesis than right. This contradicts the pumping lemma, so our original assumption, that L was regular, must have been incorrect.

31. Design a DFA that accepts the language L = {x ∈ {0, 1}* : x has an even number of 0's and an even number of 1's {verify your design for at least two strings that are accepted by this DFA and 2 strings that are rejected. **]2070 Chaitra]**

Let, the required DFA be

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

Let us analyse the language L. This language generates strings like {00, 11, 001100, 1100, ...}

Here,

$\theta = \{q_0, q_1, q_2, q_3\}$

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

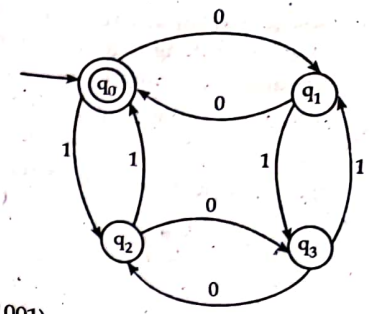
$q_0 = \{q_0\}$

$F = \{q_0\}$

and transition (δ) is defined as follows

Transition table

States/Σ	0	1
→* q_0	q_1	q_2
q_1	q_0	q_3
q_3	q_2	q_1
q_2	q_3	q_0



Accepted strings

Test for input string '001001'

$(q_0, 001001) \vdash M(q_1, 01001)$
$\vdash M(q_0, 1001)$
$\vdash M(q_2, 001)$
$\vdash M(q_3, 01)$
$\vdash M(q_2, 1)$
$\vdash M(q_0, \epsilon)$ Accepted

Since, q_0 is final state, string '00111' is accepted

$(q_0, 100111) \vdash M(q_2, 00111)$
$\vdash M(q_3, 0111)$
$\vdash M(q_2, 111)$
$\vdash M(q_0, 11)$
$\vdash M(q_2, 1)$
$\vdash M(q_0, \epsilon)$ Accepted

Since, q_0 is final state, string '100111' is accepted

Rejected strings

$(q_0, 100) \vdash M(q_2, 00)$
$\vdash M(q_3, 0)$
$\vdash M(q_2, \epsilon)$ Rejected

Since, q_2 is not final state, string '100' is not accepted i.e. rejected.

Test for input string '01001'

$(q_0, 01001) \vdash M(q_1, 1001)$
$\vdash M(q_3, 001)$
$\vdash M(q_2, 01)$
$\vdash M(q_3, 1)$
$\vdash M(q_1, \epsilon)$ Rejected

Since, q_1 is not final state, string '01001' is not accepted i.e. rejected.

32. Show that for any regular expression R, there is a NFA that accepts the same language represented by R. Construct a e-NFA for regular expression bb (a ∪ b)* ab. [2070 Chaitra]

R is obtained from a, (a ∈ Σ, ∈, ϕ by finite number of applications of 'U', ' . ' and '*'. (. is usually left out).

For ∈, ϕ and a we can construct NFA with ∈-moves as shown in fig (i)

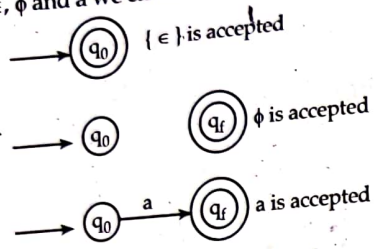


Fig. (i) NFA for ∈, ϕ for and a

Let R represents the regular set R_1 and R_1 is accepted by NFA M_1 with ∈-transitions fig (ii).

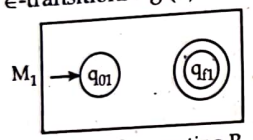


Fig. (ii) M_1 accepting R_1

Without less of generality, we can assume that each such NFA with ∈-moves has only one final states.

R_2 is similarly accepted by a NFA M_2 with ∈-transitions fig (iii)

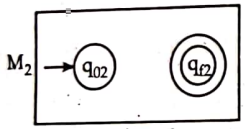


Fig. (iii) M_2 accepting R_2

Now, we can easily see that $R_1 \cup R_2$ is accepted by NFA given in fig. (iv).

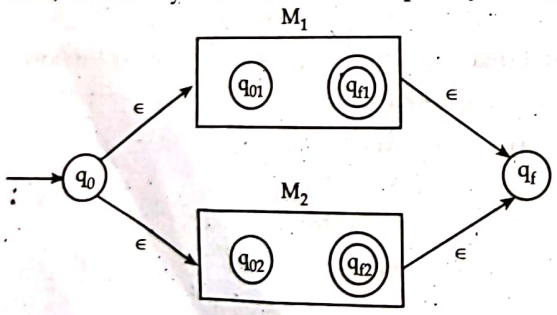


Fig. (iv)

Fig (iv) NFA for $R_1 \cup R_2$

For this NFA, q_0 is the 'start' state and qf is the 'end' state.

R_1R_2 is accepted by the NFA with ∈-moves given in fig (v)

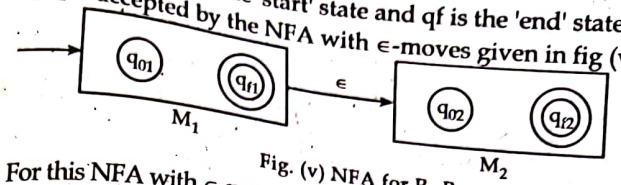


Fig. (v) NFA for $R_1 R_2$

For this NFA with ∈-moves, q_{01} is the initial state and q_{f2} is the final state.

$R_1^* = R_1^0 \cup R_1^1 = R_1^2 \cup \ldots R_1^k \cup \ldots$

$R_1^0 \{\in\}$ and $R_1^1 = R_1$

R_1^* is accepted by the NFA with e-moves given in fig (vi)

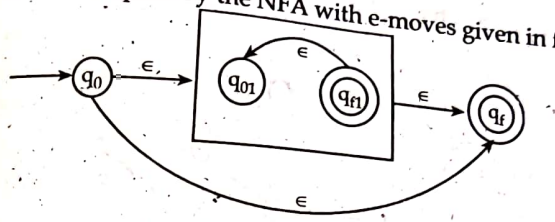


Fig. (vi) NFA for R_1^*

For this NFA with ∈-moves, q_0 is the initial state and qf is the final state it can be seen that R_1^* contains strings of the form $x_1x_2 \ldots x_k$ each $x_1 \in R_1$. To accept this string, the control goes from q_0 to q_{01} and then after reading x_1 and reaching q_{f1}, it goes to q_{01}, by an ∈-transition. From q_{01}, it again reads x_2 and goes to q_{f1}. This can be repeated a number of (k) times and finally the control goes to q_f from q_{f1} by an ∈-transition $R_1^0 = \{\in\}$ is accepted by going to q_f from q_0 by an ∈-transitions.

Thus, we have seen that for a given regular expression one can construct an equivalent NFA with ∈-transitions.

We know that we can construct an equivalent NFA without ∈-transitions from this, and can construct a DFA as show in figure below.

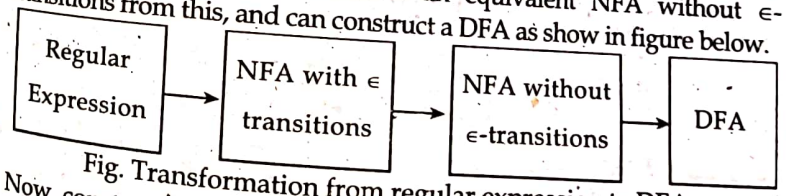
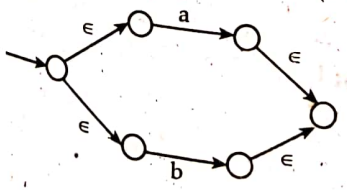


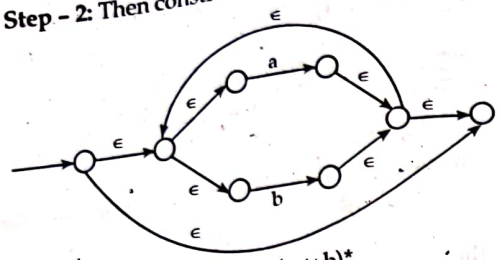
Fig. Transformation from regular expression to DFA.

Now, construction of regular expression bb(a ∪ b)* ab.

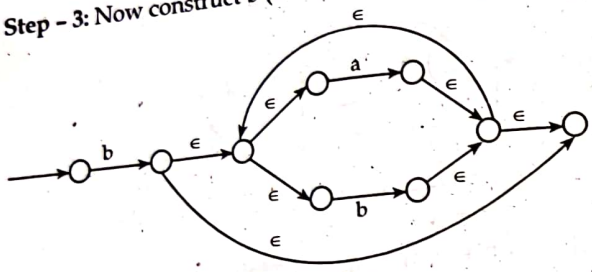
Step - 1: We first construct (a ∪ b)



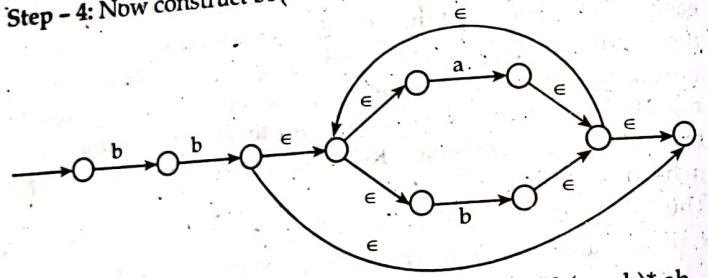
Step – 2: Then construct $(a \cup b)^*$.



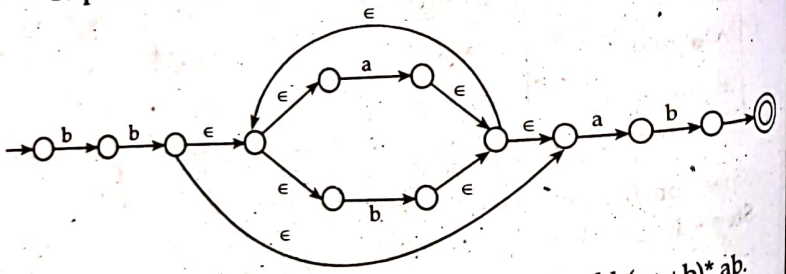
Step – 3: Now construct $b\,(a \cup b)^*$



Step – 4: Now construct $bb(a \cup b)^*$



Step – 5: In the similar manner, we can construct $bb(a \cup b)^* ab$.



This is the required e-NFA for regular expression $bb(a \cup b)^* ab$.

33. **Use pumping lemma to prove that $L = \{a^n b^{2n} : n \geq 1\}$ is not regular.**

Please see 2073 Shrawan.

◆◆◆

Chapter - 3

Context Free Grammar

Introduction

All of us know that a grammar is nothing but a set of rules to define any sentences in any languages. In this chapter, we introduce the context free grammar, which generate context free languages. Context free languages have great practical significance in defining programming languages and in simplifying the translation for programming languages.

Initially linguists were trying to define precisely valid sentences and to give structural description for these sentences. They tried to define rules for natural languages. Noam Chomsky gave a mathematical idea model for the grammars in 1956. Although it was useless to describe natural languages but it become very useful for computer languages. The original motivation for grammars was the description of natural languages. We can write rules for the grammar of natural languages as follows:

<sentence> – <noun phrase> <verb phrase>

<noun article> – < article> <noun>

According to above set of rules "we are fine" is valid sentence.

Formal definition of context free grammars

A context free grammar G is a quadruple defined as follows:

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

where

v is an alphabet (set of terminals and non-terminal

Σ (the set of terminals) is a subset of v.

R (the set of rules) is a finite subset of $(v - \Sigma) \times v^*$ and

S (the start symbol) is an element

Note: Terminals are generally denoted by small letters. non-terminals or variables are generally denoted by capital letters.

Exam Solution

1. Convert the following CFG into CNF with explanation of each steps G = (V, Σ R, S), where

V = {S, X, Y, a, b, c},

Σ = {a, b, c}

R = {S → a X b X

X → a Y / b Y / X Y / ∈

Y → a X / c

} **[2075 Ashwin, Back]**

The given set of production / rules

R = {S → aX bX

X → aY / bY / XY / ∈

Y → aX / a / c

Here, we have a null production X → ∈, Removing null production X ∈, we replace symbol X be ∈ and add the new production to rules / grammar i.e.

S → aX bX / abX / aXb / ab

X → aY / bY / XY / Y

Y → aX / a / c

Now, we have unit production X → Y, we add production of Y to X to remove it.

S → aX bX / abX / a Xb / ab

X → aY / bY / XY / aX / a / c

Y → aX / a / c

Now, we don't have neither null production nor unit production. Also, we don't have useless symbols.

Therefore, new set of productions

R' = {

S → aX bX / abX / aXb / ab

X → aY / bY / XY / aX / a /C

Y → aX / a / C

Now, this set is ready to be converted into Chomsky normal form

Let, $C_a \rightarrow a$

$C_b \rightarrow b$

Then S → $C_aX\ C_bX$ / $C_a\ C_bX$ / $C_aX\ C_b$ / C_aC_b

X → C_aY / C_bY / XY / C_aX / a / C

Y → C_aX / a / C

Since, CNF is of form A → BC, A → a (i.e. expressions on right hand side of production must be two variable (non-terminals) or a single terminal, we will introduce new-variables (non-terminals) to get CNF.

For S → $C_aX\ C_bX$, replace D_1 by $CAX\ C_b$ and so on

S → D_1X

D_1 → $D_2\ C_b$

D_2 → C_aX

For S → $C_a\ XC_b$, we have

S → $D_2\ C_b$

D_2 → C_aX

For S → $C_a\ C_bX$, we replace D_3 by $C_a\ C_b$

S → $D_3\ X$

D_3 → $C_a\ C_b$

Now, all of the productions are of the form CNF

R' = {

S → D_1X / D_3X / $D_2\ C_b$ / $C_a\ C_b$

X → C_aY / C_bY / XY / C_aX / a / C

Y → C_aX / a / C

D_1 → $D_2\ C_b$

D_2 → C_aX

D_3 → $C_a\ C_b$

Ca → a

Cb → b

}

This is the required CNF form

2. Convert the following CGF into CNF with explanation of each steps G = {V, Σ, R, S) where

V = {S, A, B, a, b}

Σ = {a, b}

R = {

S → ASB / ∈

A → aAS / a

B → AB / b / ∈

} **[2074 Ashwin, Back]**

Given sets of productions

R = {

S → ASB / ∈

A → aAS / a

B → AB / b / ∈

}

Step-I: Elimination of useless.

Here, no productions are useless. So, no need to remove useless symbols.

Step-II: Elimination of null (∈) productions.

Here, two null productions are present

S → ∈.

B → ∈

First, remove S → ∈ on doing so, we replace S by ∈ and all the new productions to the grammar (i.e.)

S → ASB / AB

A → aAS / aA / a

B → AB / b / ∈

Now, we remove B → ∈. On doing so, we replace B by ∈ and add all the new productions to the grammar (i.e.)

S → ASB / AB / AS / A

A → aAS / aA / a

B → AB / b / A

Now, all null productions are removed

Step-III: Elimination of unit productions.

Here, unit productions are

S → A

B → A

First, remove S → A. On doing so, we add all production of A to the production of S

S → ASB / AB / AS / aAS/ aA / a

A → aAS / aA / a

B → AB / A / b

Secondly, remove B → A, on doing so, we add all productions of A to the production of B.

S → ASB / AB / aAS / AS / aA / a

A → aAS / aA / a

B → AB / aAS / aA / a / b

Now, we have new set of production R'

R' = {

S → ASB / AB / aAS / AS / aA / a

A → aAS / aA / a

B → AB / aAS / aA / a / b

}

For CNF, we make all productions either as combination of two non-terminals or a single terminal.

Let Ca → a then

S → ASB / AB / C_aAS / AS / C_aA / a

A → C_aAS / C_aA / a

B → AB / C_aAS / C_aA / a / b

For S → ASB, we have

S → D_1B

D_1 → AS

For S → C_aAS, we have

S → D_2S

D_2 → C_aA

For A → C_aAS, we have

A → D_2S

D_2 → C_aA

For B → C_aAS, we have

B → D_2S

D_2 → C_aA

Finally new set of production P' is

P' = {

S → D_1B / AB / D_2S / AS / C_aA / a

A → D_2S / C_aA / a

B → AB / D_2S / C_aA / a / b
C_a → a
D_1 → AS
D_2 → C_aA
}

This is the required CNF

3. Define Chomsky normal form (CNF). Convert the following CFG into CNF G = (V, Σ R, S)

where

V = {S, A, B, a, b}
Σ = {a, b}
R = {
S → A
S → B
A → a Ba
A → e
B → b Ab
B → e
} where e is empty symbol. **[2073 Chaitra]**

For definition of Chomsky normal form (CNF), please see theory part.

Here, the given set of productions

R = {
S → A / B
A → aBa / e
B → b Ab / e
}

Step-I: Elimination of useless symbols none of the productions / symbols are useless.

Step-II: Elimination of null (∈) production null productions are

A → e
B → e

First, we remove A → e on doing so, we replace A by e and add new productions to the grammar

S → A / B / e
A → a Ba
B → b Ab / bb / e

Secondly, we remove B → e, on doing so, we replace B by e and add new productions to the grammar

S → A / B / e
A → a Ba / aa
B → b Ab / bb

Note: no need to write two 'e' in production of A.

Now, we have new null production S → e, removing S → e, we replace S by e and add new productions to the grammar

S → A / B
A → a Ba / aa
B → b Ab / bb

Step-III: Elimination of unit productions.

Here, unit productions are

S → A
S → B

To remove unit production S → A, we add every production of A to the reproduction of S. i.e.

S → aBa / aa / B
A → aBa / aa
B → bAb / bb

To remove unit production S → B, we add every production of B to the production of S.

S → aBa / aa / bAb / bb
A → aBa / aa
B → bAb / bb

In order to convert CGE into CNF, we make each production either as combination of two non-terminals or a single terminal.

We have new set of production R' as

R' = {
S → a BA / aa / bAb / bb
A → a Ba / aa
B → b Ab / bb
}

Let,

$C_a \rightarrow a$

$C_b \rightarrow b$

$S \rightarrow C_a BC_a / C_a C_a / C_b A C_b / C_b C_b$

$A \rightarrow C_a BC_a / C_a C_a$

$B \rightarrow C_b A C_b / C_b C_b$

For $S \rightarrow C_a B C_a$, we have

$S \rightarrow D_1 C_a$

$D_1 \rightarrow C_a B$

For $S \rightarrow C_b A C_b$, we have

$S \rightarrow D_2 C_b$

$D_2 \rightarrow C_b A$

For $A \rightarrow C_a B C_a$, we have

$A \rightarrow D_1 C_a$

$D_1 \rightarrow C_a B$

For $B \rightarrow C_b A C_b$, we have

$B \rightarrow D_2 C_b$

$D_2 \rightarrow C_b A$

Finally, we have new set of production R' as

R' = {

$S \rightarrow D_1 C_a / C_a C_a / D_2 C_b / C_b C_b$

$A \rightarrow D_1 C_a / C_a C_a$

$B \rightarrow D_2 C_b / C_b C_b$

$C_a \rightarrow a$

$C_b \rightarrow b$

$D_1 \rightarrow C_a B$

$D_2 \rightarrow C_b A$

}

This is the required CNF form.

4. What is Chomsky normal form (CNF)?

[S, L, M, N, a, b, C], Σ = {a, b, c}

R = {

S → Ma N / bL / bM

L → ab / CN / M1 / ∈

M → a / cM

N → abN

}

For definition, please see theory part **[2073 Shrawan, Back]**

Given, we have

R = {

S → MaN / bL / bM

L → ab / cN / M1 / ∈

M → a / cM

N → abN

}

Step – I: Elimination of useless symbol.

Here, N is useless i.e. non-generating symbol as it doesn't generate any kind of terminal symbol or symbols. So, removing all productions containing 'N', we get

$S \rightarrow bL / bM$

$L \rightarrow ab / M1 / \in$

$M \rightarrow a / CM$

Step – II: Elimination of null production.

Here, null production is

$L \rightarrow \in$

So, we replace L by ∈ and add these new productions to the grammars.

$S \rightarrow bL / bM / b$

$L \rightarrow ab / M1$

$M \rightarrow a / CM$

Step – III: Elimination of unit production.

Here, no unit productions are present we have new set of productions

R' = {

S → bL / bM / b

L → ab / M1

M → a / CM

}

In order to convert CFG to CNF form, we make each production either as combination of two non-terminal symbol or a single terminal symbol.

Let $C_a \rightarrow a$
$C_b \rightarrow b$
$C_1 \rightarrow 1$
$C_c \rightarrow C$

Then, R' = {
$S \rightarrow C_bL / C_bM / b$
$L \rightarrow C_aC_b / M C_1$
$M \rightarrow a / C_cM$
$C_a \rightarrow a$
$C_b \rightarrow b$
$Cc \rightarrow C$
$C_1 \rightarrow 1$
}

This is the required CNF form.

5. Convert the following CFG into CNF

G = {V, T, P, S}
where V = {S, A, B, C, a, b, c}
T = {a, b, c}
P = {
S → ABA / abA / BC
A → aA / ∈
B → baB / c
C → aC
} **[2074 Chaitra]**

We have,
P = {
$S \rightarrow ABA / abA / BC$
$A \rightarrow aA / \in$
$B \rightarrow ba B / c$
$C \rightarrow aC$
}

Step – I: Elimination of useless symbol.

Here, symbol 'c' is useless as it doesn't generate any kind of terminal symbols. So, we need to remove all productions containing C.

$C \rightarrow AB A / ab A$
$A \rightarrow aA / \in$
$B \rightarrow ba B / c$

Step-II: Elimination of null production.

Here, $A \rightarrow \in$ is null production, so we replace A by ∈ and add the new productions to the grammar i.e.

$S \rightarrow ABA / BA / AB / B / abA / ab$
$A \rightarrow aA$
$B \rightarrow baB / c$

Step-III: Elimination of unit production we have $S \rightarrow B$ (unit production), so we add production of B to the production of S i.e.

$S \rightarrow ABA / BA / AB / baB / c / abA / ab$
$A \rightarrow aA$
$B \rightarrow baB / c$

Now, we have new set of productions

P' = {
$S \rightarrow ABA / BA / AB / baB / c / ab A / ab$
$A \rightarrow aA$
$B = baB / c$
}

Let, $C_a \rightarrow a$
$C_b \rightarrow b$

Then, P' = {
$S \rightarrow ABA / BA / AB / C_b C_aB / C / C_a C_bA / C_a C_b$
$A \rightarrow C_aA$
$B \rightarrow C_bC_aB / c$

For CNF, we make each productions of the grammar either as combination of two non-terminals or a single terminal i.e.

For $S \rightarrow ABA$, we have
$S \rightarrow D_1 A$
$D_1 \rightarrow AB$
For $S \rightarrow C_b C_aB$, we have
$S \rightarrow D_2 B$
$D_2 \rightarrow C_b C_a$

For S $\rightarrow C_a C_b$ A, we have

S $\rightarrow D_3$ A

$D_3 \rightarrow C_a C_b$

For B $\rightarrow C_b C_a$ B, we have

B $\rightarrow D_2$ B

$D_2 \rightarrow C_b C_a$

Now, set of production become

P' = {

S $\rightarrow D_1$ A / BA / AB / D_2B / c / D_3A / C_aC_b

B $\rightarrow C_a$A

B $\rightarrow D_2$B / c

$C_a \rightarrow$ a

$C_b \rightarrow$ b

$D_1 \rightarrow$ AB

$D_2 \rightarrow C_b C_a$

$D_3 \rightarrow C_a C_b$

}

This is the required CNF form.

6. Define context free grammar. Convert the given context free grammar (CFG) into equivalent CNF.

S $\rightarrow$ AB

A $\rightarrow$ a AA / e

B $\rightarrow$ bBB / e **[2072 Chaitra]**

Formal definition of context free grammar (CFG), please see theory part.

Now, we have

S $\rightarrow$ AB

A $\rightarrow$ aAA / e

B $\rightarrow$ bBB / e

Step – I: Elimination of useless symbols

Here, no symbols are useless as all productions can be reached from start symbols 's'.

Step – II: Elimination of null 'e' production.

Here, null productions are

A $\rightarrow$ e

First removing A $\rightarrow$ e, on doing so, we replace symbol A by 'e' and adding all the productions to the grammar, we get

S $\rightarrow$ AB / B

A $\rightarrow$ a AA / aA / a

B $\rightarrow$ bBB / e

Secondly removing B $\rightarrow$ e, we replace symbol 'B' by 'e' and adding all the productions to the grammar

S $\rightarrow$ AB / B / A

A $\rightarrow$ aAA / aA / a

B $\rightarrow$ bBB / bB / b

Step – III: Elimination of unit production.

Here, unit productions are

S $\rightarrow$ A

S $\rightarrow$ B

For removing S $\rightarrow$ A unit production, we add all production of A to the production of S.

S $\rightarrow$ AB / B / aAA / aA / a

A $\rightarrow$ aAA / aA / a

B $\rightarrow$ bBB / bB / b

For removing S $\rightarrow$ B, unit production, we add all production of B to the production of S

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

A $\rightarrow$ aAA / aA / a

B $\rightarrow$ bBB / bB / a

Now, we have new set of productions or grammar

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

A $\rightarrow$ aAA / aA / a

B $\rightarrow$ aBB / aB / b

In order to convert CFG into CNF, we write all productions on right hand side either as combination of two non-terminal symbol or a single terminal

Let, $C_a \rightarrow$ a

$C_b \rightarrow$ b

Then, S $\rightarrow$ AA / Ca AA / C_aA / a / C_b BB / C_bB / b

A $\rightarrow C_a$ AA / C_aA / a

B $\rightarrow C_b$ BB / C_bB / b

For S $\rightarrow C_a$ AA, we have

S $\rightarrow D_1A$

$D_1 \rightarrow C_a A$

For S $\rightarrow C_b$ BB, we have

S $\rightarrow D_2B$

$D_2 \rightarrow C_b B$

For A $\rightarrow C_a$ AA, we have

A $\rightarrow D_1A$

$D_1 \rightarrow C_a A$

For B $\rightarrow C_b$ BB, we have

B $\rightarrow D_2 B$

$D_2 \rightarrow C_bB$

New set of productions become

S $\rightarrow$ AB / D_1A / a / D_2B / C_bB / b

A $\rightarrow D_1A$ / $C_a A$ / a

B $\rightarrow D_2B$ / $C_b B$ / b

$C_a \rightarrow a$

$C_b \rightarrow b$

$D_1 \rightarrow C_a A$

$D_2 \rightarrow C_b B$

This is the required CNF from

7. Convert the following CFG to CNF

G = (V, Σ, R, S) where

V = {S, A, B, a, b}

Σ = {a, b}

R = {

S → aAb / Ba / A

A → SS / e

B → e

} **[2071 Chaitra]**

The given set of production / rules

R = {

S $\rightarrow$ aAb / Ba / A

A $\rightarrow$ SS / e

B $\rightarrow$ e

}

Step – I: Elimination of useless symbols.

Here, no symbols are useless as all symbols can be reached from start symbol 'S'

Step – II: Elimination of null (e) production

Here, null productions are

A $\rightarrow$ e

B $\rightarrow$ e

First, removing A $\rightarrow$ e, we replace A by e and add all the new productions to grammar i.e.

S $\rightarrow$ aAb / ab / Ba / e

A $\rightarrow$ SS

B $\rightarrow$ e

Secondly, removing B $\rightarrow$ e, we replace B by e and add the new productions to grammar i.e.

S $\rightarrow$ aAb / ab / Ba / a / e

A $\rightarrow$ SS

Now, we have new null production S $\rightarrow$ e, so removing S $\rightarrow$ e, we get

S $\rightarrow$ aAb / ab / Ba / a

A $\rightarrow$ SS / S

Note: no need to write A $\rightarrow$ e as we have already removed A $\rightarrow$ e.

Step – III: Elimination of unit productions.

Here, unit production is

A $\rightarrow$ S

So, removing A $\rightarrow$ S unit production, we add all production of S to the productions of A

S $\rightarrow$ aAb / ab / Ba / a

A $\rightarrow$ SS / aAb / ab / Ba / a

Therefore, new set of productions

R' = {

S $\rightarrow$ aAb / ab / Ba / a

A $\rightarrow$ SS / aAb / ab / Ba / a

}

For CNF, we make all productions either as combination of two non-terminals or a single terminal

Let, $C_a \rightarrow a$

$C_b \rightarrow b$, then

S $\rightarrow C_a AC_b / C_a C_b / BC_a / a$

A $\rightarrow SS / C_a AC_b / C_aC_b / BC_a / a$

For S $\rightarrow C_a A C_b$, we have

S $\rightarrow D_1 C_b$

$D_1 \rightarrow C_a A$

For $A \rightarrow C_a A C_b$, we have

A $\rightarrow D_1 C_b$

$D_1 \rightarrow C_a A$

Finally, new set of productions R' is

R' = {

S $\rightarrow D_1 C_b / C_a C_b / BC_a / a$

A $\rightarrow SS / D_1 C_b / C_a C_b / BC_a / a$

$C_a \rightarrow a$

$C_b \rightarrow b$

$D_1 \rightarrow C_a A$

}

This is the required CNF form.

8. Convert the following CFG to Chomsky normal form (CNF)

G = {V, Σ, R, S) where V = {S, A, B, a, b}

Σ = {a, b}

R = {

S → aB bB

A → aA,

A → a

B → bB

} **[2072 Kartik, Back]**

Given, set of productional rules is

R = {

S $\rightarrow$ aA bB

A $\rightarrow$ aA / a

B $\rightarrow$ aB

}

Step – I: Elimination of useless symbols.

Here, symbol B is useless symbol as it doesn't generate any kind of terminal symbol. So, we need to remove all production containing B. i.e.

R = {A → aA / a}

Step – II: Elimination of null production.

No null productions

Step – III: Elimination of unit production.

No unit productions.

Now, we have new set of productions

R' = {A → aA / a}

For CNF, we make each production of the grammar either as combination of two non-terminal or a single terminal i.e.

Let, $C_a \rightarrow a$

Then,

R' = {S $\rightarrow C_a A / a$

$C_a \rightarrow a$

}

This is the required CNF form.

9. Define a context free grammar convert the following productions into Chomsky normal form.

S → ab AB

A → b AB / ∈

B → BA a / A / ∈ **[2073 Shrawan, Back]**

For definition of context free grammar, please see theory part.

Given, the set of productions

R = { S $\rightarrow$ ab AB

A $\rightarrow$ b AB / ∈

B $\rightarrow$ BA a / A / ∈

}

Step I: Elimination of useless symbols.

Here, no symbols are useless

Step II: Elimination of null (∈) productions

Here, null productions are

A $\rightarrow$ ∈

B $\rightarrow$ ∈

First, we remove $A \to \epsilon$, on doing so, we replace symbol A by ϵ and add all the new productions to the grammar.

S $\to$ abAB /abB
A $\to$ b AB / bB
B $\to$ BA a / Ba / ϵ

Secondly, we remove $B \to \epsilon$, on doing so, we replace symbol B by ϵ and add all the new productions to the grammar

S $\to$ abAB /abA / abB
A $\to$ bAB / bA / bB
B $\to$ BA a / Ba / Aa

Step III: Elimination of unit production

Here, no unit productions are present

Therefore, we have new set of productions

R' = { S $\to$ abAB /abA / abB
A $\to$ bAB / bA / bB
B $\to$ BAa / Ba / Aa
}

For CNF, we write all productions either as combination of two non-terminals or a single terminal.

Let, $C_a \to a$
$C_b \to b$

Then, S $\to C_a C_b$ AB / $C_a C_b$ A / $C_a C_b$ B
A $\to C_b$ AB / C_b A/ C_b B
B $\to$ BA C_a / B C_a / A C_a

For $S \to C_a C_b$ AB, we have
S $\to D_1$ B
$D_1 \to D_2$ A
$D_2 \to C_a C_b$

For $S \to C_a C_b$ A, we have
$D \to D_2$ A
$D_2 \to C_a C_b$

For $S \to C_a C_b$ B, we have
$S \to D_2$ B
$D_2 \to C_a C_b$

For $A \to C_b$ AB, we have
$A \to D_3$ B
$D_3 \to C_b$ A

For $A \to$ BA C_a, we have
$A \to D_4 C_a$
$D_4 \to$ BA

Finally new set of production

R' = {
S $\to D_1$ B/ D_2 A / D_2 B
A $\to D_3$ B/ C_b A / C_b B
B $\to D_4 C_a$/ B C_a / A C_a
$C_a \to a$
$C_b \to b$
$D_1 \to D_2$ A
$D_2 \to C_a C_b$
$D_3 \to C_b$ A
$D_4 \to$ BA
}

This is the required CNF form.

10. Convert the following CFG into CNF with explanation of each step G = {V, Σ, R, S) where

V = {S, X, Y, Z, a, b, c}
Σ = {a, b, c}
R = {
S → XYZ / XY / aZ
X → abX / ∈
Y → bY / cZ / ab
Z → aXZ
} **[2071 Shrawn, Back]**

We have given set of productions

R = {
S → XYZ / XY / aZ
X → abX / ∈
Y → bY / cZ / ab
Z → aXZ
}

Step I: Elimination of useless symbols.

Here, Z is useless symbol as it doesn't generate any kind of terminal symbol. So, we need to remove all these productions containing Z i.e.

$S \rightarrow XY$

$X \rightarrow ab\,X \mid \epsilon$

$Y \rightarrow aY \mid ab$

Step II: Elimination of null (ϵ) productions.

Here, null production is

$X \rightarrow \epsilon$

On removing $X \rightarrow \epsilon$, we replace symbol X by ϵ and add all the new productions to the grammar.

$S \rightarrow XY \mid Y$

$X \rightarrow abX \mid ab$

$Y \rightarrow bY \mid ab$

Step III: Elimination of unit production

Here, unit production is

$S \rightarrow Y$

On removing $S \rightarrow Y$, we add all productions of Y to the production of S i.e.

$S \rightarrow XY \mid bY \mid ab$

$X \rightarrow abX \mid ab$

$Y \rightarrow bY \mid ab$

Therefore, we have new set of productions

$R' = \{$

$S \rightarrow XY \mid bY \mid ab$

$X \rightarrow abX \mid ab$

$Y \rightarrow bY \mid ab$

$\}$

For CNF, we write all productions either as combination of two terminals or a single terminal.

Let, $C_a \rightarrow a$

$C_b \rightarrow b$

Then, $S \rightarrow XY \mid C_b Y \mid C_a C_b$

$X \rightarrow C_a C_b X \mid C_a C_b$

$Y \rightarrow C_b Y \mid C_a C_b$

For $X \rightarrow C_a C_b X$, we have

$X \rightarrow D_1 X$

$D_1 \rightarrow C_a C_b$

Finally, we have new set of productions

$R' = \{$

$S \rightarrow XY \mid C_b Y \mid C_a C_b$

$X \rightarrow D_1 X \mid C_a C_b$

$Y \rightarrow C_b Y \mid C_a C_b$

$C_a \rightarrow a$

$C_b \rightarrow b$

$D_1 \rightarrow C_a C_b$

$\}$

This is the required CNF form.

11. What is additional feature PDA has when compared with finite automata? Explain. Design a pushdown automata (PDA) which accepts all the strings of language

$L = \{a^n b^m c^{2n} : n, m > 0\}$ **[2075, Ashwin, Back]**

The additional feature PDA has got when compared with finite automata is a storage device for inputs called stack. A stack is a "first in last out" or "last in first out" (LIFO) that is, symbols can be entered or removed only at the top of the list. When a new symbol is entered at the top, the symbol previously at the top becomes second and so on. Due to the stack, PDA can store the input it has read and process i.e. POP out and keep the input symbols.

Let, the required PDA be

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

where,

$\theta = \{q_0, q_1, q_2, q_3, q_4, q_5\}$

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

$\Gamma = \{a, b, c, Z_0\}$

$q_0 = \{q_0\}$

$Z_0 = \{Z_0\}$

$F = \{q_5\}$

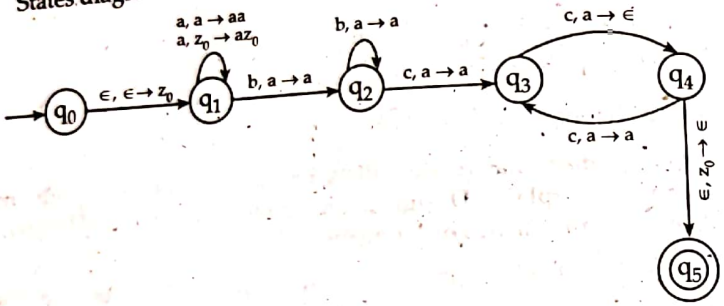
and δ is defined as follows:

δ = {

1: $((q_0, \in, \in), \quad (q_1, Z_0))$
2. $((q_1, a, Z_0), \quad (q_1, aZ_0))$
3. $((q_1, a, a), \quad (q_1, aa))$
4. $((q_1, b, a), \quad (q_2, a))$
5. $((q_2, b, a), \quad (q_2, a))$
6. $((q_2, c, a), \quad (q_3, a))$
7. $((q_3, c, a), \quad (q_4, \in))$
8. $((q_4, c, a), \quad (q_3, a))$
9. $((q_4, \in, Z_0), \quad (q_5, \in))$

}

States diagram



Test for input string 'abbcc'

S.No.	State	Unread input	Stack	Transition used
1.	q_0	abbcc	∈	–
2.	q_1	abbcc	Z_0	1
3.	q_1	bbcc	a Z_0	2
4.	q_2	bcc	a Z_0	4
5.	q_2	cc	a Z_0	4
6.	q_3	c	a Z_0	6
7.	q_4	∈	Z_0	7
8.	q_5	∈	∈	9

Accepted

Since, stack is empty and the final state is accepting state, so string 'abbcc' is accepted.

12. Mention role of parse tree in context free grammar Design a PDA that accepts $L = \{a^n b^{2n+1}, n > 0\}$ and check it for string aabbbbb.

[2074 Ashwin, Back]

The role of parse tree in context free grammar are as follows:

(a) From parse tree, it helps us to generate intermediate strings at any part of derivation.

(b) It detects and reports any syntax errors in deviation part.

(c) We can analyse the useless symbols in derivation of a particular string.

(d) From parse tree, we can determine whether the derivation of string is either leftmost or rightmost.

(e) It helps to remove ambiguity in context free grammar.

Let, the required PDA be

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

where

$\theta = \{q_0, q_1, q_2, q_3, q_4\}$

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

$\Gamma = \{a, b, Z_0\}$

$q_0 = \{q_0\}$

$F = \{q_4\}$

$Z_0 = \{Z_0\}$

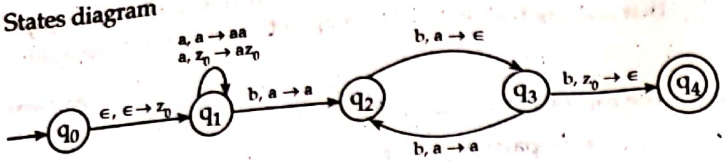
and δ is defined as follows:

δ = {

1. $((q_0, \in, \in), \quad (q_1, Z_0))$
2. $((q_1, a, Z_0), \quad (q_1, aZ_0))$
3. $((q_1, a, a), \quad (q_1, aa))$
4. $((q_1, b, a), \quad (q_2, a))$
5. $((q_2, b, a), \quad (q_3, \in))$
6. $((q_3, b, a), \quad (q_2, a))$
7. $((q_3, b, Z_0), \quad (q_4, \in))$

}

States diagram



Test for input string 'aabbbbb'

S.No.	State	Unread input	Stack	Transition used
1.	q_0	aabbbbb	∈	–
2.	q_1	aabbbbb	Z_0	1
3.	q_1	abbbbb	a Z_0	2
4.	q_1	bbbbb	aa Z_0	3
5.	q_2	bbbb	aa Z_0	4
6.	q_3	bbb	a Z_0	5
7.	q_2	bb	a Z_0	6
8.	q_3	b	Z_0	5
9.	q_4	∈	∈	7

Accepted

Since, stack is empty and the final state is accepting state i.e. q_4, string 'aabbbbb' is accpeted.

13. Design a pushdown automata (PDA) for $L = \{a^n b^{2n} : n \geq 1\}$. Hence test for "aaabbb" and "aabbbb". [2074 Chaitra]

Let, the required PDA be

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

where

$\theta = \{q_0, q_1, q_2, q_3, q_4\}$

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

$\Gamma = \{a, b, Z_0\}$

$q_0 = \{q_0\}$

$Z_0 = \{Z_0\}$

$F = \{q_4\}$

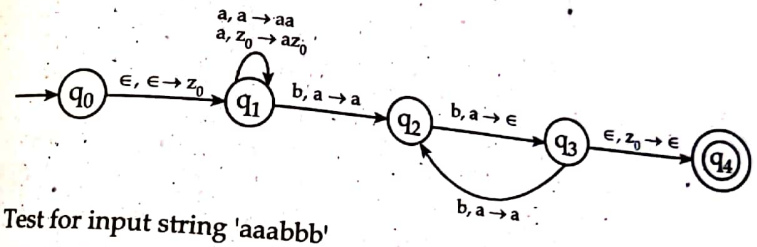
and transition (δ) is defined as follows:

$\delta = \{$

1. $((q_0, \in, \in), (q_1, Z_0))$
2. $((q_1, a, Z_0), (q_1, aZ_0))$
3. $((q_1, a, a), (q_1, aa))$
4. $((q_1, b, a), (q_2, a))$
5. $((q_2, b, a), (q_3, \in))$
6. $((q_3, b, a), (q_2, a))$
7. $((q_3, \in, Z_0), (q_4, \in))$

$\}$

States diagram



Test for input string 'aaabbb'

S.No.	State	Unread input	Stack	Transition used
1.	q_0	aaabbb	∈	–
2.	q_1	aaabbb	Z_0	1
3.	q_1	aabbb	a Z_0	2
4.	q_1	abbb	aa Z_0	3
5.	q_1	bbb	aaa Z_0	3
6.	q_2	bb	aaa Z_0	4
7.	q_3	b	aa Z_0	5
8.	q_2	∈	aa Z_0	6

Accepted

Since, there is no transition defined for state q_2 on input '$\in$' i.e. $\delta(q_2, \in, a) \rightarrow \phi$, so string 'aaabbb' is rejected.

Test for input string 'aabbbb'

S.No.	State	Unread input	Stack	Transition used
1.	q_0	aabbbb	$\in$	–
2.	q_1	aabbbb	Z_0	1
3.	q_1	abbbb	a Z_0	2
4.	q_1	bbbb	aa Z_0	3
5.	q_2	bbb	aa Z_0	4
6.	q_3	bb	a Z_0	5
7.	q_2	b	a Z_0	6
8.	q_3	$\in$	Z_0	5
9.	q_4	$\in$	$\in$	7

Accepted

Since, stack is empty and final state is also accepting state. So, string 'aabbbb' is accepted.

14. Design a PDA that accepts those strings "having total number of 'a' equal to the sum of number of 'b' and 'c' with sequence of a, b, c (i.e. $a^i\ b^j\ c^k : i = j + k$). Hence, test your design for the string "aaaabbcc". [2073 Chaitra]

Let, the required PDA be

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

where

$\theta = \{q_0, q_1, q_2, q_3, q_4\}$

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

$\Gamma = \{a, b, Z_0\}$

$q_0 = \{q_0\}$

$Z_0 = \{Z_0\}$

$F = \{q_4\}$

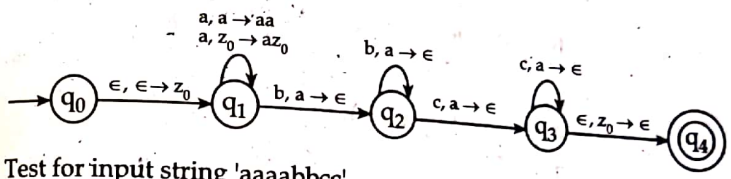
and transition (δ) is defined as follows:

$\delta = \{$

1. $((q_0, \in, \in), \quad (q_1, Z_0))$
2. $((q_1, a, Z_0), \quad (q_1, aZ_0))$
3. $((q_1, a, a), \quad (q_1, aa))$
4. $((q_1, b, a), \quad (q_2, \in))$
5. $((q_2, b, a), \quad (q_2, \in))$
6. $((q_2, c, a), \quad (q_3, \in))$
7. $((q_3, c, a), \quad (q_3, \in))$
8. $((q_3, \in, Z_0), \quad (q_4, \in))$

$\}$

States diagram



Test for input string 'aaaabbcc'

S.No.	State	Unread input	Stack	Transition used
1.	q_0	aaaabbcc	$\in$	–
2.	q_1	aaaabbcc	Z_0	1
3.	q_1	aaabbcc	a Z_0	2
4.	q_1	aabbcc	aa Z_0	3
5.	q_1	abbcc	aaa Z_0	3
6.	q_1	bbcc	aaaa Z_0	3
7.	q_2	bcc	aaa Z_0	4
8.	q_2	cc	aa Z_0	5
9.	q_3	c	a Z_0	6
10.	q_3	$\in$	Z_0	7
11.	q_4	$\in$	$\in$	8

Accepted

Since, stack is empty and final state is also accepting state so string 'aaaabbcc' is accepted

15. Construct a PDA which accepts the language $L = \{a^n b^{n+m} c^m : n, m \geq 1\}$. Verify your design by taking s string "abbcc" as example. **[2073 Shrawan, Back]**

Let, the required PDA be

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

where

$\theta = \{q_0, q_1, q_2, q_3, q_4, q_5\}$

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

$\Gamma = \{a, b, c, Z_0\}$

$q_0 = \{q_0\}$

$Z_0 = \{Z_0\}$

$F = \{q_5\}$

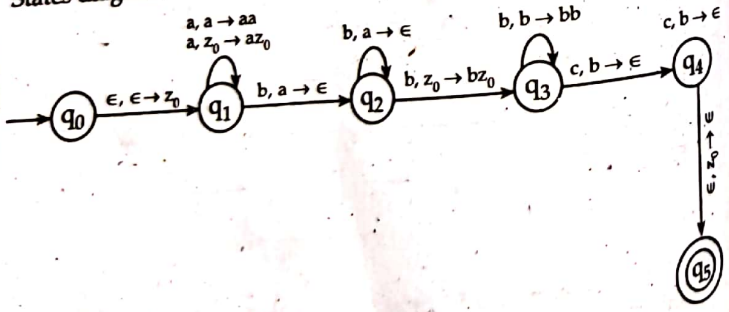
and transition (δ) is defined as follows:

$\delta = \{$

1. $((q_0, \in, \in), \quad (q_1, Z_0))$
2. $((q_1, a, Z_0), \quad (q_1, aZ_0))$
3. $((q_1, a, a), \quad (q_1, aa))$
4. $((q_1, b, a), \quad (q_2, \in))$
5. $((q_2, b, a), \quad (q_2, \in))$
6. $((q_2, b, Z_0), \quad (q_3, bZ_0))$
7. $((q_3, b, b), \quad (q_3, bb))$
8. $((q_3, c, b), \quad (q_4, \in))$
9. $((q_4, c, b), \quad (q_4, \in))$
10. $((q_4, \in, Z_0), \quad (q_5, \in))$

$\}$

States diagram



Test for input string 'abbbcc'

S.No.	State	Unread input	Stack	Transition used
1.	q_0	abbbcc	$\in$	–
2.	q_1	abbbcc	Z_0	1
3.	q_1	bbbcc	a Z_0	2
4.	q_2	bbcc	Z_0	4
5.	q_3	bcc	b Z_0	6
6.	q_3	cc	bb Z_0	7
7.	q_4	c	b Z_0	8
8.	q_4	$\in$	Z_0	9
9.	q_5	$\in$	$\in$	10

Accepted

Since, stack is empty and final state is also accepting state. So, string 'abbbcc' is accepted.

16. Define configuration of PDA. Design a PDA that accepts $L = \{a^{3n} b^n, n > 0\}$ and check the string aaaaaabb. **[2072 Chaitra]**

The configuration of PDA is

$$\delta (q, a, X) \rightarrow (q', k)$$

where

$q \rightarrow$ state of machine before input

$a \rightarrow$ input symbol in Σ

$X \rightarrow$ stack symbol on top of stack / symbol to be popped out.

The output of δ give pairs (q', k) where q' is new state

q' is new state

k is stack symbol that replaces X on the top of stack

Case I: If $k = \in$, then X is popped out

Case II: If $k : X$, then stack is unchanged

Case III: If $k = YX$, then X is pushdown into stack and Y is on top of stack (or X is popped out and YX is pushed down into the stack)

Let, the required PDA be

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

where

$\theta = \{q_0, q_1, q_2, q_3, q_4, q_5\}$

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

$\Gamma = \{a, b, Z_0\}$

$q_0 = \{q_0\}$

$Z_0 = \{Z_0\}$

$F = \{q_5\}$

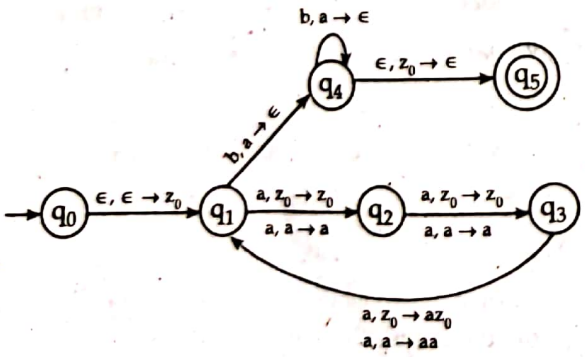
and transition (δ) is defined as follows:

$\delta = \{$

1. $((q_0, \epsilon, \epsilon), \quad (q_1, Z_0))$
2. $((q_1, a, Z_0), \quad (q_2, Z_0))$
3. $((q_2, a, Z_0), \quad (q_3, Z_0))$
4. $((q_3, a, Z_0), \quad (q_1, aZ_0))$
5. $((q_3, a, a), \quad (q_1, aa))$
6. $((q_1, b, a), \quad (q_4, \epsilon))$
7. $((q_4, b, a), \quad (q_4, \epsilon))$
8. $((q_4, \epsilon, Z_0), \quad (q_5, \epsilon))$
9. $((q_1, a, a), \quad (q_2, a))$
10. $((q_2, a, a), \quad (q_3, a))$

$\}$

States diagram



Test for input string 'abbbcc'

S.No.	State	Unread input	Stack	Transition used
1.	q_0	aaaaaabb	ϵ	-
2.	q_1	aaaaaabb	Z_0	1
3.	q_2	aaaaabb	Z_0	2
4.	q_3	aaaabb	Z_0	3
5.	q_1	aaabb	$a Z_0$	4
6.	q_2	aabb	$a Z_0$	9
7.	q_3	abb	$a Z_0$	10
8.	q_1	bb	$aa Z_0$	5
9.	q_4	b	$a Z_0$	6
10.	q_4	ϵ	Z_0	7
11.	q_5	ϵ	ϵ	8

Since, stack is empty and final state is also accepting state. So, string 'aaaaaabb' is accepted

Accepted

17. Design a pushdown automaton to accept $L = \{ww^R \in \{a, b\}^*$. Show how it accepts the string 'abbbba". [2072 Kartik, Back]

Let, the required PDA be

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

where

$\theta = \{q_1, q_2, q_3, q_4\}$

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

$\Gamma = \{a, b, Z_0\}$

$q_0 = \{q_1\}$

$Z_0 = \{Z_0\}$

$F = \{q_4\}$

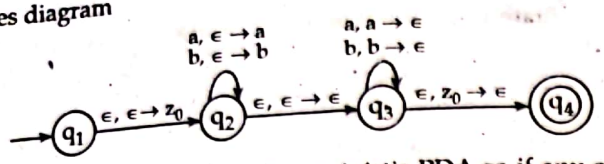
and transition (δ) is defined as follows:

$\delta = \{$

1. $((q_1, \epsilon, \epsilon), \quad (q_2, Z_0))$
2. $((q_2, a, \epsilon), \quad (q_2, a))$
3. $((q_2, b, \epsilon), \quad (q_2, b))$
4. $((q_2, \epsilon, \epsilon), \quad (q_3, \epsilon))$
5. $((q_3, a, a), \quad (q_3, \epsilon))$
6. $((q_3, b, b), \quad (q_3, \epsilon))$
7. $((q_3, \epsilon, Z_0), \quad (q_4, \epsilon))$

$\}$

States diagram



Here, the given PDA is non-deterministic PDA so if any of the final states is accepting state, then string will be accepted. In above state diagram, we can see there is empty transition on input empty '∈' symbol between states q_2 and q_3. Any string can be resolved into strings like there exists empty symbol '∈' between two consecutive alphabet in string i.e. abba → ∈ a ∈ b ∈ b ∈ a ∈.

In many of the final state of given non-deterministic PDA, final state will be accepting state if empty symbol '∈' is taken as input exactly in middle or centre of the string. And final state will be rejecting state of any input empty symbol '∈' is taken before or after that except the beginning and the end. And after that, if matches the input with previously stored element in stack and POPS out it. And finally stack becomes empty and string is accepted if it follows ww^R.

Test for input string 'abbbba'

S.No.	State	Unread input	Stack	Transition used
1.	q_1	abbbba	∈	–
2.	q_2	abbbba	Z_0	1
3.	q_2	bbbba	a Z_0	2
4.	q_2	bbba	ba Z_0	3
5.	q_2	bba	bba Z_0	3
6.	q_3	bba	bba Z_0	4
7.	q_3	ba	ba Z_0	6
8.	q_3	a	a Z_0	6
9.	q_3	∈	Z_0	5
10.	q_4	∈	∈	7

Accepted

Note: In S. No. 5, transition $\delta(q_2, \in, \in) \rightarrow (q_3, \in)$ is used as centre of string is reached.

Since, stack is empty and the final state is also accepting state. So, string 'abbbba' is accepted.

18. Describe the transition function of push-down automata.

Please see 2072 Chaitra [2071 Shrawan, Back]

19. Design a PDA that accepts language, $L = \{a^n b^{3n} : n \geq 1\}$. Test your design for string "abbb". [2071 Chaitra]

Let, the required PDA be

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

where

$\theta = \{q_0, q_1, q_2, q_3, q_4, q_5\}$

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

$\Gamma = \{a, b, Z_0\}$

$q_0 = \{q_0\}$

$Z_0 = \{Z_0\}$

$F = \{q_5\}$

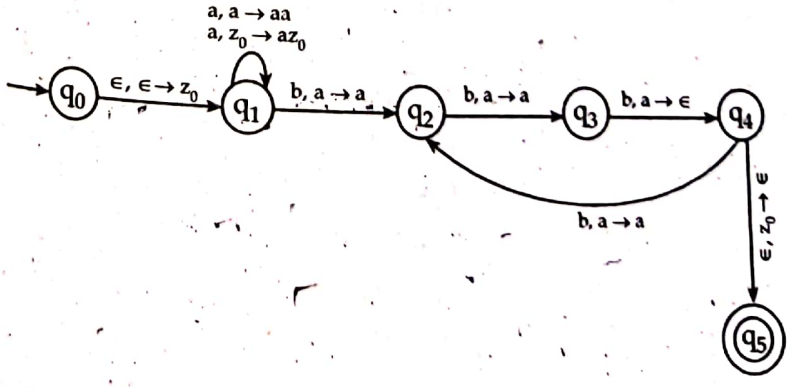
and transition (δ) is defined as follows:

$\delta = \{$

1. $((q_0, \in, \in), \quad (q_1, Z_0))$
2. $((q_1, a, Z_0), \quad (q_1, aZ_0))$
3. $((q_1, a, a), \quad (q_1, aa))$
4. $((q_1, b, a), \quad (q_2, a))$
5. $((q_2, b, a), \quad (q_3, a))$
6. $((q_3, b, a), \quad (q_4, a))$
7. $((q_4, b, a), \quad (q_2, a))$
8. $((q_4, \in, Z_0), \quad (q_5, \in))$

$\}$

States diagram



Test for input string 'abbb'

S.No.	State	Unread input	Stack	Transition used
1.	q_0	abbb	ϵ	-
2.	q_1	abbb	Z_0	1
3.	q_1	bbb	a Z_0	2
4.	q_2	bb	a Z_0	4
5.	q_3	b	a Z_0	5
6.	q_4	ϵ	Z_0	6
7.	q_5	ϵ	ϵ	8

Accepted

Since, stack is empty and final state is also accepting state, so bb accepted.

20. Design a PDA that accepts all the palindromes defined over {a, b}*. Your design should accept strings like ϵ, a, b, aba, bab, abba, babab, etc. [2070 Chaitra]

Let, the required PDA (non-deterministic) be

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

where

$\theta = \{q_0, q_1, q_2, q_3\}$

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

$\Gamma = \{a, b, Z_0\}$

$q_0 = \{q_0\}$

$Z_0 = \{Z_0\}$

$F = \{q_3\}$

and transition (δ) is defined as follows:

$\delta = \{$

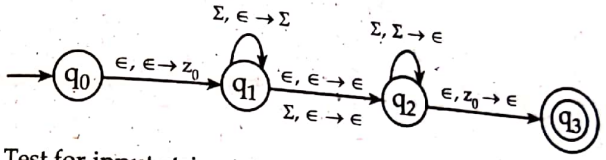
1. $((q_0, \epsilon, \epsilon), \quad (q_1, Z_0))$
2. $((q_1, \Sigma, \epsilon), \quad (q_1, \Sigma))$
3. $((q_1, \epsilon, \epsilon), \quad (q_2, \epsilon))$
4. $((q_1, \Sigma, \epsilon), \quad (q_2, \epsilon))$
5. $((q_2, \Sigma, \Sigma), \quad (q_2, \epsilon))$
6. $((q_2, \epsilon, Z_0), \quad (q_3, \epsilon))$

$\}$

Here, transition $((q_1, \Sigma, \epsilon), (q_1, \Sigma))$ means that on every input alphabet to state q_1, PDA doesn't POP, out anything but it pushes that read input into stack and stays in q_1. All the other transitions including 'Σ' symbol can be understood in similar manner.

Here, the given PDA is non-deterministic. So, if any of the final states is accepting state, then string will be accepted. In between q_1 and q_2, PDA chooses transition $\epsilon, \epsilon \rightarrow \epsilon$ for even-palindromes whereas it chooses transition $\Sigma, \epsilon \rightarrow \epsilon$ for odd palindromes non-deterministically which leads to accepting state. It is guaranteed that all the possible transitions between q_1 and q_2 except the mentioned above leads to rejecting state.

States diagram



Test for input string 'ϵ'

S.No.	State	Unread input	Stack	Transition used
1.	q_0	ϵ	ϵ	-
2.	q_1	ϵ	Z_0	1
3.	q_2	ϵ	Z_0	3
4.	q_3	ϵ	ϵ	6

Accepted

Test for input string 'a'

S.No.	State	Unread input	Stack	Transition used
1.	q_0	a	ϵ	-
2.	q_1	a	Z_0	1
3.	q_2	ϵ	Z_0	4
4.	q_3	ϵ	ϵ	6

Accepted

Parse tree

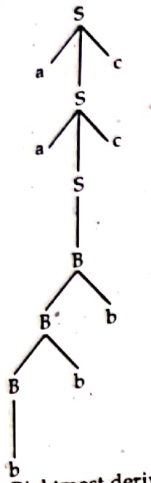


Fig. Rightmost derivation

22. Define pumping lemma for context free language prove that language L = {ww / w∈ {1, 0}* is not context free. [2070 Ashwin, Back]

Please see theory part and its example -2 for proof.

23. Define context free grammar (CFG). Show that L = {$a^n b^{2n} c^{3n}$: n > 0} is not context free language by using pumping lemma for CFL. [2074 Chaitra]

For formal definition of CFG, please see theory part.

Pumping lemma states that, "if a language A is context free language, then it must have a pumping length 'p' such that any string 's' ∈ A having length is S ≥ p can be divided into 5 parts uvxyz such that following conditions must be true.

(1) $uv^i xy^i z \in A$ for every $i \geq 0$.

(2) $|vy| \geq 1$

(3) $|vxy| \leq p$

We will prove it by contradiction using pumping lemma.

Let us assume language L = {$a^n b^{2n} c^{3n}$: n > 0} be CFL (context free language)

Let, $S \in L = a^p b^{2p} c^{3p}$

Let p = 3

Then, $S = a^3 b^6 c^9$

= aaa bbbbbb ccccccccc

Dividing S into u, v, x, y and z, we get

Case I: When v and y contain only one kind of symbol is

S = a aa bbbbbbbccccc cc cc
u v x y z

checking for condition 1

Let, i = 2

$S = uv^2 xy^2z$

= a aaaa bbbbbb ccccccccccc

$= a^5 b^6 c^{11} \notin L$

Here, condition 1 fails we can see that 'b' is not twice as many as 'a' and 'c' is not thrice as many as 'a'.

Case II: when either v or y contains two different kind of symbol.

S = ab ab bbbbb ccccc cc cc
u v x y z

checking for condition 1

Let, i = 2

$S = uv^2 xy^2z$

= aa abab bbbbb ccccccccc ∉ L

Here, again condition 1 fails as we can see pattern is not followed.

So, none of the cases follow all three conditions of context free language as stated by pumping lemma. Therefore language L is not context free language (CLF).

24. Define ambiguity in CFG write CFG for L = {w {a, b}* : w is a palindrome} and also draw parse tree for the derivation of any two strings of length even and odd. [2073, Chaitra]

For ambiguity in CFG, please see 'ambiguous grammars in theory part.

Let G = (V, Σ, R, S) be the required CFG

where,

V = {S, a, b}

Σ = {a, b}

R = {

$S \rightarrow a\,Sa / bsb / a / b / \in$

(For derivation of string of even length, we put $S \rightarrow \in$ while terminating symbol 's' whereas we put S → a or b according to our requirement in case of string of odd length).

Let two strings of even length be 'baab' and 'abaaba'

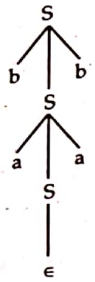
Derivation of string 'baab'

$S \Rightarrow bSb$

$\Rightarrow baSab$

$\Rightarrow baab$

Parse tree



Derivation of string 'abaaba'

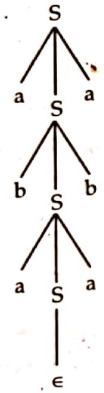
$S \Rightarrow aSa$

$\Rightarrow abSba$

$\Rightarrow abaSaba$

$\Rightarrow abaaba$

Parse tree



Let two strings of odd length be 'baaab'

Derivation of string 'baaab'

$S \Rightarrow bSb$

$\Rightarrow baSab$

$\Rightarrow baaab$

Parse tree

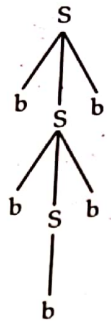
Derivation of string 'bbb bb'

$S \Rightarrow bSb$

$\Rightarrow bbSbb$

$\Rightarrow bbbbb$

Parse tree



25. Construct a CFG for the language $L = a^n b^{2n}$, $n > 0$ and use this grammar to generate string aabbbb. Also, construct the parse tree.

[2073 Shrawan, Back]

Let, $G = (V, \Sigma, R, S)$ be the required CFG for $L = a^n b^{2n} : n > 0$.

where,

$V = \{S, a, b\}$

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

and production R is defined as

$R = \{$

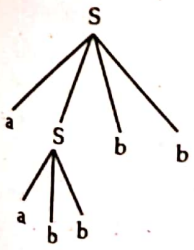
$S \rightarrow aSbb \mid abb$

Let us derive string aabbbb

$S \rightarrow aSbb$

$\rightarrow aabbbb$

Parse tree



26. Write a CFG for the regular expression $R = 0^* 1 (0 \cup 1)^*$ [2072 Chaitra]

Let, $G = \{V, \Sigma, R, S\}$ be the required CFG.

where

$V = \{S, A, B, 0, 1\}$

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

Production R is defined as

$R = \{$

$S \rightarrow A\,1\,B$

$A \rightarrow 0A / \in$

$B \rightarrow 0B / 1B / \in$

$\}$

Let, us derive string '01001'

$S \Rightarrow A\,1\,B$

$\Rightarrow 0\,A\,1\,B$

$\Rightarrow 01\,B$

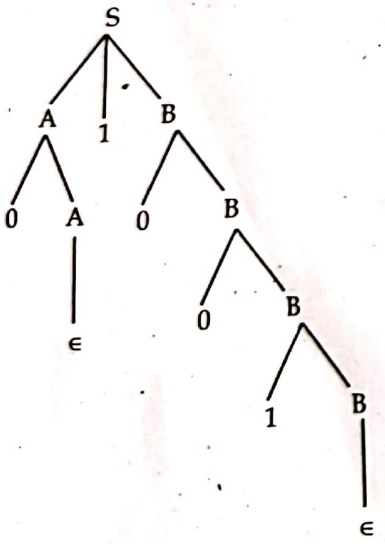
$\Rightarrow 010\,B$

$\Rightarrow 0100\,B$

$\Rightarrow 01001\,B$

$\Rightarrow 01001$

Parse tree



Let, us derive string '1001'

$S \Rightarrow A\,1\,B$

$\Rightarrow 1\,B$

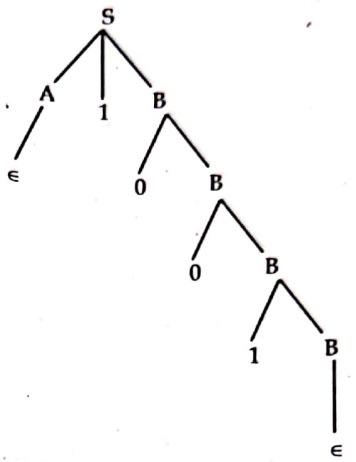
$\Rightarrow 10\,B$

$\Rightarrow 100\,B$

$\Rightarrow 1001\,B$

$\Rightarrow 1001$

Parse tree



27. Use concept of closure property to prove that intersection of context free language is not context free. [2072 Chaitra]

We know that languages

$L_1 = \{ a^n b^m c^m / n, m \geq 0\}$

and $L_2 = \{a^m b^m c^n / n, m \geq 0\}$

are context free. Now let see L, $n < 2 = \{a^n b^n c^n / n \geq 0\}$ is context free. We have proven that language $L = L_1 \cap L_2 = \{a^n b^n c^n / n \geq 0\}$ is not context free using pumping lemma. So, it is proven that intersection of context free languages is not context free.

28. Define the term ambiguity and inherent ambiguity in parse tree for a CFG given by $G = \{V, \Sigma, R, S\}$ with $V = \{S\}$, $\Sigma = \{a\}$ and production rules R is defined as

$S \rightarrow S\,S$

$S \rightarrow a$

Obtain the language L(G) generated by this grammar. [2071 Chaitra]

The term ambiguity refers to a grammar $G = \{V, \Sigma, R, S\}$ which is said to be ambiguous if any string $S \in L(G)$ has two or more than two leftmost derivation, rightmost derivation or parse trees for its derivation.

A "inherently ambiguous language" is a language for which no unambiguous grammar exists.

A language 'L' is said to be "inherently ambiguous in parse tree if every construction of parse tree that generates language L is ambiguous." Otherwise it is said to be unambiguous in parse tree.

Given, CFG

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

where $V = \{S\}$

$\Sigma = \{a\}$

$R = \{$

$S \rightarrow SS$

$S \rightarrow a$

$\}$

Let us start with start symbol 'S'

$S \Rightarrow SS$

$\Rightarrow aS$

$\Rightarrow aSS$

$\Rightarrow aaS$

$\Rightarrow :$

$\Rightarrow .$

$\Rightarrow a\,a \ldots a^{n-1} S$

$\Rightarrow a\,a \ldots a^{n-1} a$

$\Rightarrow a^n$

Therefore, language L(G) generated by this grammar is

$L(G) = \{a^n : n \geq 2\}$

29. Using the pumping theorem for context free languages show that $L = \{a^n b^n c^n : n \geq 0\}$ is not context free. [2072 Kartik, Back]

Please see the example 1 of pumping lemma in theory part.

30. Write context free grammars (CFG) for the language $L_1 = \{a^m b^n c^n : m \geq 1, n \geq 1\}$ the $L_2 = \{a^n b^n c^m : m \geq 1, n \geq 1\}$. Do you think that $L = (L_1 \cap L_2)$ is also context free? If not prove that the language thus obtained is not context free by using pumping lemma for context free languages. [2070 Chaitra]

Given, languages L_1 and L_2 are

$L_1 = \{a^m b^n c^n : m \geq 1, n \geq 1\}$

$L_2 = \{a^n b^n c^m : m \geq 1, n \geq 1\}$

Let, $G_1 = (V, \Sigma, R, S)$ be the required CFG for language L_1 where

$V = \{S, A, B, a, b, c\}$

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

$R = \{$

$S \rightarrow AB$

$A \rightarrow aA / a$

$B \rightarrow b\,BC / bc$

$\}$

Let us generate string 'aaabbcc'

$S \Rightarrow AB$

$\Rightarrow a\,AB$

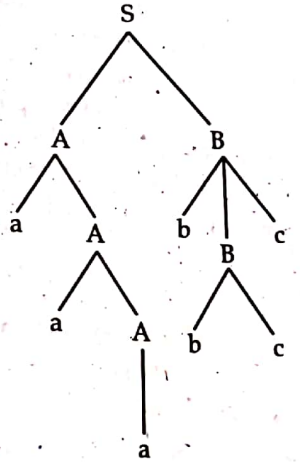
$\Rightarrow aa\,AB$

$\Rightarrow aaa\,B$

$\Rightarrow aaa\,b\,Bc$

$\Rightarrow aaa\,b\,bc\,c$

Parse tree



Let $G_2 = (V, \Sigma, R, S)$ be the required CFG for the language L_2 where

$V = \{S, M, N, a, b, c\}$

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

$R = \{$

$S \rightarrow MN$

$M \rightarrow a\,M\,b / a\,b$

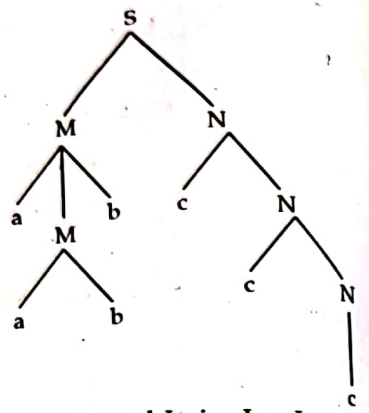
$N \rightarrow c\,N / c$

$\}$

Let us generate string 'aabbccc'

$S \Rightarrow MN$
$\Rightarrow aMbN$
$\Rightarrow aabbN$
$\Rightarrow aabbcN$
$\Rightarrow aabbccN$
$\Rightarrow aabbcc$

Parse tree



Let L be the intersection of the language L_1 and L_2 i.e. $L = L_1 \cap L_2$ $= \{a^n b^n c^n : n \geq 1\}$. Here, the language L is not context free language (CFL) we will prove it using pumping lemma.

Pumping lemma states that, "if a language L is context free language, then it must have a pumping length 'p' such that any string 's' $\in$ L having length $|S| \geq p$ can be divided into S parts uvxyz such that following conditions must be true.

(1) $uv^i xy^i z \in L$ for every $i \geq 0$
(2) $|vy| \geq 1$
(3) $|vxy| \leq p$

To prove: L is not CFL where $L = \{a^n b^n c^n : n \geq 1\}$

Proof: Let, $S \in L = a^p b^p c^p$

Let, $p = 3$

Then $S = a^3 b^3 c^3$
$= aaa\ bbb\ ccc$

Dividing S into u, v, x, y and z, we get.

Case I: When 'v' any 'y' contain only one type of symbol.

$S = \underbrace{a}_{u}\ \underbrace{a}_{v}\ \underbrace{abbbc}_{x}\ \underbrace{c}_{y}\ \underbrace{c}_{z}$

checking for condition 1,

Let, i = 2

$S = uv^2 xy^2 z$
$= a\ aa\ abbbc\ cc\ c$
$= a^4 b^3 c^4 \notin L$

Here, condition 1 fails.

Case II: Either 'v' or 'y' contains more than one kind of symbol.

$S = \underbrace{aa}_{u}\ \underbrace{ab}_{v}\ \underbrace{bbc}_{x}\ \underbrace{c}_{y}\ \underbrace{c}_{z}$

Checking for condition 1,

Let, i = 2

$S = uv^2 xy^2 z$
$= a\ a\ a\ b\ ab\ bbc\ ccc \notin L$

Here, it doesn't follow the pattern $a^n b^n c^n$. So, condition 1 fails. So, none of the cases satisfy all the three conditions of context free language as stated by pumping lemma. Therefore, language L which is intersection of two CFL L_1 and L_2 and is assumed to be CFL is not context free, language CFL using pumping lumping lemma. Hence, proved.

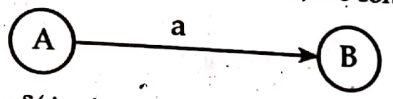
31. Consider the regular grammar G = (V, Σ, R, S) where,

V = {S, A, B, a, b}, Σ = {a, b}
R = {
S → abA / B / baB / e
A → bS / a
B → aS
}

[2070 Chaitra]

The rules for constructing equivalent finite automata for regular grammar

(a) For every production, A → aB, we construct an edge.

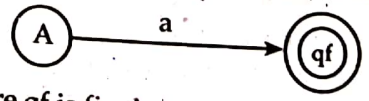


where $\delta(A, a) = B$

(b) For every production, A → B we construct an edge

where $\delta(A, \in) = B$

(c) For every production A → a, we construct an edge



where qf is final state.

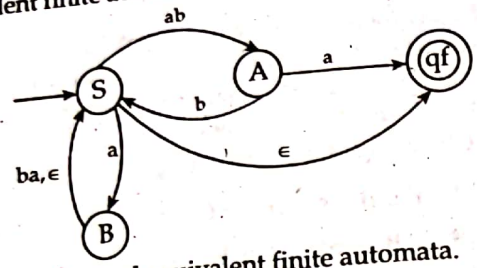
(d) For every production, A → e or ∈, we make state the final state i.e.

Note that the number of states is equal to the number of non-terminals plus one

Given,

$R = \{$
$S \rightarrow abA / B / baB / e$
$A \rightarrow bS / a$
$B \rightarrow aS$
$\}$

Equivalent finite automata such that L(M) = L(G).



This is the required equivalent finite automata.

32. Describe the transition function of push-down automata. **[2072 Shrawan, Back]**

The transition function (δ) of push-down automata (non-deterministic) is given as

$$\delta : (\theta \times \Sigma^* \times \Gamma^*) \rightarrow (\theta' \times x)$$

where

- θ → current state of the control unit
- Σ^* → current input symbol
- Γ^* → current symbol on top of the stack.
- θ' → next state
- x → string that is put on top of the stack in place of single symbol there before.

For example;

(a) The interpretation of $((q, a, z), (p, y)) \in \delta$ where $p, q \in \theta$, a is an alphabet, z and y in Γ^*) is that PDA whenever is in state q, with z on the top of the stack may read 'a' from the input tape, replace z by y on the top of the stack and enter state p.

(b) To push a symbol on the stack, just add it on the top of the stack. This can be achieved by the transition $((q, a, \epsilon), (p, a))$.

(c) Similarly, to POP a symbol is to remove it from the top of the stack. The transition $((q, a, z), (p, \epsilon))$ pops z from the stack.

(d) Suppose PDA is in state 'p', reads input 'a' with 'z' on top of stack. And PDA wants to stay in same state and do nothing neither POP nor push. Then, it can be achieved by following transition

$((p, q, z), (p, z))$

Chapter 4

Turing Machine

Introduction

A Turing machine is like a pushdown automata with infinite input tape. It is the next model of computation under machine based approach and context sensitive or phase structure grammar in the corresponding model under grammatical approach.

A Turing machine consists of a finite control, a tape and a head that can be used for reading or writing on that tape. The tape consists of a infinite long series of "squares" each of which can hold a single symbol. The 'tape head' or 'read write head' can read a symbol from the tape, write a symbol to the tape and more one square in either direction. There are two types of Turing machines.

(a) Deterministic Turing machine.

(b) Non-deterministic Turing machine.

Here, we will discuss about deterministic Turing machines only.

Mathematical definition of Turing machine

A Turing machine M is a 7-tuple

$(\theta, \Sigma, \Gamma, b, \delta, q_0, F)$

where, θ is a finite set of states

Σ is a finite set of symbols "input alphabets"

Γ is a finite set of symbols "tape alphabets"

δ is the partial transition function $\delta : \theta \times \Gamma \rightarrow \theta \times \Gamma' \times \{L, R, N\}$

b is a symbol called 'blank symbol'

$q_0 \in \theta$ is the initial state.

$F \subseteq \theta$ is a set of final states.

Operation of Turing machine

In essence a Turing machine consists of a finite state control unit and a tape communication between two is provided by a single head which