

Module-1

Basic Notations and Terminologies used in Automata Theory:

Alphabet: A language consists of various symbols from which the words, statements etc., can be obtained. These symbols are called alphabets. Formally, an alphabet is defined as a finite non-empty set of symbols. The symbol Σ denotes the set of alphabets of a language.

(eg): $\Sigma = \{a, b\}$, $\Sigma = \{0, 1\}$

2.1 String: The sequence of symbols obtained from the alphabets of a language is called a string. Formally a string is defined as a finite sequence of symbols from the alphabet Σ .

Note: An empty string is denoted by the symbol ϵ (epsilon) or λ (lambda) & $\epsilon \notin \Sigma$

(eg) Let $\Sigma = \{a, b\}$ is set of alphabets.

The various strings that can be obtained from Σ are $\{a, b, aa, bb, aba, bba, \dots\}$

Concatenation of Two strings: The concatenation of two strings u and v is the string obtained by writing the letters of string u followed by the letters of string v (i.e. appending the symbols of v to the right of u)

eg: $u = a_1 a_2 a_3 \dots a_n$ & $v = b_1 b_2 b_3 \dots b_m$
 $uv = a_1 a_2 a_3 \dots a_n b_1 b_2 b_3 \dots b_m$

Substring: Let w is a string obtained from the symbols in Σ . The string w if it can be decomposed into ' n ' no ~~these~~ strings, such ' n ' no of strings are called as sub strings.

(eg): $w = xyz$

then x is a substring, y is a substring & z is a substring of string w .

Prefix: A prefix is string of any number of leading symbols.

(eg) In the string "sita", the various prefixes are ϵ , "s", "si", "sit" and "sita".

Suffix: The suffix is a string of any number of trailing symbols.

(eg) In the string "sita", the various suffixes are string ϵ , "a", "ta", "ita", "sita".

Reversal of a string: The reversal of a string is obtained by writing the symbols in reverse order.

(eg) $V = a_1 a_2 a_3 \dots a_n$

The reverse of V is denoted by V^R & is given by

$$V^R = a_n a_{n-1} \dots a_3 a_2 a_1.$$

Length of a string: The length of a string w is the number of symbols in w and is denoted by $|w|$.

(eg): $w = a_1 a_2 a_3 \dots a_n$

then length of w is given by

$$|w| = n.$$

Power of an alphabet: The power of an alphabet is denoted by Σ^i is the set of words of length i .

(eg) If $\Sigma = \{a, b\}$

Then

$\Sigma^0 = \{\epsilon\}$ denotes empty string.

$\Sigma^1 = \{a, b\}$ " set of words of length 1

$\Sigma^2 = \{aa, ab, ba, bb\}$ " " " " " 2

$\Sigma^3 = \{aaa, aab, aba, abb, bba, bab, bbb, baa\}$... length 3
& so on.

2.2 Language: A language can be defined as a set of strings obtained from Σ^* where Σ is set of alphabets of a particular language.
In other words, a language is subset of Σ^* which is denoted by $L \subseteq \Sigma^*$.

(eg) i) $\{\epsilon, 01, 0011, 000111, 0101, \dots\}$ A language of strings consisting of equal number of 0's & 1's.

ii) $\{\epsilon, ab, aabb, aaabbb, \dots\}$ A language of strings consisting of n number of 'a's followed by n number of 'b's.

Sentence: A string that belongs to a language is called word or sentence of that language.

(eg): $\Sigma = \{0, 1\}$

$\{\epsilon, 01, 10, 0011, 000111, \dots\}$

Each word separated by comma is a sentence (also called word)

So 01 is a sentence

10 " " "

0011 " " "

Language Hierarchy

Theorem: Concatenation & Reverse of Strings.

Theorem: If w and x are strings, then $(wx)^R = x^R w^R$

Eg: $(\text{name tag})^R = (\text{tag})^R (\text{name})^R = \text{gateman}$.

Proof: The proof is by induction on $|x|$.

Base case: $|x| = 0$

Then $x = \epsilon$

and $(wx)^R = (w\epsilon)^R = (w)^R = \epsilon w^R = \epsilon^R w^R = x^R w^R$

Prove: $\forall n \geq 0 ((|x| = n) \rightarrow ((wx)^R = x^R w^R)) \rightarrow ((|x| = n+1) \rightarrow ((wx)^R = x^R w^R))$

Consider any string x , where $|x| = n+1$.

Then $x = ua$ for some character a and $|u| = n$.

So

$$(wx)^R = (w(ua))^R$$

$$= ((wu)a)^R$$

$$= a(wu)^R$$

$$= a(u^R w^R)$$

$$= (au^R)w^R$$

$$= (ua)^R w^R$$

$$= x^R w^R$$

rewrite x as ua

associativity of concatenation

definition of reversal

induction hypothesis

associativity of concatenation

definition of reversal

rewrite ua as x .

2-2.2

Techniques for defining Languages

We use variety of techniques for defining the language that we wish to consider. Since languages are sets, we can define them using any of the set-defining techniques.

Ex 2.3: All a's precede All b's

Let $L = \{w \in \{a,b\}^* : \text{all } a\text{'s precede all } b\text{'s in } w\}$.

The strings $\epsilon, a, aa, aabbb, bb$ are in L .

Ex 2.4: Strings That End in a

Let $L = \{x : \exists y \in \{a,b\}^* (x = ya)\}$

$L = \{a, aa, aaaa, bbaa, \epsilon, ba\}$ are in L .

Ex 2.5: The perils of Using English to describe languages

Let $L = \{x \# y : x, y \in \{0,1,2,3,4,5,6,7,8,9\}^* \text{ and when } x \text{ and } y \text{ are viewed as the decimal representations of natural numbers, } \text{Square}(x) = y\}$

The strings $3 \# 9$ and $12 \# 144$ are in L .

The strings $3 \# 8, 12$ and $12 \# 12 \# 12$ are not in L .

Ex 2.6: The empty language

Let $L = \{\} = \emptyset$, L is the language that contains no strings

Ex: Using Repetition to Define a Language

Let $L = \{a^n : n \geq 0\}$

$L = \{\epsilon, a, aa, aea, aaaaa, \dots\}$

Languages are sets, the computational definition of language, could specify either:

- * a language generator, which enumerates (lists) the elements of the language, or
- * a language recognizer, which decides whether or not a candidate string is in the language and returns True if it is and False if it isn't.

Lexicographic order (written $<_L$):

In certain cases, when enumeration of language L is considered, we may care about the order in which elements of L are generated. If there exists a total order D of the elements of E_L , then we can use D to define on L a useful total order called lexicographic order (written $<_L$)

* Shorter strings precede longer ones:

$$\forall x(\forall y((|x| < |y|) \rightarrow (x <_L y))), \text{ and}$$

* Of strings that are the same length, sort them in dictionary order using D .

We say that a program lexicographically enumerates the elements of L iff it enumerates them in lexicographic order.

Ex Lexicographic Enumeration

Let $L = \{x \in \{a, b\}^* : \text{all } a\text{'s precede all } b\text{'s}\}$.

The lexicographic enumeration of L is

$\epsilon, a, b, aa, ab, bb, aaa, aab, abb, bbb, aaaa, aaab, aabb, abbb, bbbb, aaaaa, \dots$

2.2.3 What is the cardinality of language?

The smallest language over any alphabet is $\emptyset$, whose cardinality is 0.

The largest language over any alphabet Σ is Σ^* .

$|\Sigma^*|$? Suppose $\Sigma = \emptyset$ then $\Sigma^* = \{\epsilon\}$ & $|\Sigma^*| = 1$

Since any language over Σ is a subset of Σ^* , the cardinality of every language is at least 0 & at most $|\Sigma^*|$. So all languages are either finite or countably infinite.

Theorem 2.2 : The cardinality of Σ^*

Theorem : If $\Sigma \neq \emptyset$ then Σ^* is countably infinite

Proof : The elements of Σ^* can be lexicographically enumerated by a straightforward procedure that :

* Enumerates all strings of length 0, then length 1, then length 2, and so forth.

* Within the strings of a given length, enumerates them in dictionary order.

This enumeration is infinite since there is no longest string in Σ^* . Since there exists an infinite enumeration of Σ^* , it is countably infinite.

2.2.4 How many languages are there?

The set of languages defined on Σ is $P(\Sigma^*)$, the power set of Σ^* or the set of all subsets of Σ^* . If $\Sigma = \emptyset$ then Σ^* is $\{\epsilon\}$ & $P(\Sigma^*)$ is $\{\emptyset, \{\epsilon\}\}$.

Theorem : An uncountable infinite number of languages

Theorem : If $\Sigma \neq \emptyset$ then the set of languages over Σ is uncountably infinite.

Proof : The set of languages defined on Σ is $P(\Sigma^*)$. By theorem the cardinality of Σ^* , Σ^* is countably infinite. If S is a countably infinite set, $P(S)$ is uncountably infinite. So $P(\Sigma^*)$ is uncountably infinite.

2.2.5 Functions on Languages

The set functions applied to languages are Union, Intersection, difference & complement to be useful. Complement will be defined with Σ^* as the universe unless we explicitly state otherwise.

Ex: Set functions applied to languages...

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

$L_1 = \{\text{strings with an even number of a's}\}$

$L_2 = \{\text{strings with no b's}\} = \{\epsilon, a, aa, aaa, aaaa, aaaaa, \dots\}$

$L_1 \cup L_2 = \{\text{all strings of just a's plus strings that contain b's \& an even number of a's}\}$

$L_1 \cap L_2 = \{\epsilon, aa, aaaa, aaaaaa, \dots\}$

$L_2 - L_1 = \{a, aaa, aaaaa, aaaaaa, \dots\}$

$\neg(L_2 - L_1) = \{\text{strings with at least one b}\} \cup \{\text{string with an even no. of a's}\}$

operations defined on strings are concatenation, Kleene star & reverse.

Concatenation: Let L_1 & L_2 be two languages defined over some alphabet Σ . Then their concatenation, written $L_1 L_2$ is:

$$L_1 L_2 = \{w \in \Sigma^* : \exists s \in L_1, (\exists t \in L_2 (w = st))\}$$

(eg) Let $L_1 = \{\text{Rat, cat, dog}\}$

$L_2 = \{\text{bone, food}\}$

$L_1 L_2 = \{\text{Ratbone, Ratfood, Catbone, Catfood, dogbone, dogfood}\}$

Kleene star: Let L be a language defined over some alphabet Σ . Then the Kleene star of L , written L^* is:

$$L^* = \{\epsilon\} \cup \{w \in \Sigma^* : \exists k \geq 1 (\exists w_1, w_2, \dots, w_k \in L (w = w_1 w_2 \dots w_k))\}$$

In other words: L^* is the set of strings that can be formed by concatenating together zero or more strings from L .

(5)

Ex: Kleene Star

Let $L = \{\text{cat}, \text{rat}, \text{dog}\}$ Then:

$$L^* = \{\epsilon, \text{cat}, \text{rat}, \text{dog}, \text{catcat}, \text{catrat}, \text{catdog}, \dots, \text{catratcatdog} \dots\}$$

Reverse of L: The reverse of L , written L^R is:

$$L^R = \{w \in \Sigma^* : w = x^R \text{ for some } x \in L\}$$

in other words, L^R is the set of strings that can be formed by taking some string in L & reversing it.

Theorem: concatenation & Reverse of Languages

Theorem: If L_1 and L_2 are languages then

$$(L_1 L_2)^R = L_2^R L_1^R$$

Proof: If x and y are strings, then

$$\forall x (\forall y ((xy)^R = y^R x^R))$$

$$(L_1 L_2)^R = \{(xy)^R : x \in L_1 \text{ and } y \in L_2\}$$

$$= \{y^R x^R : x \in L_1 \text{ and } y \in L_2\}$$

$$= \underline{\underline{L_2^R L_1^R}}$$

✓

Types of finite Automata

There are three types of finite Automata

- 1) Deterministic finite Automata state machine (DFSM)
- 2) Non deterministic finite state machine (NDFSM)
- 3) Non deterministic finite state machine with ϵ -moves (ϵ -NDFSM)

Deterministic Finite State Machine

A Finite State Machine (FSM) is a computational device whose input is a string and whose output is one of two values that we can call Accept & Reject. FSMs are also sometimes called finite state automata or FSAs.

A deterministic FSM (or DFSM) M is a quintuple

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

where

Q is non empty finite set of states
 Σ is non empty finite set of input alphabets
 δ is transition function which is a mapping from $Q \times \Sigma$ to Q based on the current state & input symbol.

$q_0 \in Q$ is the start state

$F \subseteq Q$ is the set of accepting states.

δ defines the operation of a DFSM M one step at a time we can use it to define the sequence of configurations that M will enter. we start by defining the relation yields-in-one-step, written $\boxed{\vdash_M}$. yields-in-one step relates configuration₁ to configuration₂ iff M can move from conf₁ to conf₂ in one step.

we can write

$$C_1 \vdash_M^* C_2$$

When configuration C_1 yields configuration C_2 iff M can go from C_1 to C_2 in zero or more steps.

A computation by M is a finite sequence of configurations $C_0, C_1, \dots, C_n$ for some $n \geq 0$ such that:

- * C_0 is an initial configuration.

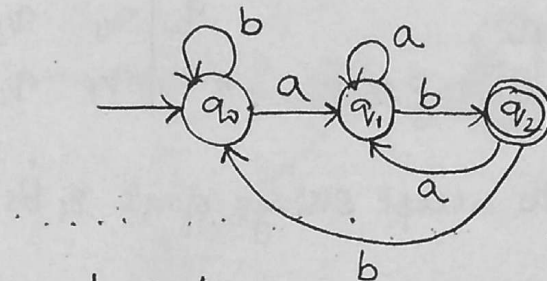
- * C_n is of the form (q, ϵ) , for some state $q \in Q_M$

(i.e., the entire input string has been read) and

- * $C_0 \vdash_M C_1 \vdash_M C_2 \vdash_M \dots \vdash_M C_n$.

(Eg) Draw a DFMSM to accept strings of a's & b's ending with the string ab.

Soln



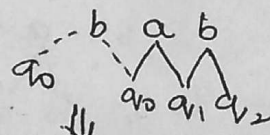
$$M = (Q, \Sigma, \delta, q_0, F)$$

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

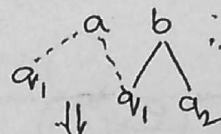
$$\Sigma = \{a, b\}$$

q_0 is start state

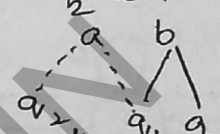
q_2 is final or accepting state



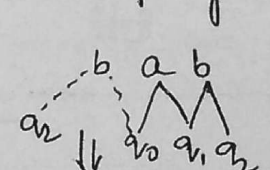
$$\delta(q_0, b) = q_0$$



$$\delta(q_1, a) = q_1$$



$$\delta(q_2, a) = q_1$$

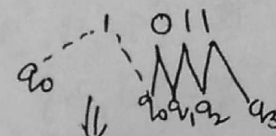
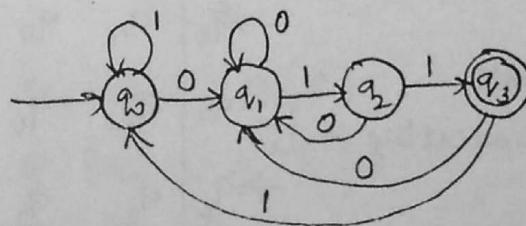


$$\delta(q_2, b) = q_0$$

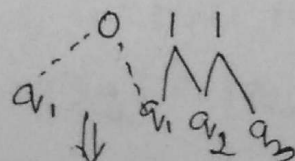
δ	a	b
$\rightarrow q_0$	q_1	q_0
q_1	q_1	q_2
q_2	q_1	q_0

(Eg) obtain a DFMSM to accept strings of 0's & 1's ending with string 011.

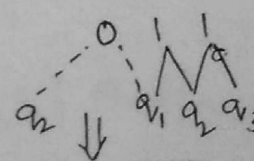
Soln



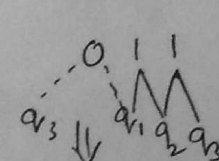
$$\delta(q_0, 1) = q_0$$



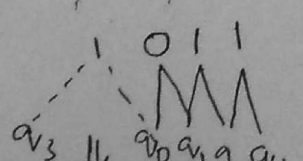
$$\delta(q_1, 0) = q_1$$



$$\delta(q_2, 0) = q_2$$



$$\delta(q_3, 0) = q_1$$



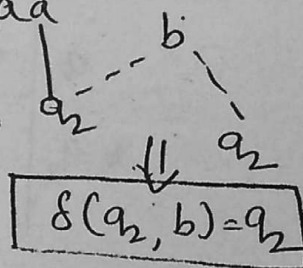
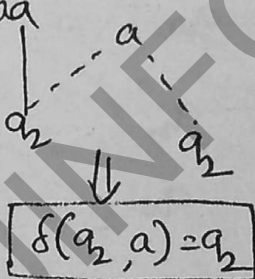
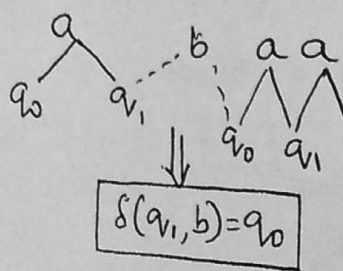
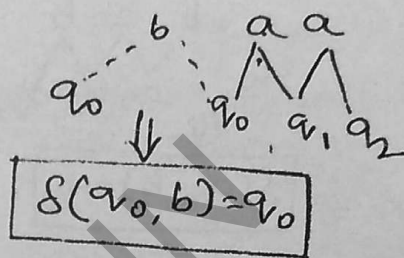
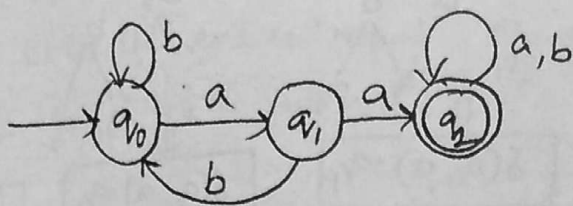
$$\delta(q_3, 1) = q_0$$

$M = (Q, \Sigma, \delta, q_0, F)$
 $Q = \{q_0, q_1, q_2, q_3\}$
 $\Sigma = \{0, 1\}$
 q_0 is start state
 q_3 is final or accepting state

δ	0	1
$\rightarrow q_0$	q_1	q_0
q_1	q_1	q_2
q_2	q_1	q_3
$*q_3$	q_1	q_0

(eg) obtain the DFSA to accept strings of a's & b's having substring aa

soln

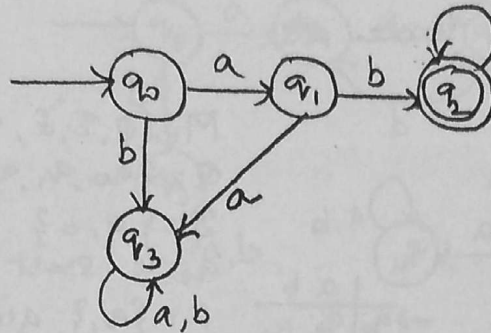


$M = (Q, \Sigma, \delta, q_0, F)$
 $Q = \{q_0, q_1, q_2\}$
 $\Sigma = \{a, b\}$
 q_0 is start state
 q_2 is final or accepting state

δ	a	b
$\rightarrow q_0$	q_1	q_0
q_1	q_2	q_0
$*q_2$	q_2	q_2

- (Eq) obtain a DFSM to accept the strings of a's & b's starting with string ab.

Soln



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

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

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

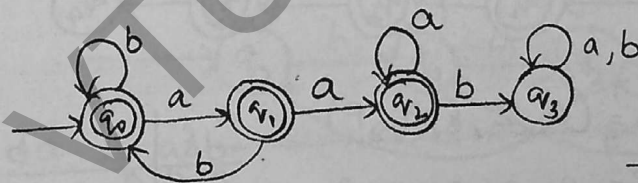
q_0 is start state

q_2 is final or accepting state

δ	a	b
$\rightarrow q_0$	q_1	q_3
q_1	q_3	q_2
$*q_2$	q_2	q_2
q_3	q_3	q_3

- (Eq) obtain a DFSM to accept a strings of a's & b's except those containing the substring aab

Soln



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

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

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

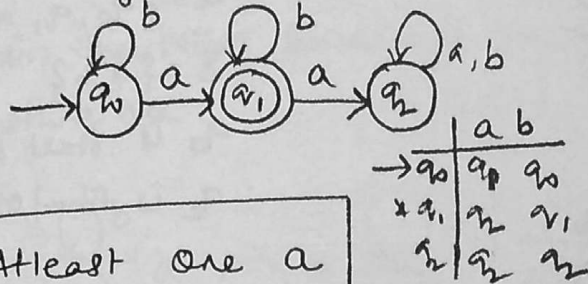
q_0 is start state

$F = \{q_1, q_2, q_3\}$ final/accepting state

δ	a	b
$\rightarrow *q_0$	q_1	q_0
$*q_1$	q_2	q_0
$*q_2$	q_2	q_3
q_3	q_3	q_3

- Q. Obtain a DFSA to accept strings of a's & b's having
- Exactly one a
 - atleast one a
 - not more than three a's

Solⁿ (i) Exactly one a



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

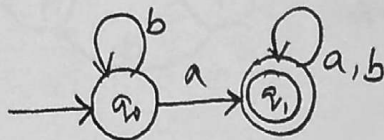
$$Q = \{q_0, q_1, q_2\}$$

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

q_0 is start state

$F = \{q_1\}$ accepting state

(ii) Atleast one a



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

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

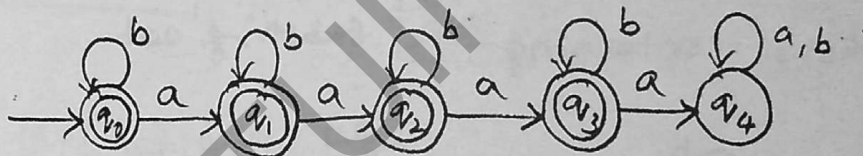
$$\Sigma = \{a, b\}$$

q_0 is start state

$F = \{q_1\}$ final/accepting state

δ	a	b
→ q ₀	q ₁	q ₀
* q ₁	q ₁	q ₁

(iii) Not more than three a's



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

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

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

q_0 is start state

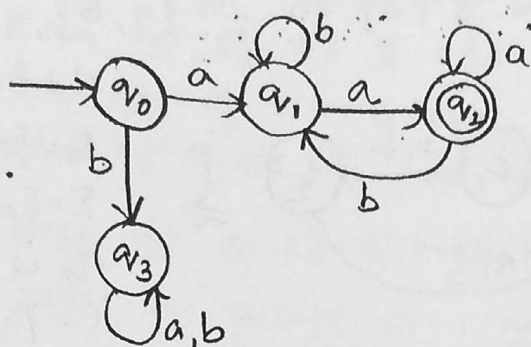
$$F = \{q_0, q_1, q_2, q_3\}$$

δ	a	b
→ q ₀	q ₁	q ₀
* q ₁	q ₂	q ₁
* q ₂	q ₃	q ₂
* q ₃	q ₄	q ₃
q ₄	q ₄	q ₄

9

(Eg) Obtain DFSM to accept the language $L = \{awa \mid w \in (a+b)^*\}$

Solⁿ



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

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

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

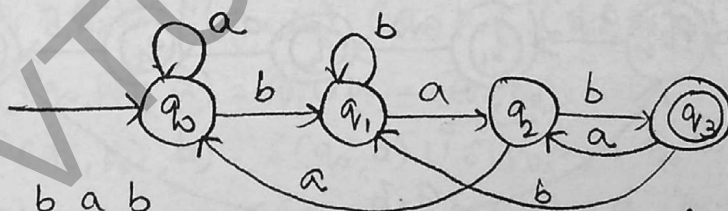
q_0 is start state

q_2 is Final state

δ	a	b
$\rightarrow q_0$	q_1	q_3
q_1	q_2	q_1
$* q_2$	q_2	q_1
q_3	q_3	q_3

(Eg) Obtain a DFSM to accept the strings ending with bab (or) obtain a DFSM to accept the language $L = \{w \mid w \text{ ends with } bab \mid w \in (a+b)^*\}$

Solⁿ



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

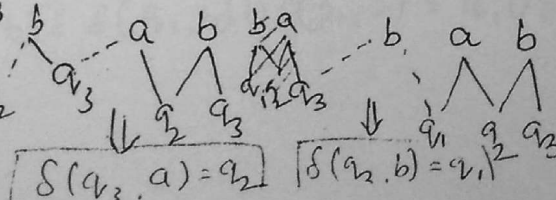
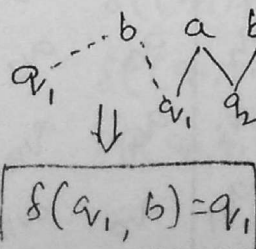
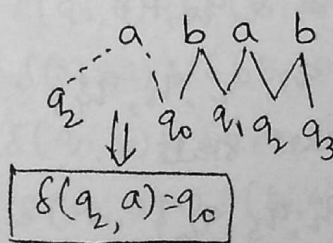
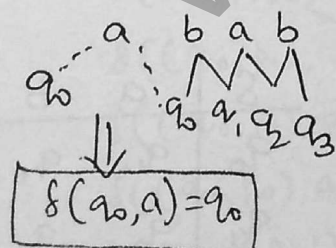
$Q = \{q_0, q_1, q_2, q_3\}$

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

q_0 is start state

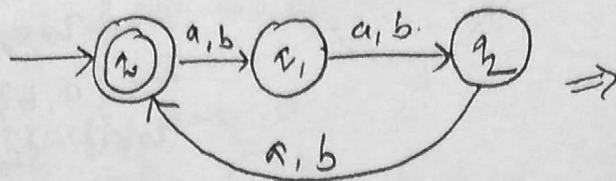
$F = \{q_3\}$

δ	a	b
$\rightarrow q_0$	q_0	q_1
q_1	q_2	q_1
q_2	q_0	q_3
$* q_3$	q_2	q_1

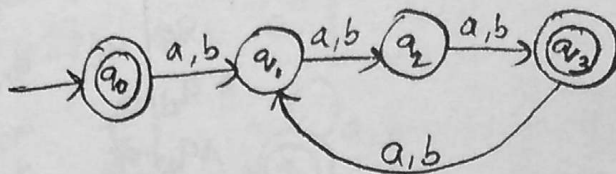


Ex. Obtain a DFSA to accept the language
 $L = \{w \mid |w| \bmod 3 = 0\}$ on $\Sigma = \{a, b\}$

Soln



or



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

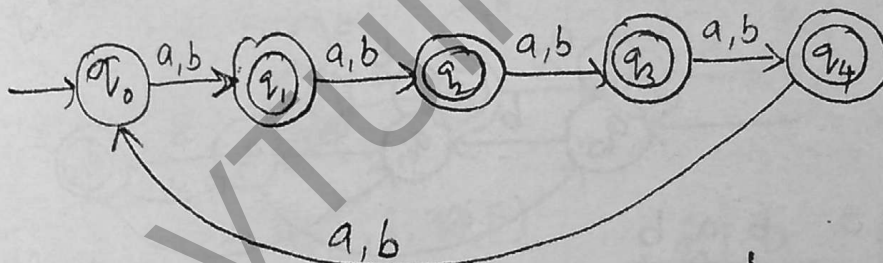
$Q = \{q_0, q_1, q_2\}$

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

q_0 is start state

$F = \{q_0\}$	δ	a	b
$\rightarrow q_0$		q_1	q_1
q_1		q_2	q_2
q_2		q_0	q_0

Ex. Obtain a DFSA to accept the language
 $L = \{w \mid |w| \bmod 5 \neq 0\}$ on $\Sigma = \{a, b\}$



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

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

q_0 is start state

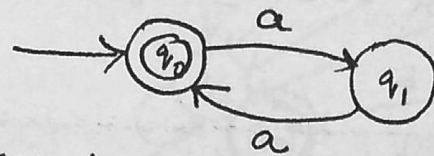
$F = \{q_1, q_2, q_3, q_4\}$

δ	a	b
q_0	q_1	q_1
q_1	q_2	q_2
q_2	q_3	q_3
q_3	q_4	q_4
q_4	q_0	q_0

Eg obtain a DFMSM to accept strings of a's & b's having

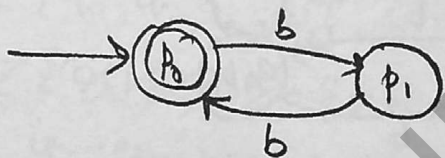
- (i) Even no. of a's & even no. of b's
- (ii) Even no. of a's & odd no. of b's
- (iii) odd no. of a's & even no. of b's
- (iv) odd no. of a's & odd no. of b's

Solⁿ: (i) Even no. of a's & even no. of b's
DFMSM to accept even no. of a's



$$Q_1 = \{q_0, q_1\}$$

DFMSM to accept ~~odd~~ Even no. of b's



$$Q_2 = \{p_0, p_1\}$$

For Even no. of a's & Even no. of b's take the cross product of $Q_1 \times Q_2$

$$\therefore Q_1 \times Q_2 = \{(q_0, p_0), (q_0, p_1), (q_1, p_0), (q_1, p_1)\}$$

$$\delta((q_0, p_0), a) = q_1 \cup p_0 = (q_1, p_0)$$

$$\delta((q_0, p_0), b) = \delta(q_0, b) \cup \delta(p_0, b) = q_0 \cup p_1 = (q_0, p_1)$$

$$\delta((q_0, p_1), a) = \delta(q_0, a) \cup \delta(p_1, a) = q_1 \cup p_1 = (q_1, p_1)$$

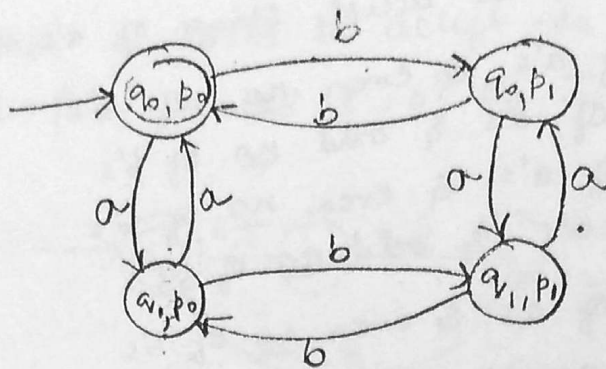
$$\delta((q_0, p_1), b) = \delta(q_0, b) \cup \delta(p_1, b) = q_0 \cup p_0 = (q_0, p_0)$$

$$\delta((q_1, p_0), a) = \delta(q_1, a) \cup \delta(p_0, a) = q_0 \cup p_0 = (q_0, p_0)$$

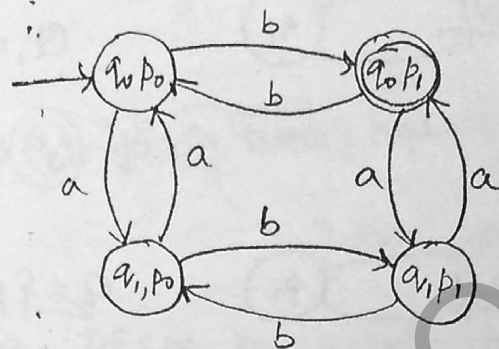
$$\delta((q_1, p_0), b) = \delta(q_1, b) \cup \delta(p_0, b) = q_1 \cup p_1 = (q_1, p_1)$$

$$\delta((q_1, p_1), a) = \delta(q_1, a) \cup \delta(p_1, a) = q_0 \cup p_1 = (q_0, p_1)$$

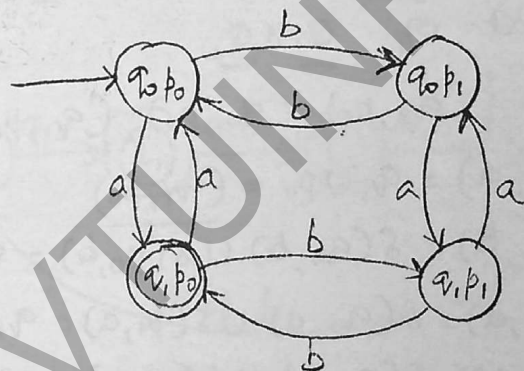
$$\delta((q_1, p_1), b) = \delta(q_1, b) \cup \delta(p_1, b) = q_1 \cup p_0 = (q_1, p_0)$$



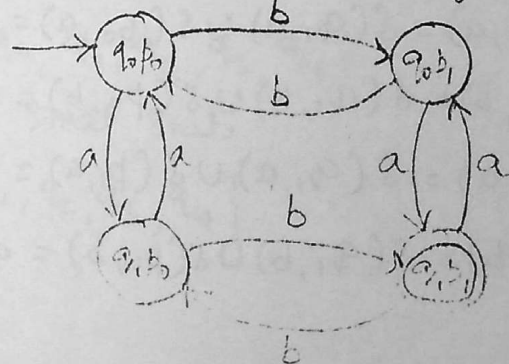
(i) Even no of a's & odd no. of b's



(ii) odd no. of a's & Even no. of b's

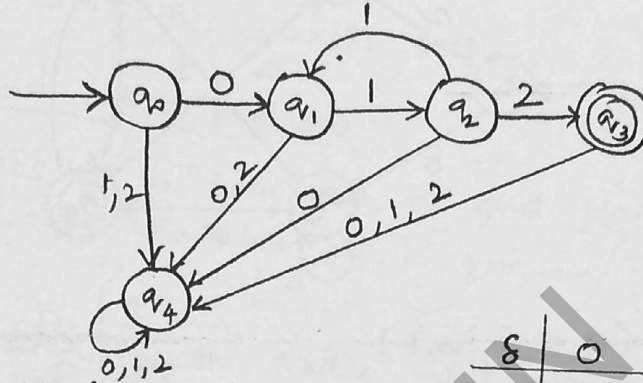


(iv) odd no. of a's & odd no. of b's



- (Eg) Obtain a DFSA to accept strings of 0's, 1's & 2's beginning with a '0' followed by odd no. of 1's & ending with a '2'.

Soln



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

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

$$\Sigma = \{0, 1, 2\} \quad \Sigma = \{0, 1, 2\}$$

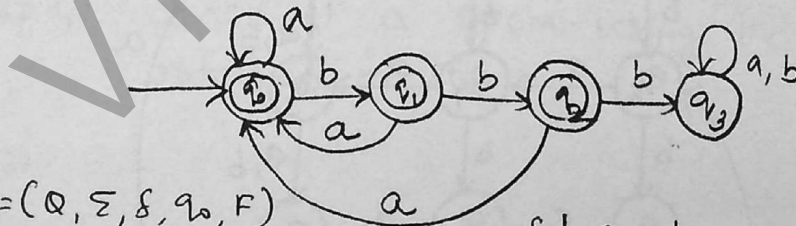
q_0 is start state

$$F = \{q_3\}$$

δ	0	1	2
$\rightarrow q_0$	q_1	q_4	q_4
q_1	q_4	q_2	q_4
q_2	q_4	q_1	q_3
$*q_3$	q_4	q_4	q_4
q_4	q_4	q_4	q_4

- (Eg) Obtain a DFSA to accept strings of a's & b's with at most two consecutive 'b's.

Soln



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

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

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

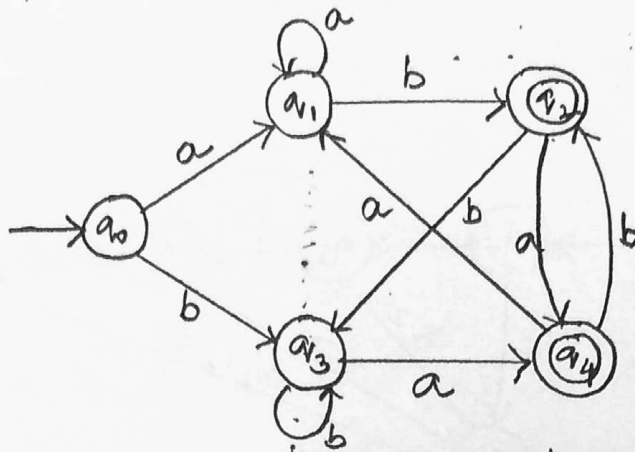
q_0 is start state

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

δ	a	b
$\rightarrow *q_0$	q_0	q_1
$*q_1$	q_0	q_2
$*q_2$	q_0	q_3
q_3	q_3	q_3

(Eg) obtain a DFSM to accept the string ending with ab or ba

Solⁿ



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

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

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

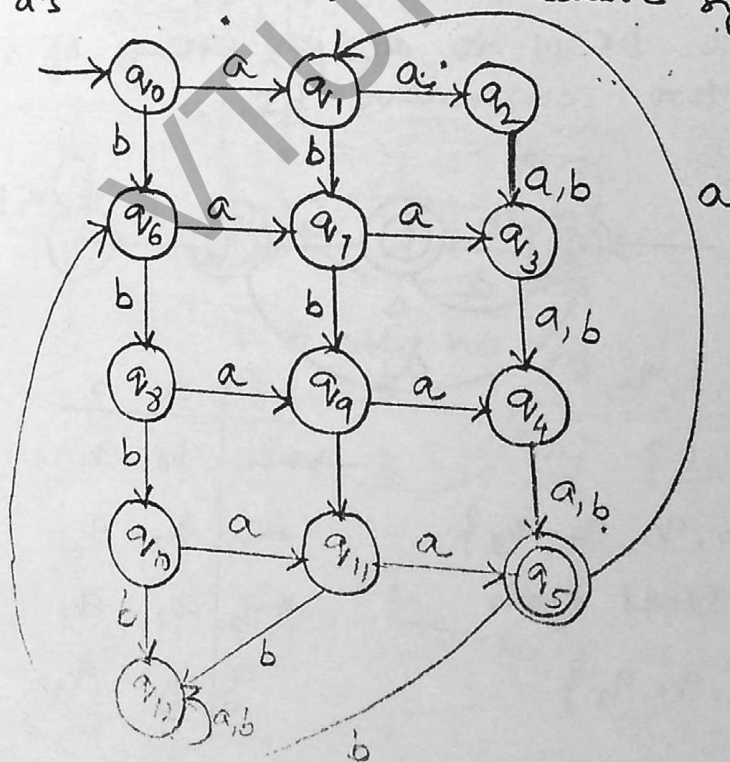
q_0 is start state

$$F = \{q_2, q_4\}$$

δ	a	b
$\rightarrow q_0$	q_1	q_3
q_1	q_1	q_2
$*q_2$	q_4	q_3
q_3	q_4	q_3
$*q_4$	q_1	q_2

(Eg) obtain a DFSM to accept strings of a's & b's such that each block of 5 consecutive symbols have atleast 2 a's

Solⁿ:



Three types of representing DFSM

- * Pattern recognition / pattern matching
- * Divisible by k problem
- * Modulo k counter problem.

Divisible by k problem

The steps to be followed to find finite state automata machine for divisible by k problem is given below.

Step 1: Identify the radix, input alphabet & the divisor k .

Step 2: compute the possible remainders, these remainders represent the states of DFSM

Step 3: find the transition

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

$$\text{where } j = (r \times i + d) \bmod k$$

Step 4: construct a DFSM using the transitions obtained in step 3.

- (Eq) Construct a DFSA which accepts strings of 0's & 1's where the value of each string is represented as a binary no. only. The strings representing 0 modulo 5 should be accepted. Ex: 0000, 0101, 1010, 1111 etc.

Soln

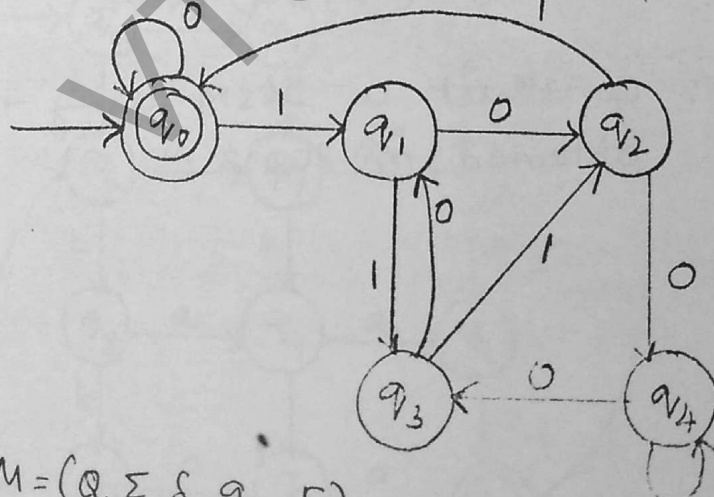
$$r = 2$$

$$d = 0, 1$$

$$k = 5$$

$$i = 0, 1, 2, 3, 4 \text{ (remainders of 5)}$$

i	d	$(r \times i + d) \bmod k = j$	$\delta(q_i, a) = q_j$
$i=0$	0	$(2 \times 0 + 0) \bmod 5 = 0$	$\delta(q_0, 0) = q_0$
	1	$(2 \times 0 + 1) \bmod 5 = 1$	$\delta(q_0, 1) = q_1$
$i=1$	0	$(2 \times 1 + 0) \bmod 5 = 2$	$\delta(q_1, 0) = q_2$
	1	$(2 \times 1 + 1) \bmod 5 = 3$	$\delta(q_1, 1) = q_3$
$i=2$	0	$(2 \times 2 + 0) \bmod 5 = 4$	$\delta(q_2, 0) = q_4$
	1	$(2 \times 2 + 1) \bmod 5 = 0$	$\delta(q_2, 1) = q_0$
$i=3$	0	$(2 \times 3 + 0) \bmod 5 = 1$	$\delta(q_3, 0) = q_1$
	1	$(2 \times 3 + 1) \bmod 5 = 2$	$\delta(q_3, 1) = q_2$
$i=4$	0	$(2 \times 4 + 0) \bmod 5 = 3$	$\delta(q_4, 0) = q_3$
	1	$(2 \times 4 + 1) \bmod 5 = 4$	$\delta(q_4, 1) = q_4$



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

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

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

$$q_0 \text{ is start state}$$

$$F = \{q_0\}$$

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

(Eg) Draw a DFSA to accept decimal strings divisible by 3.

Soln: $r = 10$ (decimal nos).

$d = 0, 1, 2, 3, 4, 5, 6, 7, 8, 9$

$k = 3$

$i = 0, 1, 2$

Note: For sake of convenience, let us put the digits from 0 to 9 based on the remainders, we get after dividing by 3 as given below.

* $\{0, 3, 6, 9\}$ with '0' as the remainder

so δ from $\{0, 3, 6, 9\} \Rightarrow \delta$ from $\{0\}$

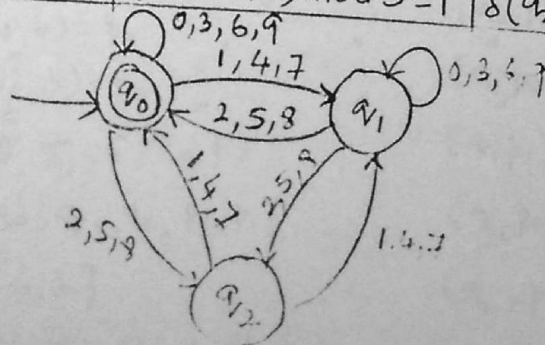
* $\{1, 4, 7\}$ with '1' as the remainder

so δ from $\{1, 4, 7\} \Rightarrow \delta$ from $\{1\}$

* $\{2, 5, 8\}$ with '2' as the remainder

so δ from $\{2, 5, 8\} \Rightarrow \delta$ from $\{2\}$

i	d	$j = (r \times i + d) \bmod k$	$\delta(q_i, a) = q_j$
$i = 0$	0	$(10 \times 0 + 0) \bmod 3 = 0$	$\delta(q_0, 0) = q_0 \Rightarrow \delta(q_0 \{0, 3, 6, 9\}, 0) = q_0$
	1	$(10 \times 0 + 1) \bmod 3 = 1$	$\delta(q_0, 1) = q_1 \Rightarrow \delta(q_0 \{1, 4, 7\}, 1) = q_1$
	2	$(10 \times 0 + 2) \bmod 3 = 2$	$\delta(q_0, 2) = q_2 \Rightarrow \delta(q_0 \{2, 5, 8\}, 2) = q_2$
$i = 1$	0	$(10 \times 1 + 0) \bmod 3 = 1$	$\delta(q_1, 0) = q_1 \Rightarrow \delta(q_1 \{0, 3, 6, 9\}, 0) = q_1$
	1	$(10 \times 1 + 1) \bmod 3 = 2$	$\delta(q_1, 1) = q_2 \Rightarrow \delta(q_1 \{1, 4, 7\}, 1) = q_2$
	2	$(10 \times 1 + 2) \bmod 3 = 0$	$\delta(q_1, 2) = q_0 \Rightarrow \delta(q_1 \{2, 5, 8\}, 2) = q_0$
$i = 2$	0	$(10 \times 2 + 0) \bmod 3 = 2$	$\delta(q_2, 0) = q_2 \Rightarrow \delta(q_2 \{0, 3, 6, 9\}, 0) = q_2$
	1	$(10 \times 2 + 1) \bmod 3 = 0$	$\delta(q_2, 1) = q_0 \Rightarrow \delta(q_2 \{1, 4, 7\}, 1) = q_0$
	2	$(10 \times 2 + 2) \bmod 3 = 1$	$\delta(q_2, 2) = q_1 \Rightarrow \delta(q_2 \{2, 5, 8\}, 2) = q_1$



Modulo k counter problem

(Eg) obtain a DFSM to accept strings of a's & b's such that no. of a's is divisible by 5 & no. of b's divisible by 3.

Solⁿ: $L = \{w \mid w \in (a+b)^*, \text{ such that } n_a(w) \bmod 5 = 0 \text{ \& } n_b(w) \bmod 3 = 0\}$

$n_a(w) \bmod 5$ gives the remainder 0, 1, 2, 3, 4 & it can be represented as

$$Q_1 = \{q_0, q_1, q_2, q_3, q_4\}$$

$n_b(w) \bmod 3$ gives the remainder 0, 1, 2 & it is represented as

$$Q_2 = \{p_0, p_1, p_2\}$$

Since each state of DFSM should keep track of $n_a(w) \bmod 5$ & $n_b(w) \bmod 3$ the possible states of the DFSM can be obtained by $Q_1 \times Q_2$ (cross product) & represented as given below:

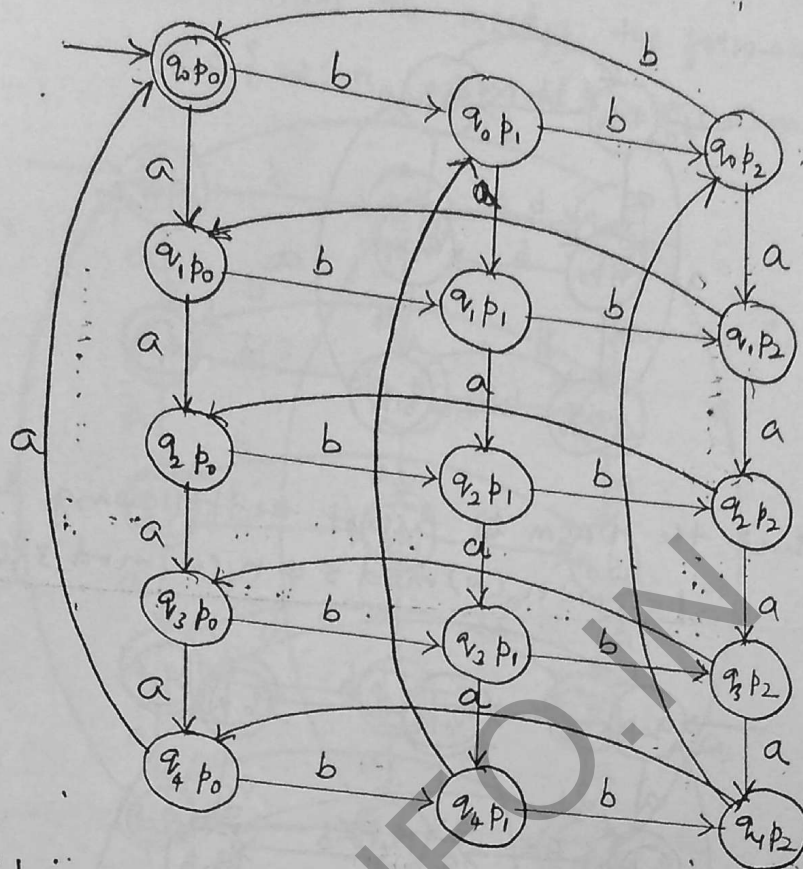
$$Q_1 \times Q_2 = \{(q_0, p_0), (q_0, p_1), (q_0, p_2), (q_1, p_0), (q_1, p_1), (q_1, p_2), (q_2, p_0), (q_2, p_1), (q_2, p_2), (q_3, p_0), (q_3, p_1), (q_3, p_2), (q_4, p_0), (q_4, p_1), (q_4, p_2)\}$$

where q_0 indicates that n_a in $w \bmod 5 = 0$

q_1	"	"	"	"	"	"	"	= 1
q_2	"	"	"	"	"	"	"	= 2
q_3	"	"	"	"	"	"	"	= 3
q_4	"	"	"	"	"	"	"	= 4

p_0 " the $n_b(w) \bmod 3 = 0$

p_1	"	"	"	"	"	"	"	= 1
p_2	"	"	"	"	"	"	"	= 2



Eg) Obtain a DFSM to accept strings of a's & b's such that $L = \{w / w \in (a+b)^* \text{ such that } n_a(w) \bmod 3 = 0 \text{ \& } n_b(w) \bmod 2 = 0\}$

Soln:

$$Q_1 = \{q_0, q_1, q_2\}$$

$$Q_2 = \{p_0, p_1\}$$

$$\therefore Q_1 \times Q_2 = \{(q_0, p_0), (q_0, p_1), (q_1, p_0), (q_1, p_1), (q_2, p_0), (q_2, p_1)\}$$

$$\delta(q_0, a) = q_1$$

$$\delta(q_1, a) = q_2$$

$$\delta(q_2, a) = q_0$$

$$\delta(p_0, b) = p_1$$

$$\delta(p_1, b) = p_0$$

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

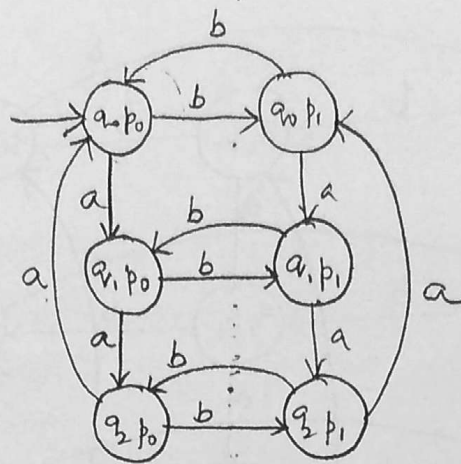
$$Q = \{q_0, q_1, q_2, p_0, p_1\}$$

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

(q_0, p_2) is start state

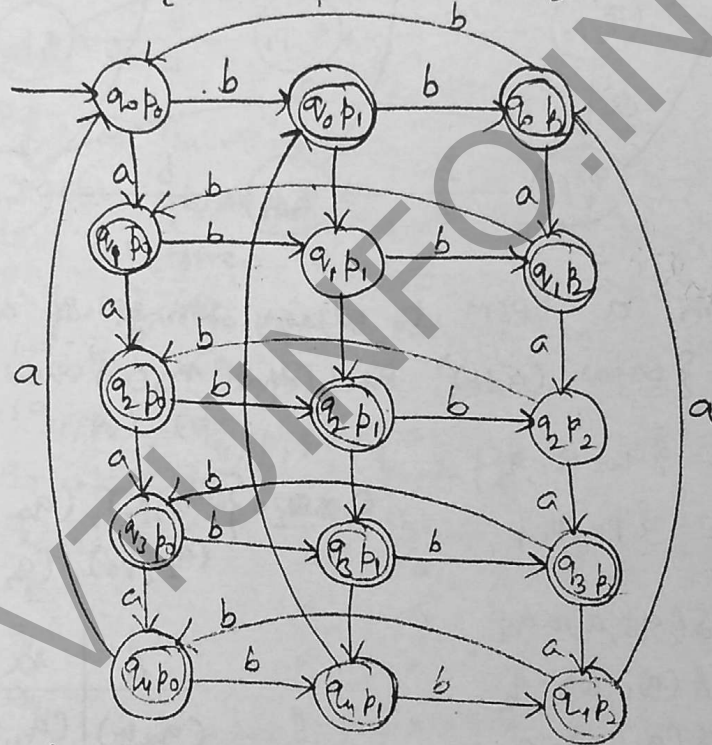
$F = (q_0, p_0)$ is accepting state

δ	a	b
(q_0, p_0)	(q_1, p_0)	(q_0, p_1)
(q_0, p_1)	(q_1, p_1)	(q_0, p_0)
(q_1, p_0)	(q_2, p_0)	(q_1, p_1)
(q_1, p_1)	(q_2, p_1)	(q_1, p_0)
(q_2, p_0)	(q_0, p_0)	(q_2, p_1)
(q_2, p_1)	(q_0, p_1)	(q_2, p_0)



(Eg) Construct the DFSM to accept the following language
 $L = \{w : n_a(w) \bmod 5 \neq n_b(w) \bmod 3\}$

Soln:



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

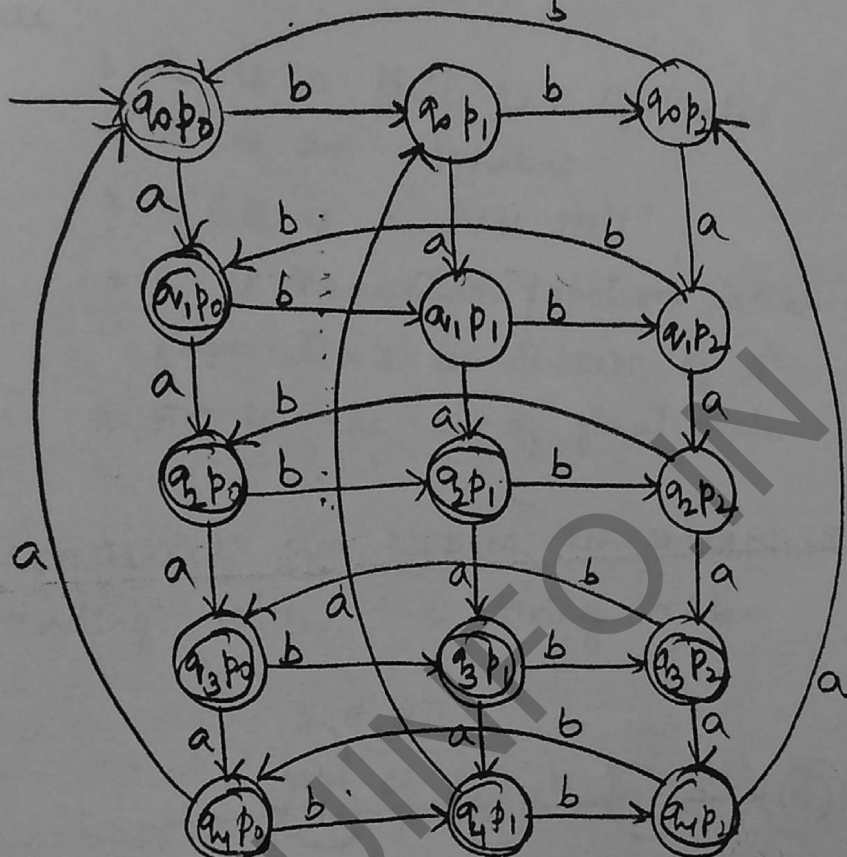
$$Q = \{(q_0, p_0), (q_0, p_1), (q_0, p_2), (q_1, p_0), (q_1, p_1), (q_1, p_2), (q_2, p_0), (q_2, p_1), (q_2, p_2), (q_3, p_0), (q_3, p_1), (q_3, p_2), (q_4, p_0), (q_4, p_1), (q_4, p_2)\}$$

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

(q_0, p_0) is start state

$$F = \{(q_0, p_1), (q_0, p_2), (q_1, p_0), (q_1, p_2), (q_2, p_0), (q_2, p_1), (q_3, p_0), (q_3, p_1), (q_4, p_0), (q_4, p_1), (q_4, p_2)\}$$

(Eg) Construct the DFSA to accept the following language
 $L = \{ w : n_a(w) \bmod 5 > n_b(w) \bmod 3 \}$



5.4 Non Deterministic Finite State Machine (NFSM) (NDFSM) (16)

A non deterministic FSM (or NDFSM) M is a quintuple

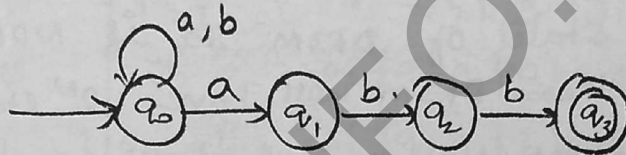
$$M = (Q, \Sigma, \delta, q_0, F)$$

where:

- * Q is a finite set of states
- * Σ is an alphabet
- * $q_0 \in Q$ is a start state
- * δ is transition function which is mapping from $Q \times \Sigma$ to subsets of Q .
- * $F \subseteq Q$ is the set of final state.

(Eg) Construct an NDFSM to accept strings of a's & b's ending with the string abb.

Soln



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

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

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

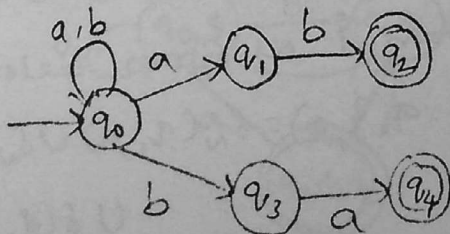
q_0 is start state

$$F = \{q_3\}$$

δ	a	b
$\rightarrow q_0$	$\{q_0, q_1\}$	q_0
q_1	$\emptyset$	q_2
q_2	$\emptyset$	q_3
$*q_3$	$\emptyset$	$\emptyset$

(Eg) Construct an NDFSM to accept the strings of a's & b's ending with string ab or ba.

Soln



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

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

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

$$F = \{q_2, q_4\}$$

δ	a	b
$\rightarrow q_0$	$\{q_0, q_1\}$	$\{q_0, q_3\}$
q_1	$\emptyset$	q_2
$*q_2$	$\emptyset$	$\emptyset$
q_3	q_4	$\emptyset$
$*q_4$	$\emptyset$	$\emptyset$

Conversion from NDFSM to DFSM

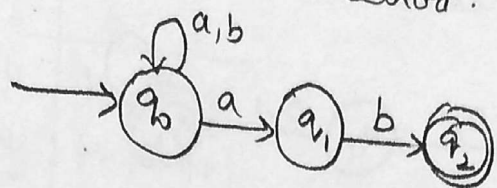
Two methods

- 1) Subset construction method
- 2) Lazy evaluation method.

Subset construction method

- Step 1: The start state of NDFSM is the start state of DFSM.
- Step 2: The input alphabets of NDFSM are the input alphabets of DFSM.
- Step 3: The set of subsets of NDFSM will be the states of DFSM. So if NDFSM has N states then DFSM will have 2^N states.
- Step 4: Identify the final states of DFSM.
If $\{q_i, q_j, \dots, q_k\}$ is a state in DFSM. Then $\{q_i, q_j, \dots, q_k\}$ will be final state of DFSM provided one of $q_i, q_j, \dots, q_k$ is the final state of NDFSM.
- Step 5: Identify the transition of DFSM for each state $\{q_i, q_j, \dots, q_k\}$ in DFSM for each input alphabet in Σ . The transition can be obtained as given below:
- $$\delta_D(\{q_i, q_j, \dots, q_k\}, a) = \delta_N(q_i, a) \cup \delta_N(q_j, a) \cup \dots \cup \delta_N(q_k, a).$$
- Thus DFSM can be obtained using subset construction method.

(Eg) Obtain DFsm for the following NDFSM using subset construction method. (17)



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

Sol:

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

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

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

q_0 is start state

δ	a	b
$\rightarrow \{q_0\}$	$\{q_0, q_1\}$	$\{q_0\}$
$\{q_1\}$	$\emptyset$	$\{q_2\}$
$\ast \{q_2\}$	$\emptyset$	$\emptyset$
$\{q_0, q_1\}$	$\{q_0, q_1\}$	$\{q_0, q_2\}$
$\ast \{q_0, q_2\}$	$\{q_0, q_1\}$	$\{q_0\}$
$\ast \{q_1, q_2\}$	$\emptyset$	$\{q_2\}$
$\ast \{q_0, q_1, q_2\}$	$\{q_0, q_1\}$	$\{q_0, q_2\}$

$\delta(\{q_0, q_1\}, a) = \{q_0, q_1\} \cup \emptyset = \{q_0, q_1\}$

$\delta(\{q_0, q_1\}, b) = \{q_0\} \cup \{q_2\} = \{q_0, q_2\}$

$\delta(\{q_0, q_2\}, a) = \{q_0, q_1\} \cup \emptyset = \{q_0, q_1\}$

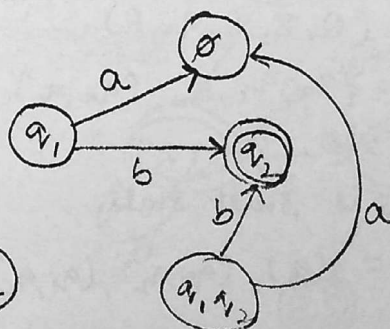
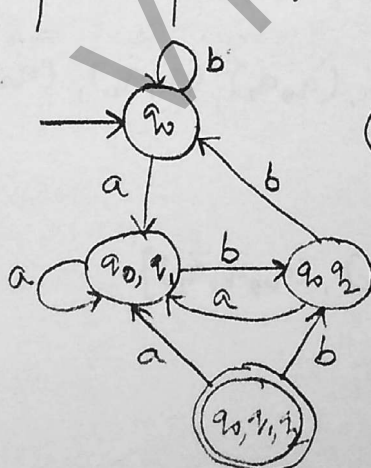
$\delta(\{q_0, q_2\}, b) = \{q_0\} \cup \emptyset = \{q_0\}$

$\delta(\{q_1, q_2\}, a) = \emptyset \cup \emptyset = \emptyset$

$\delta(\{q_1, q_2\}, b) = \emptyset \cup \emptyset = \emptyset$

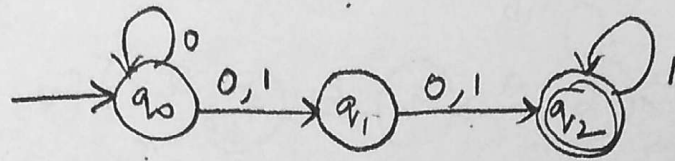
$\delta(\{q_0, q_1, q_2\}, a) = \{q_0, q_1\} \cup \emptyset \cup \emptyset = \{q_0, q_1\}$

$\delta(\{q_0, q_1, q_2\}, b) = \{q_0\} \cup \{q_2\} \cup \emptyset = \{q_0, q_2\}$



Note: From start state, if it is not possible to reach to any of the state whose state can be eliminated.

eg) obtain DFSCM for the following NDFSM using subset construction method.

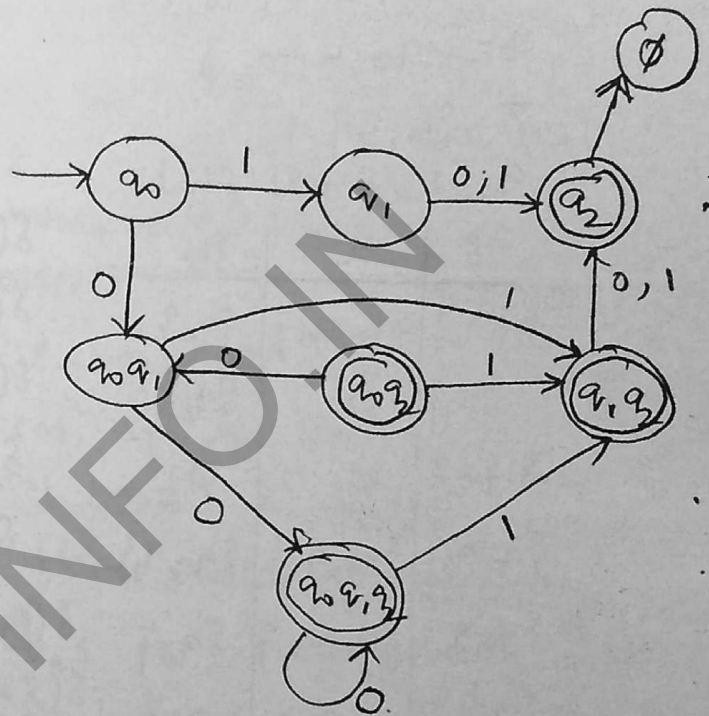


Soln

q_0 is start state

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

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



DFSCM

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

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

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

q_0 is start state

$F = \{(q_2), (q_0, q_2), (q_1, q_2), (q_0, q_1, q_2)\}$

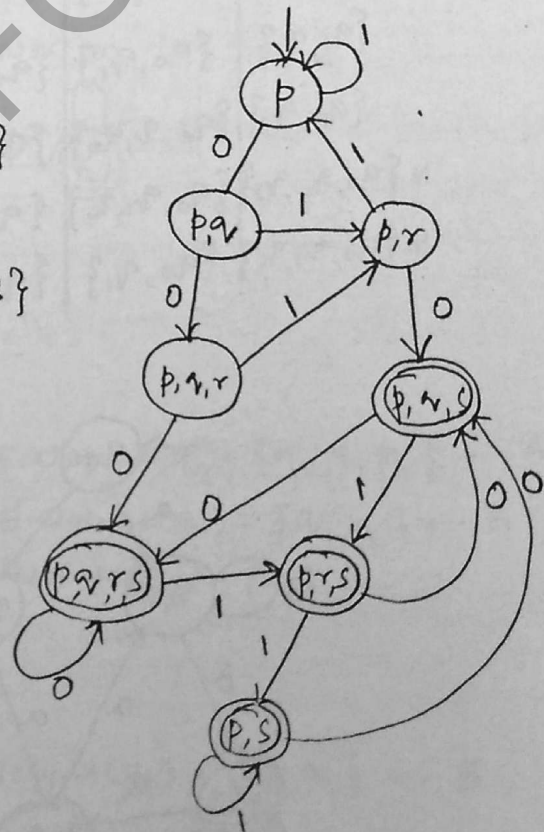
Q. Convert to a DFSA the following NDFSA using lazy evaluation method.

(18)

δ	0	1
$\rightarrow p$	$\{p, q\}$	$\{p\}$
q	$\{r\}$	$\{r\}$
r	$\{s\}$	$\{\emptyset\}$
$*s$	$\{s\}$	$\{s\}$

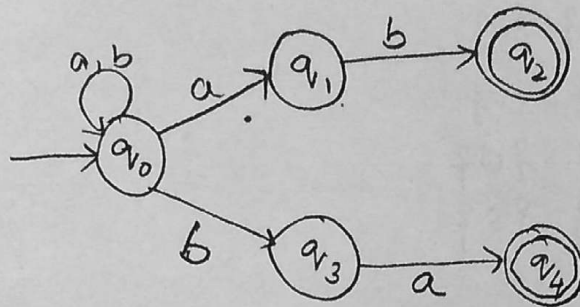
Soln

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

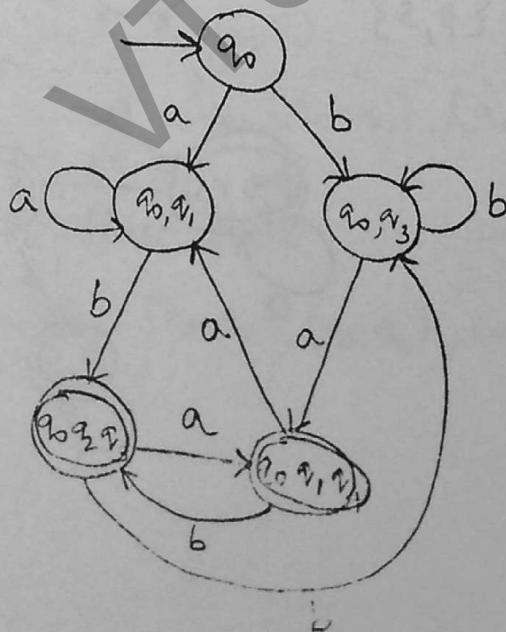


Q7) Obtain an NDFSM to accept strings of a's & b's ending with ab or ba from this obtain an equivalent DFSA using lazy evaluation method.

Soln



δ	a	b
$\rightarrow q_0$	$\{q_0, q_1\}$	$\{q_0, q_3\}$
$\{q_0, q_1\}$	$\{q_0, q_1\}$	$\{q_0, q_2, q_3\}$
$\{q_0, q_3\}$	$\{q_0, q_1, q_4\}$	$\{q_0, q_3\}$
$\times \{q_0, q_2, q_3\}$	$\{q_0, q_1, q_4\}$	$\{q_0, q_3\}$
$\times \{q_0, q_1, q_4\}$	$\{q_0, q_1\}$	$\{q_0, q_2, q_3\}$



(19)

Non Deterministic finite State Machine with Epsilon transition (ϵ -NDFSM)

The Non Deterministic finite state machine is 5 tuple or quin tuple indicating 5 components

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

where :

Q is finite set of states

Σ is the input alphabet.

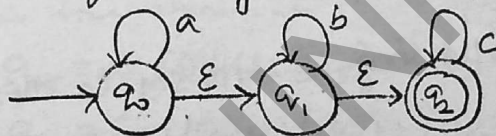
$q_0 \in Q$ is the start state

$F \subseteq Q$ is the set of accepting states

δ is transition function. It maps from

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

(Eq) Convert the following ϵ -NDFSM to DFSM



Solⁿ: Start State

$$\epsilon\text{-closure}(q_0) = \{q_0, q_1, q_2\} \text{ --- A}$$

consider the State A

$$\delta(A, a) = q_0 \cup \emptyset \cup \emptyset = q_0 = \epsilon\text{-closure}(q_0) = \{q_0, q_1, q_2\} \text{ --- A}$$

$$\delta(A, b) = \emptyset \cup q_1 \cup \emptyset = q_1 = \epsilon\text{-closure}(q_1) = \{q_1, q_2\} \text{ --- B}$$

$$\delta(A, c) = \emptyset \cup \emptyset \cup q_2 = q_2 = \epsilon\text{-closure}(q_2) = \{q_2\} \text{ --- C}$$

consider the State B

$$\delta(B, a) = \emptyset \cup \emptyset = \emptyset$$

$$\delta(B, b) = q_1 \cup \emptyset = q_1 = \epsilon\text{-closure}(q_1) = \{q_1, q_2\} \text{ --- B}$$

$$\delta(B, c) = \emptyset \cup q_2 = q_2 = \epsilon\text{-closure}(q_2) = \{q_2\} \text{ --- C}$$

Finite Automata with Final

consider the state C

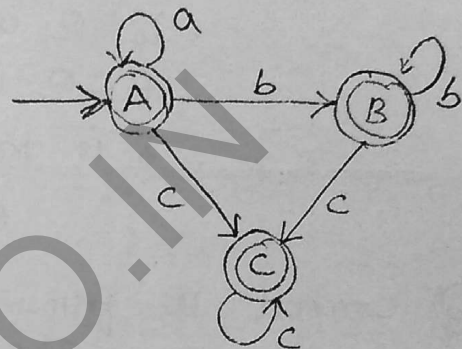
$$\delta(C, a) = \emptyset$$

$$\delta(C, b) = \emptyset$$

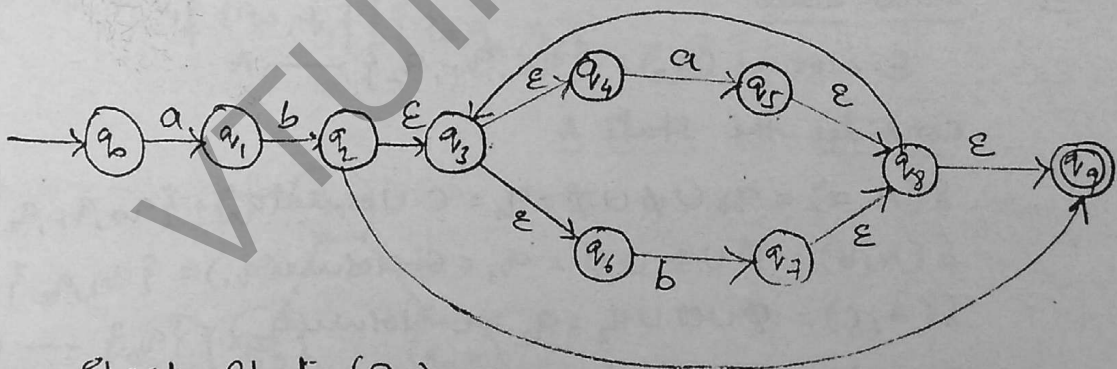
$$\delta(C, c) = q_2 = E\text{-closure}(q_2) = q_2 \text{ --- } C$$

NO more new states stops, construct transition table: δ DFSM

δ	a	b	c
$\rightarrow A$	A	B	C
$\times B$	$\emptyset$	B	C
$\times C$	$\emptyset$	$\emptyset$	C



(Eg) Convert the following ϵ -NDFSM to DFSM



Solⁿ

Start state (q_0)

$$E\text{-closure}(q_0) = q_0 \text{ --- } A$$

consider the state A

$$\delta(A, a) = q_1 = E\text{-closure}(q_1) = q_1 \text{ --- } B$$

$$\delta(A, b) = \emptyset$$

(20)

Consider the state B:

$$\delta(B, a) = \emptyset$$

$$\delta(B, b) = q_2 = E\text{-closure}(q_2) = \{q_2, q_3, q_4, q_6, q_9\} \text{ --- C}$$

Consider the state C:

$$\delta(C, a) = q_5 = E\text{-closure}(q_5) = \{q_5, q_3, q_4, q_6, q_9\} \text{ ---}$$

$$\delta(C, b) = q_7 = E\text{-closure}(q_7) = \{q_7, q_3, q_4, q_6, q_9\} \text{ --- E}$$

Consider the state D

$$\delta(D, a) = q_5 = E\text{-closure}(q_5) = \{q_3, q_4, q_5, q_6, q_8, q_9\} \text{ --- D}$$

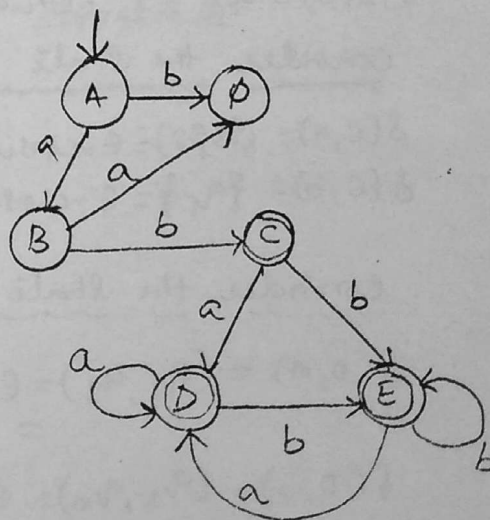
$$\delta(D, b) = q_7 = E\text{-closure}(q_7) = \{q_3, q_4, q_6, q_7, q_8, q_9\} \text{ --- E}$$

Consider the state E

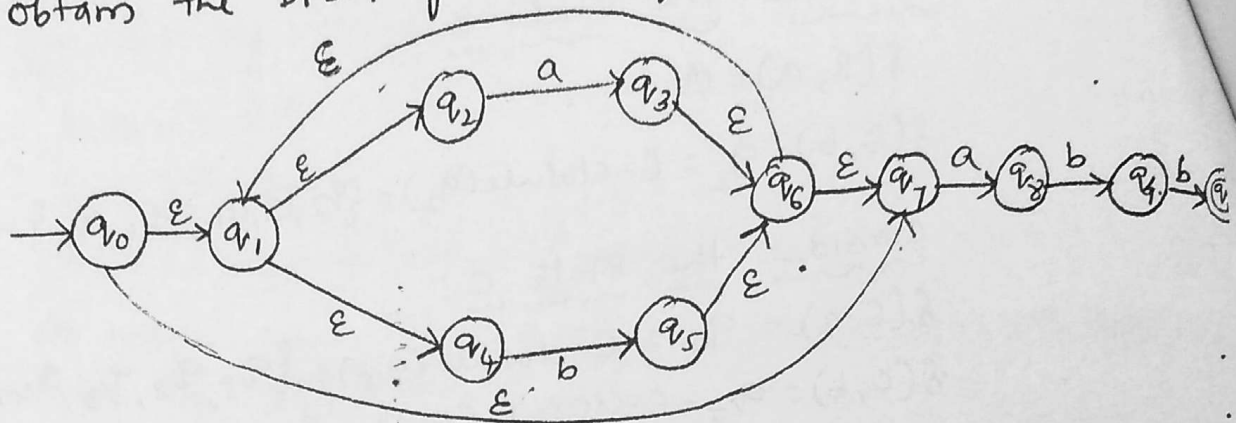
$$\delta(E, a) = q_5 = E\text{-closure}(q_5) = \{q_3, q_4, q_5, q_6, q_8, q_9\} \text{ --- D}$$

$$\delta(E, b) = q_7 = E\text{-closure}(q_7) = \{q_3, q_4, q_6, q_7, q_8, q_9\} \text{ --- E}$$

δ	a	b
$\rightarrow A$	B	$\emptyset$
B	$\emptyset$	C
* C	D	E
* D	D	E
* E	D	E



(eg) obtain the DFSM from the following ϵ -NDFSM



Soln Start State (q_0)

$$\epsilon\text{-closure}(q_0) = \{q_0, q_1, q_2, q_4, q_7\} \text{ --- A}$$

consider the state A

$$\delta(A, a) = \{q_3, q_8\} = \epsilon\text{-closure}(q_3, q_8) = \{q_1, q_2, q_3, q_4, q_6, q_7, q_8\} \text{ --- B}$$

$$\delta(A, b) = q_5 = \epsilon\text{-closure}(q_5) = \{q_5, q_6, q_7, q_1, q_2, q_4\} \text{ --- C}$$

consider the state B

$$\delta(B, a) = \{q_3, q_8\} = \epsilon\text{-closure}(q_3, q_8) = \{q_1, q_2, q_3, q_4, q_6\} \text{ --- B}$$

$$\delta(B, b) = q_5 = \epsilon\text{-closure}(q_5) = \{q_5, q_6, q_7, q_1, q_2, q_4, q_9\} \text{ --- D}$$

consider the state C

$$\delta(C, a) = \{q_3, q_8\} = \epsilon\text{-closure}(q_3, q_8) = \{q_1, q_2, q_3, q_4, q_6, q_7, q_8\} \text{ --- B}$$

$$\delta(C, b) = \{q_5\} = \epsilon\text{-closure}(q_5) = \{q_1, q_2, q_4, q_5, q_6, q_7\} \text{ --- C}$$

consider the state D

$$\delta(D, a) = (q_3, q_8) = \epsilon\text{-closure}(q_3, q_8) = \{q_1, q_2, q_3, q_4, q_6, q_7, q_8\} \text{ --- B}$$

$$\delta(D, b) = (q_5, q_{10}) = \epsilon\text{-closure}(q_5, q_{10}) = \{q_1, q_2, q_4, q_5, q_6, q_7, q_{10}\} \text{ --- E}$$

(21)

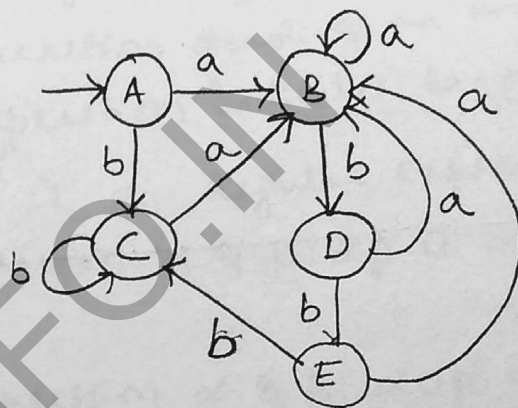
Consider the state E:

$$\delta(E, a) = \{q_3, q_8\} = E\text{-closure}(q_3, q_8)$$

$$= \{q_1, q_2, q_3, q_4, q_7, q_8\} \text{ --- B}$$

$$\delta(E, b) = \{q_5\} = E\text{-closure}(q_5) = \{q_1, q_2, q_4, q_5, q_6, q_7\} \text{ --- C}$$

δ	a	b
A	B	C
B	B	D
C	B	C
D	B	E
E	B	C



machine based hierarchy of language
classes not attached.

Regular Expression

Definition of a Regular Expression

A regular expression is a string that can be formed according to the following rules:

- * $\emptyset$ is a regular expression denoting an empty language
- * ϵ is a regular expression indicating language containing empty string.
- * Every element in Σ is a regular expression
- * Given two regular expression α & β , $\alpha\beta$ is a regular expression.
- * Given two regular expression α & β , $\alpha \cup \beta$ is a regular expression.
- * Given a regular expression α , α^+ is a regular expression.
- * Given a regular expression α , α^* is a regular expression.
- * Given a regular expression α , (α) is a regular expression.

So, if we let $\Sigma = \{a, b\}$, the following strings are regular expressions:

$$\emptyset, \epsilon, a, b, (a \cup b)^*, abba \cup \epsilon.$$

The expression obtained by applying any of rules above are regular expression.

Prepared by: Dr. Vrinda Shetty

R.E

Meaning.

a^* → String consisting of any no of a's
(Zero or more a's)

$L(a^*) = (L(a))^*$ → $L(a^*)$ is a language that is formed
by concatenating together zero or more
strings drawn from $L(a)$.

a^+ → String consisting of one or more a's

$L(a^+) = L(a a^*)$ → $L(a^+)$ is the language that is formed
by concatenating together one or more
strings drawn from $L(a)$.

$(a+b)$ → String consisting of either a or b.

$(a+b)^*$ → String consisting of any number a's
& b's including null string.

$(a+b)^*abb$ → String consisting of any number
of a's & b's ending with string abb.

$ab(a+b)^*$ → String starting with ab followed
by any no. of a's & b's.

$(a+b)^*aa(a+b)^*$ → String consisting of any no. of a's
& b's having a substring aa.

$a^*b^*c^*$ → String consisting of any no of a's
followed by any no of b's followed
by any no of c's.

$(a+b)^*(a+bb)$ → String consisting of any no of
a's & b's ending with a or bb.

$(aa)^*(bb)^*b$ → String consisting of even no of
a's followed by odd no of b's.

(Eq) obtain a regular expression (R.E) representing strings of a's & b's having length 2.

Sol:

$$R.E = aa + ab + ba + bb$$

or

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

(Eq) obtain a R.E representing strings of a's & b's having odd length.

Sol:

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

or

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

or

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

(Eq) obtain a R.E to accept strings of a's & b's of length 52

Sol:

$$E + a + b + aa + ab + ba + bb$$

or

$$(E + a + b)(E + a + b)$$

or

$$(E + a + b)^2$$

(Eq) obtain a R.E to accept strings of a's & b's whose tenth symbol from the right end is a.

Sol:

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

(Eg) Obtain a R.E to accept strings of a's & b's for the following:

(i) starting with a & ending with b

(ii) with atleast three consecutive a's.

Sol: (i) $a(a+b)^*b$

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

(Eg) Obtain a R.E to accept the ~~words~~ strings ^{of a's & b's} with 2 (or) more letters but beginning & ending with the same letter.

Sol:

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

(Eg) Obtain a R.E to accept strings of a's & b's whose length is either even or multiples of 3 or both.

Sol:

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

(Eg) Obtain a R.E to accept strings of a's & b's such that every block of 4 consecutive symbols contains least 2 a's.

Sol:

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

Ex) obtain a R.E for the language

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

(ii) $L = \{a^n b^m \mid m \geq 1, n \geq 1, nm \geq 3\}$

(iii) $L = \{a^{2n} b^{2m} \mid n \geq 0, m \geq 0\}$

Sol?

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

(ii)

Case (i): Since $nm \geq 3$

if $m=1$ then $n \geq 3$

R.E = $aaaa^*b$

Case (ii): Since $nm \geq 3$

if $n=1$ then $m \geq 3$

R.E = $bbbb^*a$

Case (iii): Since $nm \geq 3$

if $m \geq 2$ then $n \geq 2$

R.E = aaa^*bbb^*

R.E = $aaaa^*b + bbbb^*a + aaa^*bbb^*$

(iii) $(aa)^*(bb)^*$

Ex) obtain the R.E for strings of a's & b's containing not more than 3 a's.

Sol? $b^*(\epsilon + a)b^*(\epsilon + a)b^*(\epsilon + a)b^*$

Ex) obtain a R.E for the language $L = \{a^n b^m \mid n \geq 4, m \leq 3\}$

Sol?

$aaaaa^*(\epsilon + b)(\epsilon + b)(\epsilon + b)$

(Eg) obtain the R.E for the $L = \{w : n_a(w) \bmod 3 = 0 \text{ where } w \in (a,b)^*\}$

Sol: $(b^* ab^* ab^* ab^*)^*$

(Eg) obtain the R.E for the set of all strings that do not end with 01 over $\{0,1\}^*$.

Sol: $(0+1)^* (10+00+11)$

(Eg) obtain a R.E for $L = \{w : \text{strings ends with } ab \text{ or } ba \text{ where } w \in (a,b)^*\}$

Sol: $(a+b)^* (ab+ba)$
or
 $(a+b)^* ab + (a+b)^* ba$

Analyzing a simple Regular expression

$$\begin{aligned} \text{(i)} \quad L((a \cup b)^* b) &= L((a \cup b)^*) L(b) \\ &= (L((a \cup b)))^* L(b) \\ &= (L(a) \cup L(b))^* L(b) \\ &= (\{a\} \cup \{b\})^* \{b\} \\ &= \{a,b\}^* \{b\} \end{aligned}$$

$$\begin{aligned} \text{(ii)} \quad L((a \cup b)(a \cup b)^* a(a \cup b)^*) &= L((a \cup b)(a \cup b)^*) L(a) L((a \cup b)^*) \\ &= L((a \cup b)(a \cup b)^*) \{a\} (L((a \cup b)))^* \\ &= L((a \cup b)) L((a \cup b)^*) \{a\} \{a,b\}^* \\ &= \{a,b\} \{a,b\} \{a\} \{a,b\}^* \end{aligned}$$

Kleene's Theorem

Theorem: Any language that can be defined with a regular expression can be accepted by some FSM & so is regular.

(or)

Let 'R' be a regular expression then there exists finite automata $M = (Q, \Sigma, \delta, q_0, F)$ which accepts $L(R)$

(or):

prove that there exists a finite automata to accept the language $L(R)$ corresponding to the regular expression R.

Proof: By definition ϕ , ϵ , a are regular expressions, so that corresponding machines to recognize the language for the respective expressions are shown in figure below:

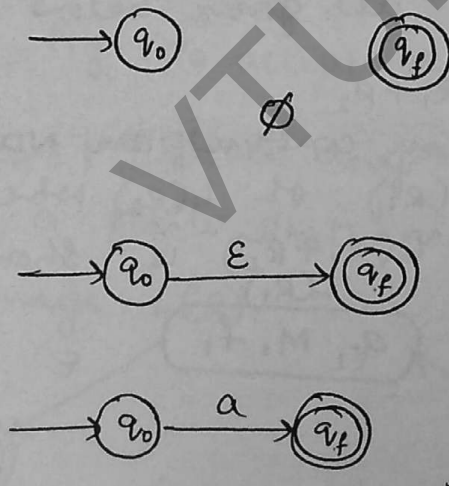


Fig: NDFS'm to accept ϕ, ϵ, a .

We must show how to build FSMs to accept language that are defined by regular expressions that exploit the operations of concatenation, union and Kleene star.

The schematic representation of R.E are to accept the language $L(R)$ is shown in the figure where 'q' is the start state & 'f' is the final state of machine M.

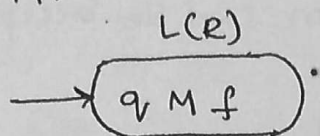


Fig: Schematic representation of FSM accepting $L(R)$.

In the definition of a R.E, it is clear that if R & S are R.E then $R+S$, R^* , $R.S$ are R.E which clearly uses 3 operators $+$, $*$, $.$

Let $M_1 = (Q_1, \Sigma_1, \delta_1, q_1, F_1)$ be a machine which accepts the language $L(R_1)$ corresponding to the R.E R_1 .
Let $M_2 = (Q_2, \Sigma_2, \delta_2, q_2, F_2)$ be a machine which accepts the language $L(R_2)$ corresponding to the R.E R_2 .

The various machines corresponding to the R.E R_1+R_2 , $R_1.R_2$, R^* are shown below.

Case (i) :- $R = R_1 + R_2$

We can construct an NDFSM which accepts either $L(R_1)$ or $L(R_2)$ which can be represented as $L(R_1+R_2)$ is shown in the figure below:

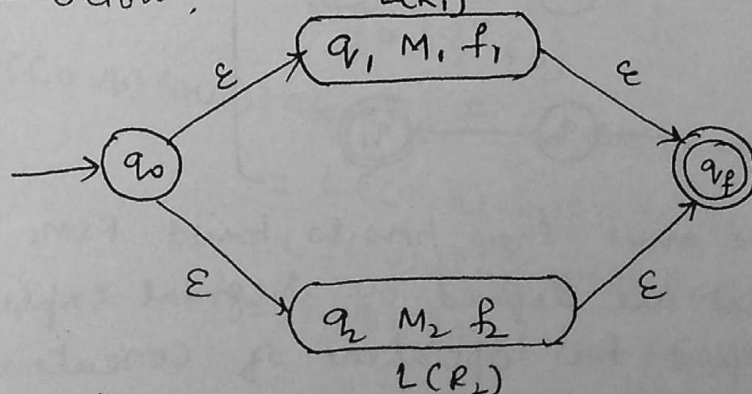


Fig: To accept the language $L(R_1+R_2)$

Case (ii) : $R = R_1 \cdot R_2$

We can construct an NDFSM which accepts $L(R_1)$ followed by $L(R_2)$ which can be represented as $L(R_1 \cdot R_2)$ as shown in figure below:

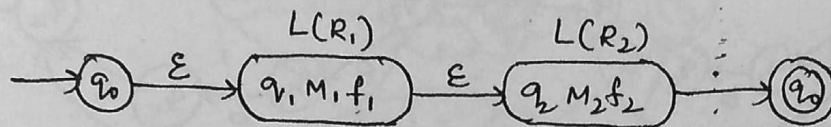


Fig: To accept the language $L(R_1 \cdot R_2)$

Case (iii) : $R_1 = (R_1)^*$

We can construct an NDFSM which accepts the language $L(R_1)^*$ as shown in figure below:

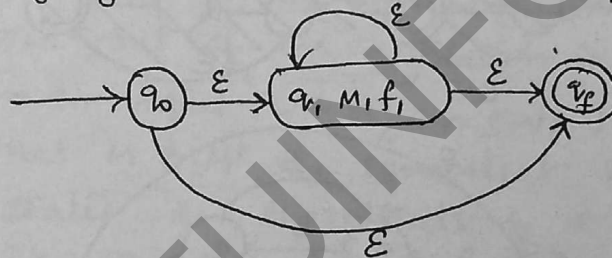


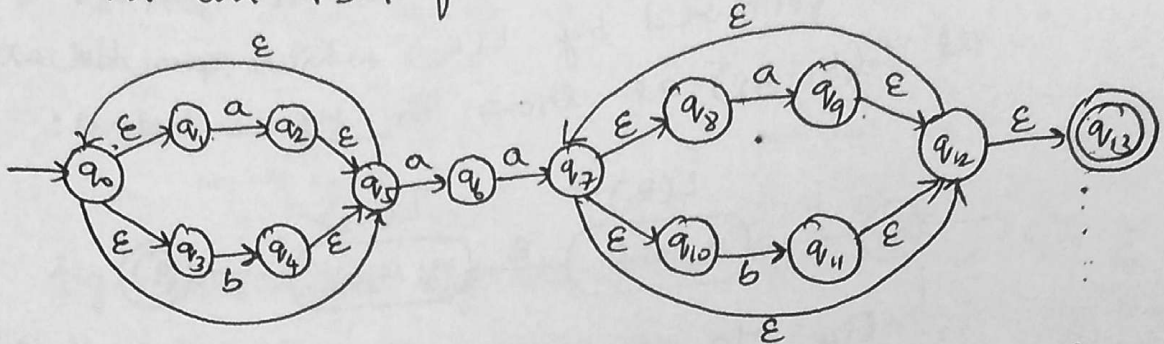
Fig: To accept the language $L(R)^*$

Hence for given any R.E it is proved that there exists a finite state machine to accept the language $L(R)$.

Building an FSM from a Regular Expression

Eg.
soln.

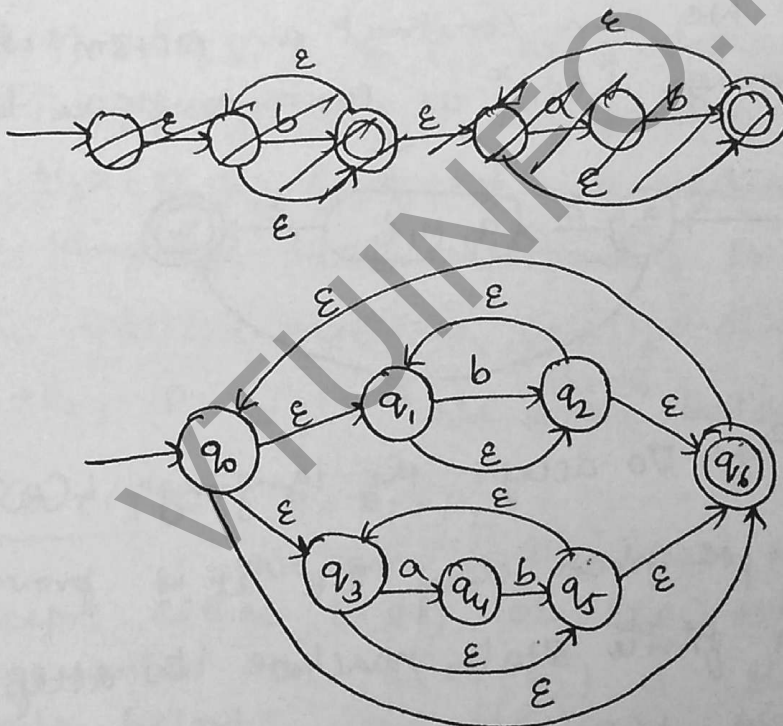
Obtain an FSM for R.E $(a+b)^* aa(a+b)^*$



Eg.

Obtain an FSM for $(b+ab)^*$ or $(bUab)^*$

soln.

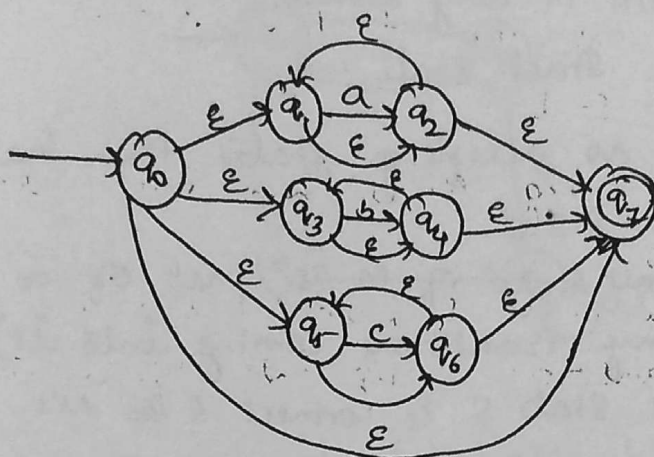


(6)

(Eq)

Soln

obtain an FSM for regular expression $a^* + b^* + c^*$

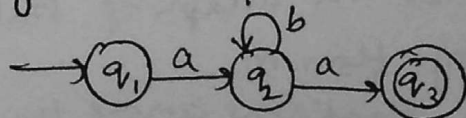


Building a Regular expression from an FSM

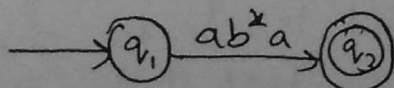
We can build a regular expression from an FSM. The goal of the algorithm is to construct, from an input FSM M , an output machine M' such that M & M' are equivalent & M' has only two states, a start state & a single accepting state. It will also have just one transition, which will go from its start state to its accepting state. The label on that transition will be a R.E that describes all the strings that could have driven the original machine M from its start state to some accepting state.

(Eq)

Building an equivalent Machine M



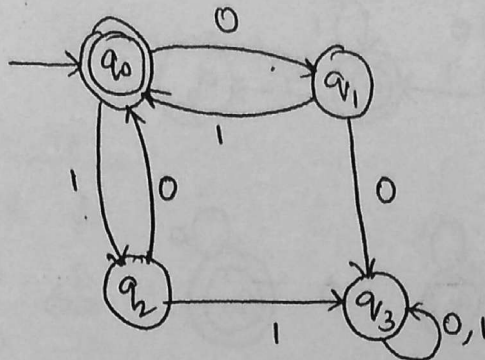
Soln



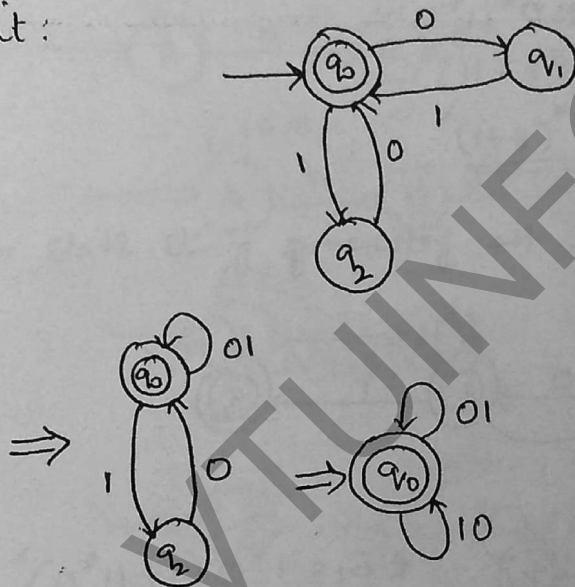
from to regex heuristic ($M: \text{FSM}$) =

- 1) Remove from M any states that are unreachable from the start state.
- 2) If M has no accepting states then halt & return the simple R.E $\emptyset$.
- 3) If the start state of M is part of a loop (i.e. it has any transitions coming into it). Create a new start state S & connect S to M 's start state via an ϵ -transition. This new start state S will have no transitions into it.
- 4) If there is more than one accepting state of M or if there is just one but there are any transitions out of it, create a new accepting state & connect each of M 's accepting states to it via an ϵ -transition. Remove the old accepting states from the set of accepting states. Note that the new accepting state will have no transitions out from it.
- 5) If, at this point, M has only one state, then that state is both the start state & the accepting state & M has no transitions. So $L(M) = \{\epsilon\}$. Halt & return the simple R.E ϵ .
6. Until only the start state & the accepting state remain do
 - 6.1: Select some state q of M . Any state except the start state or the accepting state may be chosen.
 - 6.2: Remove q from M .
 - 6.3: Modify the transitions among the remaining states so that M accepts the same strings. The labels on the rewritten transitions may be any R.E.
7. Return the R.E that labels the one remaining transition from the start state to the accepting state.

Q. Obtain the R.E for the following finite state machine



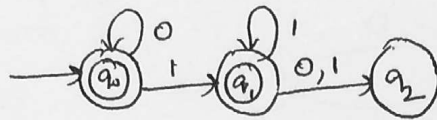
Soln: q_3 state is not reachable from ^{start} state & eliminate it:



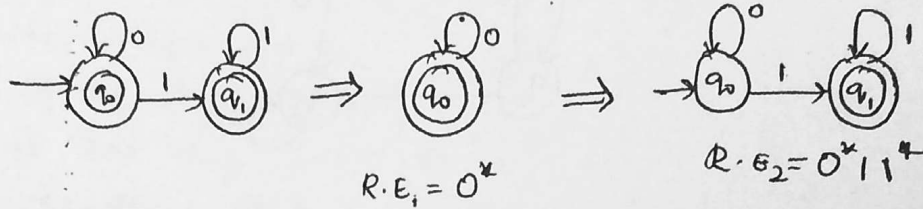
It is clear from the transition diagram that the machine accepts.

$$R.E = (01 + 10)^*$$

Ex) obtain a R.E for the following finite state machine.



Sol?



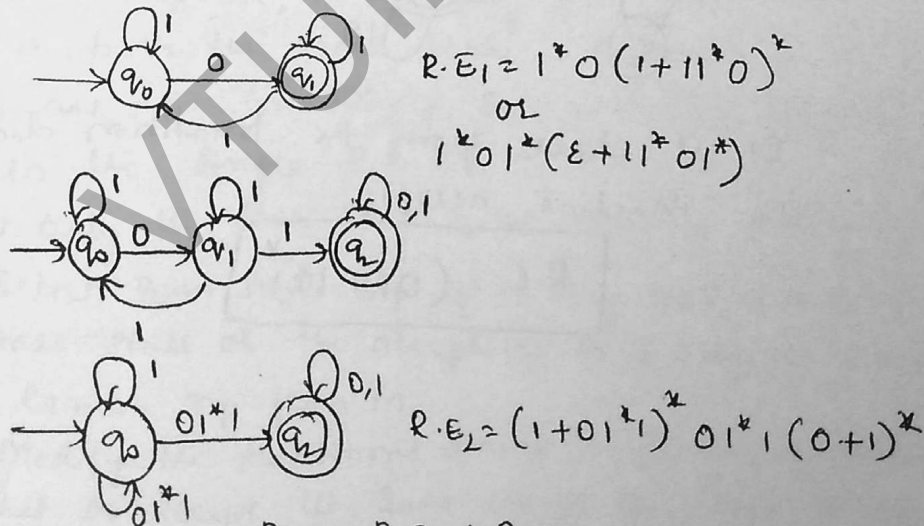
$$\begin{aligned} R.E &= R.E_1 + R.E_2 \\ &= 0^* + 0^*11^* \\ &= 0^*(\epsilon + 11)^* \\ \underline{R.E} &\Rightarrow \underline{0^*(\epsilon + 1)^*} \end{aligned}$$

Ex) obtain a R.E for the following finite state machine.

Sol?



Sol?



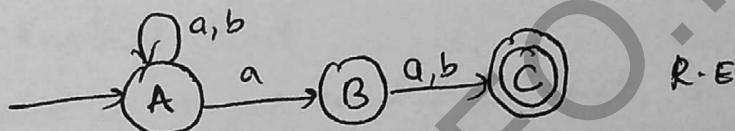
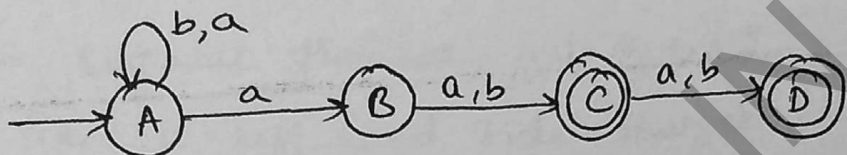
$$R.E = R.E_1 + R.E_2$$

$$\underline{R.E = 1^*0(1+11^*0)^* + (1+01^*1)^*01^*1(0+1)^*}$$

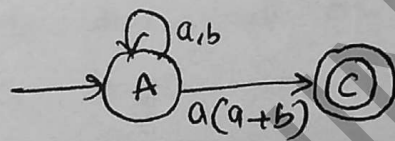
Eq) obtain an regular expression for a finite state machine given below:

	a	b
→ A	A, B	A
B	C	C
x C	D	D
x D	∅	∅

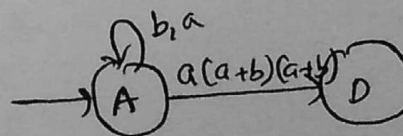
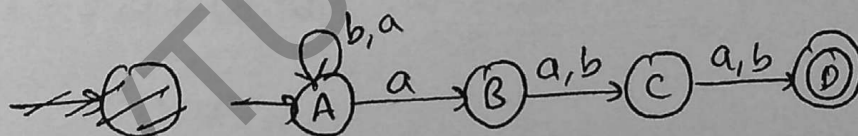
Sol:



R.E



$$R.E_1 = (a+b)^* a (a+b)$$



$$R.E_2 = (a+b)^* a (a+b) (a+b)$$

$$R.E = [(a+b)^* a (a+b)] + [(a+b)^* a (a+b) (a+b)]$$

Regular Grammar

(9)

Definition: A regular grammar G is a quadruple (V, T, P, S) where

$$\text{i.e. } G = (V, T, P, S)$$

where:

V : the set of variables or nonterminals

T : the set of terminals

P : Productions

S : Start Symbol.

In regular grammar, all rules in R must:

* have a left-hand side that is a single non-terminal and

* have a right hand side that is ϵ or a single terminal or a single terminal followed by a single non-terminal.

(Eg) Let $L = \{w \in (a,b)^* : w \text{ ends with pattern } aaaa\}$

$$S \rightarrow as$$

$$S \rightarrow bs$$

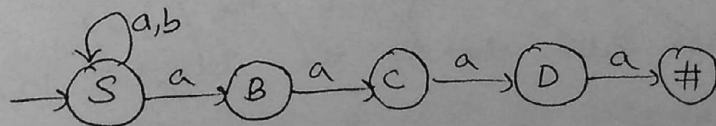
$$S \rightarrow aB$$

$$B \rightarrow aC$$

$$C \rightarrow aD$$

$$D \rightarrow a$$

Applying grammar to form to this grammar we get



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

$$V = \{S, B, C, D\}$$

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

$$P = \{ \begin{array}{l} S \rightarrow aS \\ S \rightarrow bS \\ S \rightarrow aB \\ B \rightarrow aC \\ C \rightarrow aD \\ D \rightarrow a \end{array} \}$$

S is a start symbol.

Some Important closure properties of Regular languages

Closure under

- * Union
- * concatenation
- * Kleene star
- * Complement
- * Intersection
- * difference
- * reverse
- * letter substitution.

Theorem: The regular languages are closed under Union, concatenation & Kleene star.

Proof: closure under union, concatenation & star
 If L_1 & L_2 are regular, the $L_1 \cup L_2$, $L_1 \cdot L_2$ & L_1^* are also denote the regular languages

Theorem: The regular languages are closed under Union, concatenation & Kleene star.

Proof: It is given that L_1 & L_2 are regular languages. So there exists regular expression R_1 & R_2 such that:

$$L_1 = L(R_1)$$

$$L_2 = L(R_2)$$

By the definition of regular expressions, we have

* $R_1 + R_2$ is a regular expression denoting the language $L_1 \cup L_2$.

* $R_1 \cdot R_2$ is a regular expression denoting the language $L_1 \cdot L_2$.

* R_1^+ is a regular expression denoting the language L_1^+ .

So, the regular languages are closed under Union, concatenation & star operation.

Theorem: If L & M are regular languages, then LM is regular.

Proof: Let L & M be the languages of automata

$$A_L = (Q_L, \Sigma, \delta_L, q_L, F_L) \text{ \& \& } A_M = (Q_M, \Sigma, \delta_M, q_M, F_M) \text{ \& }$$

we are assuming that the alphabets of both automata are same.

For LM we shall construct an automata A that simulates both A_L & A_M . The states of A

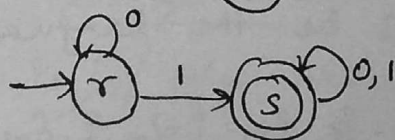
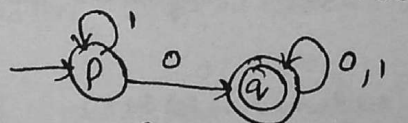
are pairs of states, the first from A_L & the second from A_m to design the transitions of A . Suppose A is in state (p, q) where p is the state of A_L & q is the state of A_m . If a is the input symbol, we see what A_L does on input a , say it goes to state s . we also see what A_m does on input a , say it makes a transition to state t . Then the next state of A will be (s, t) . In that manner A has simulated the effect of both A_L & A_m .

The start state of A is the pair of start states of A_L & A_m . Since we want to accept if & only if both automata accept, we select as the accepting states of A all those pairs (p, q) such that p is an accepting state of A_L & q is an accepting state of A_m . Formally we define

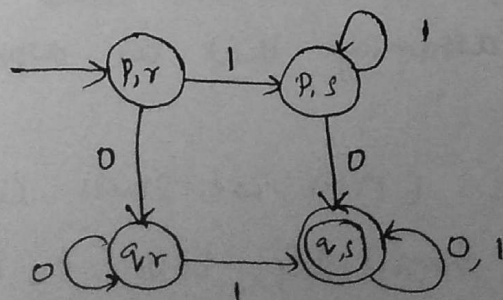
$$A = (Q_L \times Q_m, \Sigma, \delta, (q_{L1}, q_{m1}), F_L \times F_m)$$

The w is accepted by A if & only if both A_L & A_m accept w . Thus A accepts the intersection of L & M .

Ex



Soln.



Module-3

Context-Free Grammar (CFG)

A context-free grammar (or CFG) to be a grammar in which each rule must:

- * have a left-hand side that is a single nonterminal $\&$
- * have a right-hand side that is ϵ or terminal; nonterminal one or more.

A context grammar G is 4 tuple (or) Quad tuple

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

where

V is set of variables / non terminals

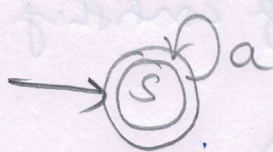
T is set of terminals

P is set of production

Each production is of the form $\alpha \rightarrow \beta$ where α is a string from variable $\&$ α cannot be ϵ , but β is string from $(V \cup T)^*$. Hence it can include ϵ also.

(Eg)

Obtain grammar to generate string consisting of any no of a's



$$\begin{aligned} \delta(S, a) &= S & S &\rightarrow as / \epsilon \\ \Downarrow & & \Downarrow & \\ S &\rightarrow aS & & \\ S &\rightarrow \epsilon & & \end{aligned}$$

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

$$V = \{S\}$$

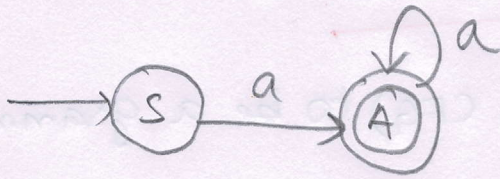
$$T = \{a\}$$

$$P = \{S \rightarrow as / \epsilon\}$$

S is start symbol.

(Eq) Obtain a grammar to generate string consisting of atleast 1a.

Sol?



$$\delta(S, a) = A$$

$$S \rightarrow aA$$

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

$$A \rightarrow aA$$

$$A \rightarrow \epsilon$$

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

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

$$T = \{a\}$$

$$P = \{S \rightarrow aA$$

$$A \rightarrow aA / \epsilon\}$$

S is start symbol.

(Eq) Obtain a grammar to generate string consisting of even no of a's

Sol?



$$\delta(S, a) = A$$

$$S \rightarrow aA$$

$$S \rightarrow \epsilon$$

$$\delta(A, a) = S$$

$$A \rightarrow aS$$

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

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

$$T = \{a\}$$

$$P = \{S \rightarrow aA / \epsilon$$

$$A \rightarrow aS\}$$

S is start symbol.

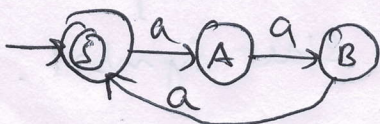
(Eq) Obtain a grammar to generate string consisting of multiples of 3 a's.

Sol?



$$S \rightarrow aaas / \epsilon$$

or



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

$$V = \{S\}$$

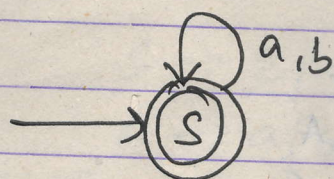
$$T = \{a\}$$

$$P = \{S \rightarrow aaas / \epsilon\}$$

S is start symbol.

③. obtain grammar to generate string consisting of any number of a's & b's

Soln



$$S \rightarrow \epsilon$$

$$S \rightarrow aS$$

$$S \rightarrow bS$$

$$\text{so } S \rightarrow \epsilon / aS / bS$$

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

$$V = \{ S \}$$

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

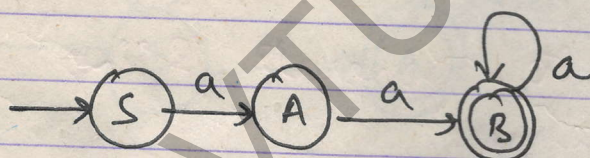
$$P = \{$$

$$S \rightarrow \epsilon / aS / bS \}$$

S is start symbol

④. obtain grammar to generate string consisting of at least two a's

Soln



$$S \rightarrow aA$$

$$A \rightarrow aB$$

$$B \rightarrow \epsilon$$

$$B \rightarrow aB$$

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

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

$$T = \{ a \}$$

$$P = \{$$

$$S \rightarrow aA$$

$$A \rightarrow aB$$

$$B \rightarrow \epsilon / aB \}$$

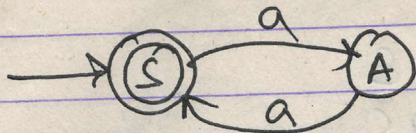
$$\text{Btw } aB \}$$

S is the start symbol

(eg)

obtain grammar to generate string consisting of even number of a's.

Solⁿ



$$S \rightarrow \epsilon$$

$$S \rightarrow aA$$

$$A \rightarrow aS$$

or

$$S \rightarrow \epsilon / aas \Rightarrow \text{Diagram: } S((S)) \xrightarrow{a} A((A)) \xrightarrow{a} S$$

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

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

$$T = \{a\}$$

$$P = \left\{ \begin{array}{l} S \rightarrow \epsilon / aA \\ A \rightarrow aS \end{array} \right\}$$

where S is start symbol.

(eg)

obtain grammar to generate string consist of multiples of three a's.

Solⁿ

$$S \rightarrow \epsilon / aaas$$

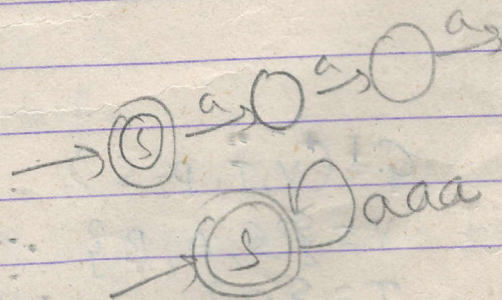
$$G = (V, T, P, S)$$

$$V = \{S\}$$

$$T = \{a\}$$

$$P = \{ S \rightarrow \epsilon / aaas \}$$

where S is start symbol.



(eg)

obtain grammar to generate strings of a's & b's such that string length is multiple of 3

Soln

$$S \rightarrow \epsilon / AAAS$$

$$A \rightarrow a/b$$

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

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

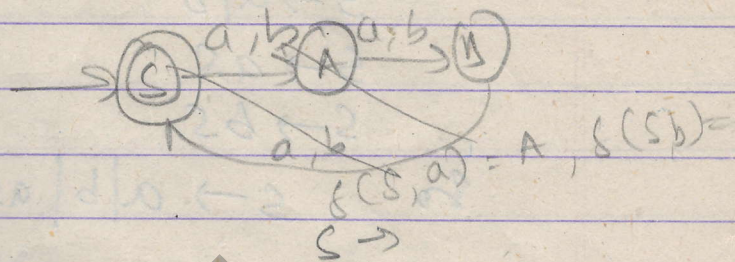
$$T = \{a, b\}$$

$$P = \{$$

$$S \rightarrow \epsilon / AAAS$$

$$A \rightarrow a/b\}$$

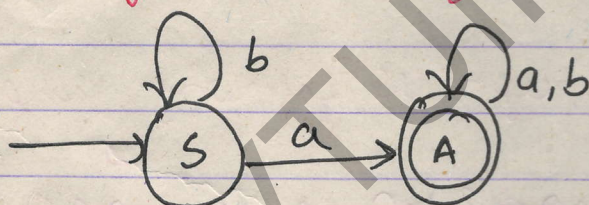
S is the start symbol.



(eg)

obtain grammar to generate string consisting of any number of a's & b's with atleast one a.

Soln



$$S \rightarrow bS$$

$$S \rightarrow aA$$

$$A \rightarrow \epsilon / aA / bA$$

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

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

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

$$P = \{ S \rightarrow aA / bS$$

$$A \rightarrow aA / bA / \epsilon \}$$

S is the start symbol.

(eg) obtain grammar to generate string consisting of any no of a's & b's with at least one a or at least one b.

Sol

$$S \rightarrow a|b$$

$$S \rightarrow aS$$

$$S \rightarrow bS$$

$$\text{So } S \rightarrow a|b|aS|bS$$

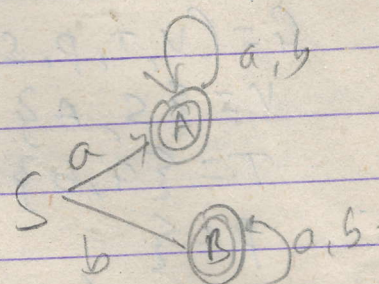
$$G = (V, T, P, S)$$

$$V = \{S\}$$

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

$$P = \{S \rightarrow a|b|aS|bS\}$$

S is the start symbol.



$$S \rightarrow aA|bB$$

$$S \rightarrow aA$$

$$A \rightarrow aA|bA|\epsilon$$

$$S \rightarrow bB$$

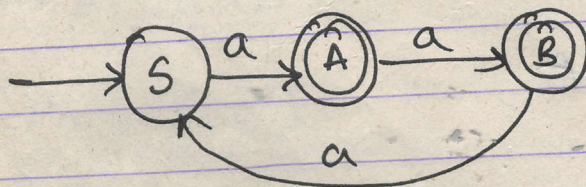
$$B \rightarrow aB|bB|\epsilon$$

(eg) obtain grammar to accept the following language

$$L = \{w : |w| \bmod 3 \neq 0 \text{ where } w \in \{a\}^*\}$$

Sol

$$S \rightarrow a|aa|aaaS$$



$$S \rightarrow aA$$

$$A \rightarrow aB|\epsilon$$

$$B \rightarrow aS|\epsilon$$

$$1 \bmod 3 = 1$$

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

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

$$T = \{a\}$$

$$P = \{S \rightarrow aA$$

$$A \rightarrow aB / \epsilon$$

$$B \rightarrow aS / \epsilon\}$$

S is the start symbol.

Grammar from Regular Expression

(eg). obtain grammar to generate strings of a's & b's having a substring ab.

Soln. The RE representing strings of a's & b's having a substring is given by

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

$$\begin{array}{ccc} \Downarrow & \Downarrow & \Downarrow \\ A & ab & A \end{array}$$

$$S \rightarrow AabA$$

$$A \rightarrow \epsilon \mid aA \mid bA$$

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

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

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

$$P = \{ S \rightarrow AabA$$

$$A \rightarrow \epsilon \mid aA \mid bA \}$$

S is start symbol.

(eg). obtain grammar to generate string of a's ending with string ab.

Soln

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

$$\begin{array}{cc} \Downarrow & \Downarrow \\ A & ab \end{array}$$

$$S \rightarrow Aab$$

$$A \rightarrow \epsilon \mid aA \mid bA$$

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

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

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

$$P = \{$$

$$S \Rightarrow Aab$$

$$A \rightarrow aA \mid bA \mid \epsilon \}$$

S is the start symbol.

eg. obtain grammar to generate strings of a's & b's starting with ab.

Soln

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

$$\Downarrow \quad \Downarrow$$

$$ab \quad A$$

$$S \rightarrow abA$$

$$A \rightarrow \epsilon \mid aA \mid bA$$

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

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

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

$$P = \{ S \rightarrow abA$$

$$A \rightarrow \epsilon \mid aA \mid bA \}$$

S is the start symbol.

eg. obtain the grammar to generate the following language

$$L = \{w : n_a(w) \bmod 2 = 0 \text{ where } w \in \{a, b\}^*\}$$

Solⁿ

$$\begin{array}{c} b^* a b^* a b^* \\ \downarrow \downarrow \downarrow \downarrow \downarrow \\ S a S a S \end{array}$$

$$S \rightarrow S a S a S$$

$$S \rightarrow \epsilon / b S$$

$$S \rightarrow \epsilon / b S / S a S a S$$

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

$$V = \{S\}$$

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

$$P = \{ S \rightarrow \epsilon / b S / S a S a S \}$$

S is start symbol.

$$\begin{array}{l} b^* a b^* a b^* \\ S \Rightarrow A a A a A / b A / \epsilon \\ A \rightarrow \epsilon / b A \\ \cancel{S \Rightarrow \epsilon} \end{array}$$

Grammars for other languages

(Eg). obtain a grammar to generate the following language

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

Solⁿ

$$S \rightarrow \epsilon / a S b$$

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

$$V = \{S\}$$

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

$$P = \{ S \rightarrow \epsilon / a S b \}$$

S is the start symbol

$$S \Rightarrow \epsilon$$

$$S \Rightarrow a S b$$

$$\Rightarrow ab$$

$$S \Rightarrow a S b$$

$$\Rightarrow a a S b b$$

$$\Rightarrow \underline{\underline{a a a b b b}}$$

(Eq) - obtain a grammar to generate the following language

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

soln.

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

$$S \rightarrow ab | aSb$$

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

$$V = \{S\}$$

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

$$P = \{S \rightarrow ab | aSb\}$$

S is the start symbol.

(Eq) - obtain a grammar to generate the following lang

$$L = \{a^{n+1} b^n : n \geq 0\}$$

soln.

$$L = \{a \boxed{ab}, a \boxed{abb}, a \boxed{aaabbb}, \dots\}$$

$$S \rightarrow a | aSb$$

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

$$V = \{S\}$$

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

$$P = \{S \rightarrow a | aSb\}$$

S is start symbol.

(eg) obtain a grammar to generate the following lang.

$$L = \{a^n b^{n+1} : n \geq 0\}$$

Soln.

$$S \rightarrow b | aSb$$

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

$$V = \{S\}$$

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

$$P = \{S \rightarrow b | aSb\}$$

S is start symbol.

(eg) obtain a grammar to generate the following lang.

$$L = \{a^n b^{n+2} : n \geq 0\}$$

Soln. Two extra b's should be generated. So, the final grammar to generate given lang is

$$S \rightarrow bb | aSb$$

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

$$V = \{S\}$$

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

$$P = \{S \rightarrow bb | aSb\}$$

S is start symbol.

(Eg) obtain a grammar to generate the following language

$$L = \{a^n b^{2n} : n \geq 0\}$$

Solⁿ

$$S \rightarrow \epsilon / aSbb$$

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

$$V = \{S\}$$

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

$$P = \{S \rightarrow \epsilon / aSbb\}$$

S is start symbol.

(Eg) let $\Sigma = \{a, b\}$ obtain a grammar G generating set of all palindromes over Σ .

Solⁿ

$$S \rightarrow \epsilon / a / b / aSa / bSb$$

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

$$V = \{S\}$$

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

$$P = \{S \rightarrow \epsilon$$

$$S \rightarrow a / b$$

$$S \rightarrow aSa / bSb\}$$

S is the start symbol.

Madam

$$S \rightarrow aSa / bSb$$

(eg) obtain a grammar to generate a lang consisting of all non-palindromes over $\{a, b\}$

sol

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

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

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

$$P = \{$$

$$S \rightarrow aSa \mid bSb$$

$$S \rightarrow A$$

$$A \rightarrow aBb \mid bBa$$

$$B \rightarrow aB \mid bB \mid \epsilon \}$$

S is start symbol.

(eg) obtain a grammar to generate the following language

$$L = \{WW^R \text{ where } W \in \{a, b\}^*\}$$

$$L = \{aa, bb, abba, baab, \dots\}$$

sol

$$S \rightarrow \epsilon \mid aSa \mid bSb$$

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

$$V = \{S\}$$

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

$$P = \{S \rightarrow \epsilon \mid aSa \mid bSb\}$$

S is start symbol.

(eg) Obtain the grammar to generate the language

$$L = \{0^m 1^n 2^n \mid m \geq 1, n \geq 0\}$$

Soln

$$L = \{0^m 1^n 2^n \mid m \geq 1, n \geq 0\}$$

$$S \rightarrow A B$$

The variable A should produce m no of 0's & followed by n no of 1's.

$$A \rightarrow 01 \mid 0A1$$

B should produce any no of 2's

$$B \rightarrow \epsilon \mid 2B$$

$$S \rightarrow A B$$

$$A \rightarrow 01 \mid 0A1$$

$$B \rightarrow \epsilon \mid 2B$$

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

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

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

$$P = \{ S \rightarrow A B, \\ A \rightarrow 01 \mid 0A1, \\ B \rightarrow \epsilon \mid 2B \}$$

S is start symbol.

(eg) obtain the grammar to generate the lang

$$L = \{ w \mid n_a(w) = n_b(w) \}$$

soln

$$S \rightarrow \epsilon / asb / bsa / SS \quad \text{string starting & ending with the same str}$$

eg: abba, bnaab

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

$$V = \{ S \}$$

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

$$P = \{$$

$$S \rightarrow \epsilon / asb / bsa / SS$$

$$\}$$

S is start symbol.

(eg) obtain a grammar to generate a string balanced parentheses.

soln

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

$$V = \{ S \}$$

$$T = \{ (,) \}$$

$$P = \{ S \rightarrow (S) / SS / \epsilon \}$$

S is start symbol.

$$S \rightarrow (S) / \epsilon / SS$$

(eg)

obtain a grammar to generate the lang:

$$L = \{a^i b^j \mid i \neq j, i \geq 0 \text{ \& } j \geq 0\}$$

Solⁿ

$$L = \{a^i b^j \mid i \neq j, i \geq 0, j \geq 0\}$$

$$S \rightarrow a S b \mid A \mid B$$

$$A \rightarrow a \mid a A$$

$$B \rightarrow b \mid b B$$

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

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

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

$$P = \{ S \rightarrow a S b \mid A \mid B \}$$

$$A \rightarrow a \mid a A$$

$$B \rightarrow b \mid b B \}$$

S is start symbol.

$$L = \{0^i 1^j \mid i \neq j, i \geq 0, j \geq 0\}$$

$$S \rightarrow 0 S 1 \mid A \mid B$$

$$A \rightarrow 0 \mid 0 A$$

$$B \rightarrow 1 \mid 1 B$$

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

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

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

$$P = \{ S \rightarrow 0 S 1 \mid A \mid B \}$$

$$A \rightarrow 0 \mid 0 A$$

$$B \rightarrow 1 \mid 1 B$$

(eg)

obtain a grammar to generate the language

$$L = \{a^{n+2} b^m \mid n \geq 0 \text{ \& } m > n\}$$

Solⁿ

$$S \rightarrow a A B$$

$$A \rightarrow a A b \mid a b$$

$$B \rightarrow b B \mid \epsilon$$

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

$$V = \{S, A, B\}, T = \{a, b\} P = \{ S \rightarrow a A B, A \rightarrow a A b \mid a b, B \rightarrow b B \mid \epsilon \}$$

S is start symbol

(eg)

obtain a grammar to generate the language

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

Sol?

$$n=0$$

$$n=1$$

$$n=2$$

$$m \geq 1$$

$$m \geq 2$$

$$m \geq 3$$

$$L = \{\epsilon b b^*, a b b^*, a a b b b^*, \dots\}$$

~~S → aSb / B~~

$S \rightarrow aSb / B$ [Generates $a^n b^n \mid n \geq 0$ followed by at least one b]

$B \rightarrow bB / b$ [Generates one or more b 's]

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

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

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

$$P = \{S \rightarrow aSb / B, \\ B \rightarrow bB / b\}$$

S is start symbol.

(eg)

obtain a grammar to generate the language

$$L = \{a^n b^{n-3} \mid n \geq 3\}$$

Sol?

$$n=3$$

$$n=4$$

$$n=5$$

$$L = \{aaa, aaaab, aaaaaabb, \dots\}$$

$$P = \{S \rightarrow aaaaA \\ A \rightarrow aAb \mid \epsilon\}$$

Derivation.

Defⁿ: Let $A \rightarrow \alpha B \gamma$ & $B \rightarrow \beta$ are production in grammar G , where α, β and γ are strings of terminal and/or non-terminals. A and B are non-terminals. The non-terminal A produces the string $\alpha \beta \gamma$ by replacing the non-terminal B in $\alpha B \gamma$ by the string β by applying the production $B \rightarrow \beta$. It can be written as

$$A \Rightarrow \alpha \beta \gamma$$

This process of obtaining string of terminals &/or non-terminals from the start symbol by applying some or all productions is called derivation.

If a string is obtained by applying only one prodⁿ, then it is called one-step derivation & is denoted by the symbol ' $\Rightarrow$ '. If one or more productions are applied to get the string $\alpha \beta \gamma$ from A then we write

$$A \Rightarrow^+ \alpha \beta \gamma$$

If 2 or more prodⁿ are applied to get the string $\alpha \beta \gamma$ from A , then we write

$$A \Rightarrow^* \alpha \beta \gamma.$$

(Eg)^{FA} Consider the grammar shown below from which any arithmetic expⁿ can be obtained.

$$E \rightarrow E + E$$

$$E \rightarrow E - E$$

$$E \rightarrow E * E$$

$$E \rightarrow E / E$$

$$E \rightarrow id.$$

E is considered to be the start symbol. Obtain the string $id + id * id$ & show the derivation for it.
Same

Sol?

$$E \Rightarrow E + E$$

$$\Rightarrow id + E$$

$$\Rightarrow id + E * E$$

$$\Rightarrow id + id * E$$

$$E \Rightarrow \underline{id + id * id}$$

Sentence or Sentential form

Defⁿ: Let $G = (V, T, P, S)$ be a grammar. The string w obtained from the grammar G such that $S \Rightarrow^+ w$ is called Sentence of grammar G . Here w is string of terminals.

In the above (eg) $id + id * id$ is the sentence of the grammar.

If there is a derivation $S \xRightarrow{*} \alpha$, where α contains

Strings of terminals and/or non-terminals, then α is called ~~sentential~~^{sent} & sentential form of G .
In the derivation shown in above eg.

$E + E, id + E, id + E * E, id + id * E, id + id * id$
are all sentential forms of the grammar.

Language

Defⁿ: Let $G = (V, T, P, S)$ be a grammar. The language $L(G)$ generated by the grammar G is

$$L(G) = \{ w \mid S \xrightarrow{*} w \text{ \& } w \in T^* \}$$

i.e., w is a string of terminals (may be ϵ) obtained from the start symbol S by applying an arbitrary number of productions.

Leftmost Derivation

Defⁿ: In the derivation process if a left most variable is replaced at every step, then the derivation is said to be leftmost.

(eg) obtain the leftmost derivation for the string $id + id * id$ using the following grammar

$$E \rightarrow E + E$$

$$E \rightarrow E - E$$

$$E \rightarrow E * E$$

$$E \rightarrow E / E$$

$$E \rightarrow E \wedge E$$

$$E \rightarrow id$$

$\wedge$: denotes exponential

Solⁿ

$$E \xRightarrow[\text{lm}]{+} E + E$$

$$\Rightarrow id + E$$

$$\Rightarrow id + E * E$$

$$\Rightarrow id + id * E$$

$$\Rightarrow id + id * id$$

The string $id + id * id$ is obtained from the start symbol E by applying leftmost derivation. It can be written as

$$E \xRightarrow[\text{lm}]{+} id + id * id$$

(Eg) ✓ obtain the leftmost derivation for the string $aaabbabbba$ using the following grammar

$$S \rightarrow aB \mid bA$$

$$A \rightarrow aS \mid bAA \mid a$$

$$B \rightarrow bS \mid aBB \mid b$$

Soln

$$S \xRightarrow{lm} aB$$

(Applying $S \rightarrow aB$)

$$\Rightarrow a a B B$$

($B \rightarrow a B B$)

$$\Rightarrow a a a B B B$$

($B \rightarrow a B B$)

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

($B \rightarrow b$)

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

($B \rightarrow b$)

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

($B \rightarrow a B B$)

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

($B \rightarrow b$)

$$\Rightarrow a a a b b a b b S$$

($B \rightarrow b S$)

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

($S \rightarrow b A$)

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

($A \rightarrow a$)

Rightmost Derivation

Defⁿ: In the derivation process if a right most variable is replaced at every step, then the derivation is said to be rightmost.

(Eg) obtain the rightmost derivation for the string $id + id * id$ using following grammar.

$$E \rightarrow E + E$$

$$E \rightarrow E * E$$

$$E \rightarrow E - E$$

$$E \rightarrow E / E$$

$$E \rightarrow id$$

Solⁿ

$$E \xRightarrow{rm} E + E$$

$$\Rightarrow E + E * E$$

$$\Rightarrow E + E * id$$

$$\Rightarrow E + id * id$$

$$\Rightarrow id + id * id.$$

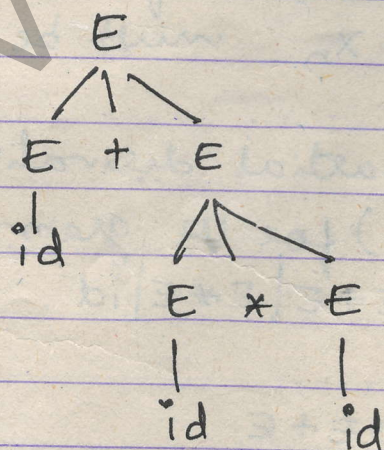
Derivation Tree (parse tree)

The derivation can be shown in the form of a tree. Such trees are called derivation or parse trees.

Defⁿ (Parse tree or derivation tree): Let $G = (V, T, P, S)$ be a CFG. The tree is derivation tree (parse tree) with the following properties.

1. The root has the label S
2. Every vertex has a label which is in $(V \cup T \cup E)$
3. Every leaf node has label from T & an interior vertex has a label from V .
4. If a vertex is labeled A & if $X_1, X_2, X_3, \dots, X_n$ are all children of A from left, then
 $A \rightarrow X_1 X_2 X_3 \dots X_n$ must be a production in P .

parse tree for above eg.



The string $id + id * id$ is called yield of the tree.

Yield of the tree

The yield of a tree is the string of symbols obtained by only reading the leaves of the tree from left to right with considering the ϵ -symbols. The yield of a tree is always a terminal string.

Defⁿ (partial parse tree or partial derivation tree): Let $G = (V, T, P, S)$ be a CFG. The tree is partial derivation tree (parse tree) with the following properties:

- ① The ^{root} has the label S
- ② Every vertex has a label which is in $(V \cup T \cup \epsilon)$
- ③ Every leaf node has a label from $(V \cup T \cup \epsilon)$.
- ④ If a vertex is labeled A & if $x_1, x_2, x_3, \dots, x_n$ are all children of A from left. Then $A \rightarrow x_1 x_2 x_3 \dots x_n$ must be a prodⁿ in P .

eg) consider the partial derivation (by applying right most derⁿ) for the grammar.

$$E \rightarrow E + E \mid E * E \mid id.$$

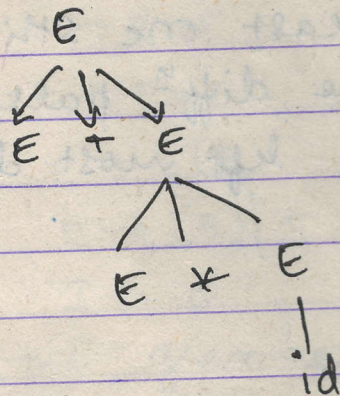
is shown below

$$E \rightarrow E + E$$

$$\Rightarrow E + E * E$$

$$\Rightarrow E + E * id$$

for this partial right most derivation, the partial derivation tree is shown below



It is clear from the parse tree & partial parse tree that all the leaves in parse tree are the symbols from $(T \cup E)$ whereas in partial parse tree the leaves will be from $(V \cup T \cup E)$.

Ambiguous Grammar

Defⁿ: Let $G = (V, T, P, S)$ be a context free grammar. A grammar G is ambiguous if & if there exists at least one string $w \in T^*$ for which two or more diff^y parse trees exists by applying either the left most derivation or right most derivation.

Note:

- 1) obtain the leftmost derivation & get a string w . obtain the rightmost derivation & get a string w . For both the derivations construct the parse tree. If there are two diff^y parse trees, then the grammar is ambiguous.
- 2) obtain the string w by applying leftmost derivation twice & construct the parse tree. If the two parse trees are different, the grammar is ambiguous.
- 3) obtain the string w by applying rightmost derivation twice & construct the parse tree. If the two parse trees are diff^y, the grammar is ambiguous.

eg) Consider the grammar shown below

$$E \rightarrow E + E$$

$$E \rightarrow E - E$$

$$E \rightarrow E * E$$

$$E \rightarrow E / E$$

$$E \rightarrow (E) | I$$

$$I \rightarrow id$$

Show that the grammar is ambiguous.

The sentence $id + id * id$ can be obtained from leftmost derivation in two ways as shown below

$$E \Rightarrow E + E$$

$$\Rightarrow id + E$$

$$\Rightarrow id + E * E$$

$$\Rightarrow id + id * E$$

$$\Rightarrow id + id * id$$

$$E \Rightarrow E * E$$

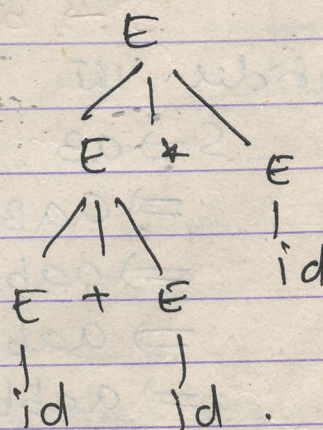
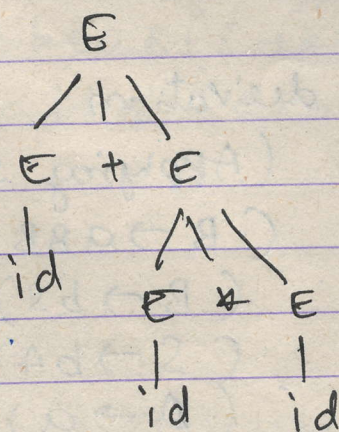
$$\Rightarrow E + E * E$$

$$\Rightarrow id + E * E$$

$$\Rightarrow id + id * E$$

$$\Rightarrow id + id * id$$

The corresponding derivation trees for the two leftmost ~~derivation~~ derivations are shown below



Since the two parse trees are diffⁿ for the same sentence $id + id * id$ by applying leftmost derivation, the grammar is ambiguous.

(Eg)

Is the following grammar ambiguous?

$$S \rightarrow as | x$$

$$x \rightarrow ax | a$$

Sol

Consider the two leftmost derivations for the string aaaa.

$$S \Rightarrow as$$

$$\Rightarrow aas$$

$$\Rightarrow aaas$$

$$\Rightarrow aaax$$

$$\Rightarrow aaaa$$

$$S \Rightarrow x$$

$$\Rightarrow ax$$

$$\Rightarrow aax$$

$$\Rightarrow aaax$$

$$\Rightarrow aaaa$$

Since there are two leftmost derivations for the same sentence aaaa, the given grammar is ambiguous.

Show that the grammar is ambiguous for the string

(Eg)

Is the following grammar ambiguous? aa.bbab

$$S \rightarrow aB | bA$$

$$A \rightarrow aS | bAA | a$$

$$B \rightarrow bS | aBB | b$$

$$\begin{aligned} S &\Rightarrow aB \\ &\Rightarrow abs \\ &\Rightarrow abab \\ &\Rightarrow abab \end{aligned}$$

Sol

Consider the leftmost derivation

$$S \Rightarrow aB$$

$$\Rightarrow aabb$$

$$\Rightarrow aabbsB$$

$$\Rightarrow aabbbaB$$

$$\Rightarrow aabbbaB$$

$$\Rightarrow aabbbaab$$

(Applying $S \rightarrow aB$)

($B \rightarrow aBB$)

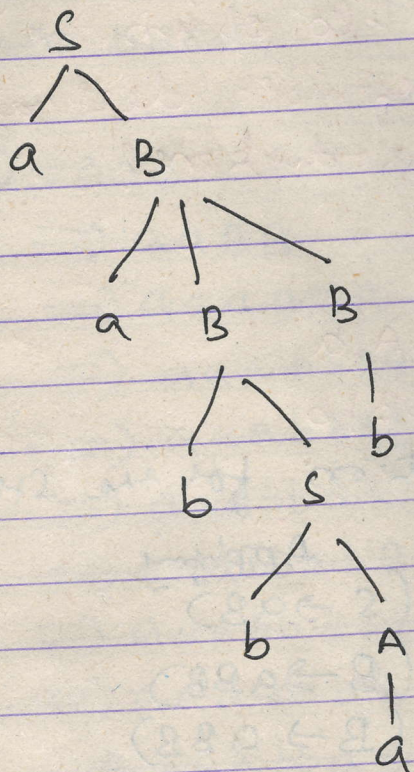
($B \rightarrow bS$)

($S \rightarrow bA$)

($A \rightarrow a$)

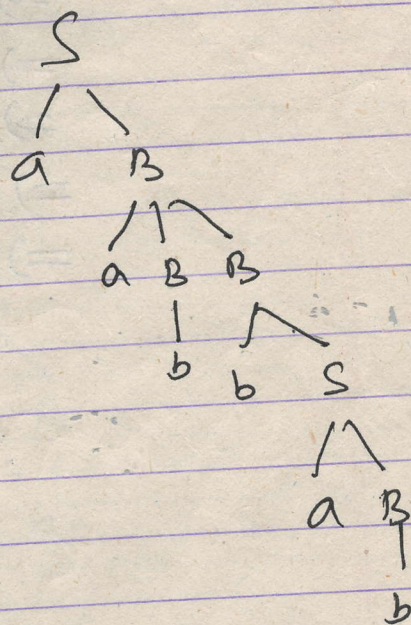
($B \rightarrow b$)

parse tree



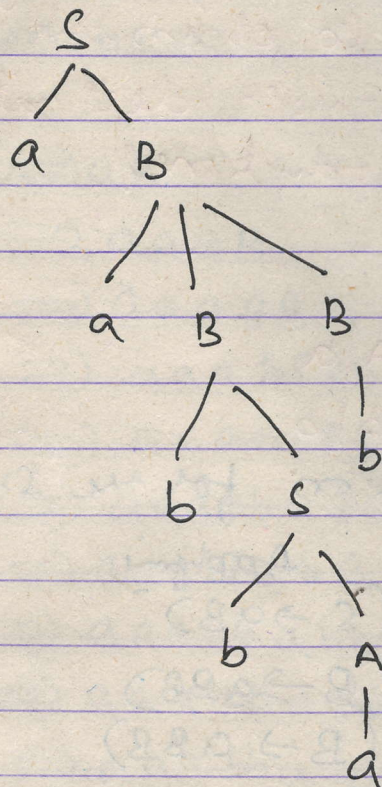
The same string aabbab can be obtained again by applying leftmost derivation as shown below.

$S \Rightarrow aB$ ($S \rightarrow aB$)
 $\Rightarrow aabb$ ($B \rightarrow aBB$)
 $\Rightarrow aabb$ ($B \rightarrow b$)
 $\Rightarrow aabbs$ ($B \rightarrow bS$)
 $\Rightarrow aabbab$ ($S \rightarrow aB$)
 $\Rightarrow aabbab$ ($B \rightarrow b$)



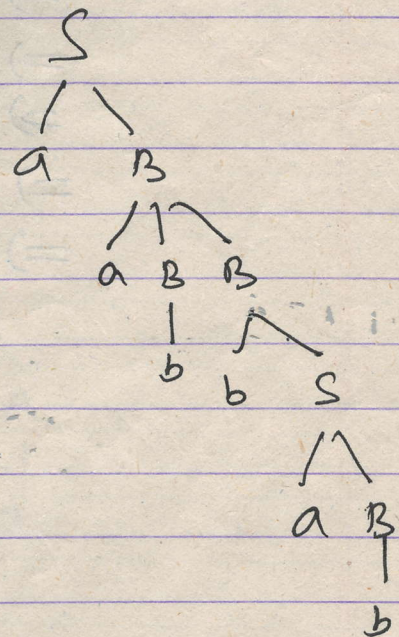
There are two parse trees for the string aabbab by applying leftmost derivation $\Rightarrow$ so the given grammar is ambiguous.

parse tree



The same string aabbab can be obtained again by applying leftmost derivation as shown below.

$S \Rightarrow aB$ ($S \rightarrow aB$)
 $\Rightarrow aabb$ ($B \rightarrow aBB$)
 $\Rightarrow aabbB$ ($B \rightarrow b$)
 $\Rightarrow aabbS$ ($B \rightarrow bS$)
 $\Rightarrow aabbab$ ($S \rightarrow aB$)
 $\Rightarrow aabbab$ ($B \rightarrow b$)



There are two parse trees for the string aabbab by applying leftmost derivation $\Rightarrow$ to the grammar is ambiguous.

(Eg) Obtain the string $aaabbabbba$ by applying leftmost derivation & the parse tree for the grammar shown below. Is it possible to obtain the same string again by applying leftmost derivation but by selecting different productions?

$$S \rightarrow aB \mid bA$$

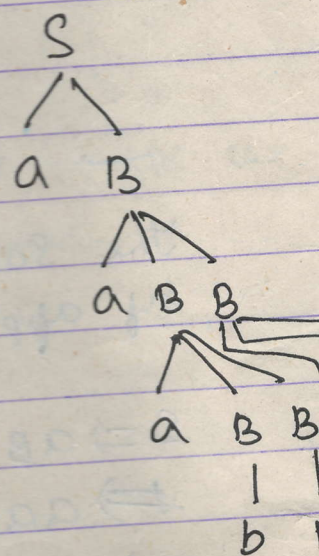
$$A \rightarrow aS \mid bAA \mid a$$

$$B \rightarrow bS \mid aBB \mid b$$

Sol?

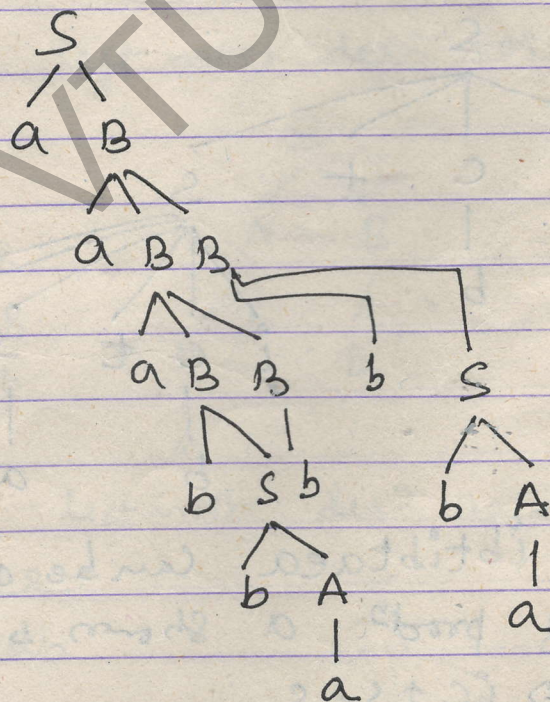
The leftmost derivation for the string $aaabbabbba$

Applying

$$\begin{aligned} S &\Rightarrow aB && (S \rightarrow aB) \\ &\Rightarrow aaBB && (B \rightarrow aBB) \\ &\Rightarrow aaaBBB && (B \rightarrow aBB) \\ &\Rightarrow aaaaBBB && (B \rightarrow b) \\ &\Rightarrow aaabbBB && (B \rightarrow b) \\ &\Rightarrow aaabbabbB && (B \rightarrow aBB) \\ &\Rightarrow aaabbabbB && (B \rightarrow b) \\ &\Rightarrow aaabbabbbs && (B \rightarrow bS) \\ &\Rightarrow aaabbabbbaA && (S \rightarrow bA) \\ &\Rightarrow aaabbabbbaa && (A \rightarrow a) \end{aligned}$$


The leftmost derⁿ for the same string
 $aaabbabbba$ but by applying diffⁿ set of prodⁿ
 is shown below:

$S \Rightarrow aB$ ($S \rightarrow aB$)
 $\Rightarrow aaBB$ ($B \rightarrow aBB$)
 $\Rightarrow aaaBBB$ ($B \rightarrow aBB$)
 $\Rightarrow aaabSBB$ ($B \rightarrow bS$)
 $\Rightarrow aaabbABB$ ($S \rightarrow bA$)
 $\Rightarrow aaabbabb$ ($A \rightarrow a$)
 $\Rightarrow aaabbabbS$ ($B \rightarrow bS$)
 $\Rightarrow aaabbabbba$ ($S \rightarrow bA$)
 $\Rightarrow aaabbabbba$ ($A \rightarrow a$)



Ex) Is the following grammar ambiguous?

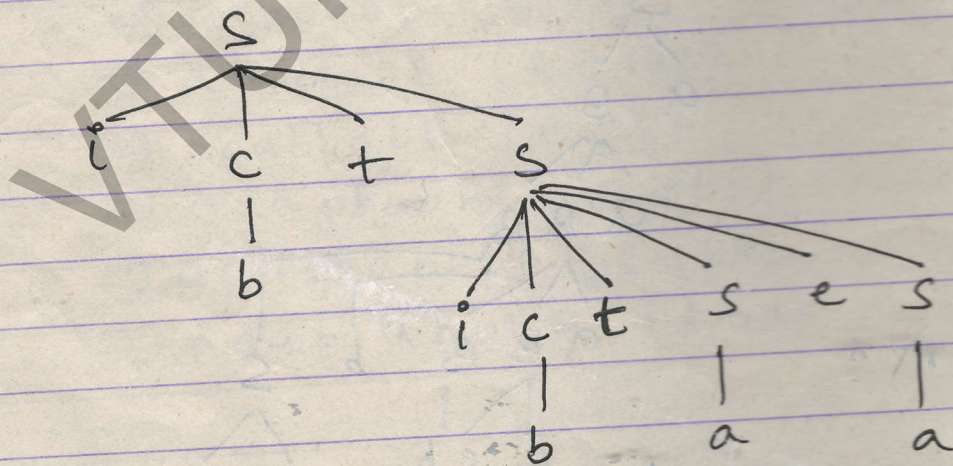
$S \rightarrow icts | ictses | a$

$C \rightarrow b$

Soln. The string ibtibtaea can be obtained by applying the leftmost deriv as shown below

$S \Rightarrow icts$
 $\Rightarrow ibts$
 $\Rightarrow ibticts$
 $\Rightarrow ibtibts$
 $\Rightarrow ibtibtaes$
 $\Rightarrow ibtibtaea$

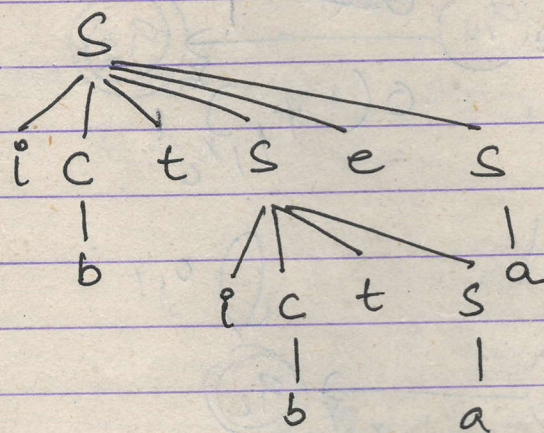
parse tree



The string ibtibtaea can be obtained by using diffⁿ set of prodⁿ as shown below -

$S \Rightarrow ictses$
 $\Rightarrow \cancel{ib}tses$
 $\Rightarrow ibticts$
 $\Rightarrow ibtibts$
 $\Rightarrow ibtibtaes$
 $\Rightarrow ibtibtaea$

The parse tree for this is shown below



Is the grammar ambiguous?

$$S \rightarrow AB/aab$$

$$A \rightarrow a/Aa$$

$$B \rightarrow b.$$

$$S \rightarrow AB \quad S \rightarrow aab$$

$$\Rightarrow Aab$$

$$\Rightarrow \underline{aab}$$

$$\Rightarrow \underline{aab}$$

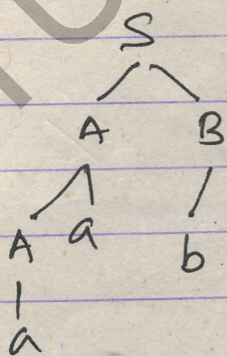
Consider the left most derⁿ for the string aab

$$S \Rightarrow AB$$

$$\Rightarrow AaB$$

$$\Rightarrow aaB$$

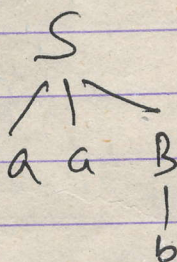
$$\Rightarrow aab$$



consider the left most derⁿ again for the string aab

$$S \Rightarrow aab$$

$$\Rightarrow aab$$



Since there are two parse trees for the string aab, the given grammar is ambiguous.

Properties of context-free languages

The goal of this section is to show that every CFL (without ϵ) is generated by a CFG in which all prod^s are of the form $A \rightarrow BC$ or $A \rightarrow a$, where A, B & C are variables & a is a terminal. This form is called Chomsky Normal form (CNF). To get there, we need to make a number of preliminary simplifications, which are themselves useful in various ways:

- 1) We must eliminate useless symbols, those variables or terminals that do not appear in any derivation of a terminal string from the start symbol.
- 2) We must eliminate ϵ -prod^s, those of the form $A \rightarrow \epsilon$ for some variable A .
- 3) We must eliminate unit prod^s, those of the form $A \rightarrow B$ for variable A & B .

Eliminating useless symbols:

We say a symbol x is useful for a grammar $G = (V, T, P, S)$ if there is a some derivation of the form $S \xRightarrow{*} \alpha x \beta \xRightarrow{*} w$, where w is in T^* . Note that x may be in either V or T , & the sentential form $\alpha x \beta$ might be the first or last in the derivation. If x is not useful, we say it is useless. Evidently, omitting useless symbols from a grammar will not change the language generated. So we may as well detect & eliminate all useless symbols.

our approach to eliminating useless symbols begins by identifying the two things a symbol has to be able to do to be useful:

- 1) we say x is generating if $x \xRightarrow{*} w$ for some terminal string w . Note that every terminal is generating, since w can be that terminal itself, which is derived by zero steps.
- 2) we say x is reachable if there is a derivation $S \xRightarrow{*} \alpha x \beta$ for some α & β .

Clearly a symbol that is useful will be both generating & reachable. If we eliminate the symbols that are not generating first & then eliminate from the remaining grammar those symbols that are not reachable we have only the useful symbols left.

Ex) Eliminate useless symbols from the following grammar

$$S \rightarrow AB|a$$

$$A \rightarrow a.$$

Ex: Step: Find the set of var & the prod^s from which we get only string of terminals.

ov	nv	prod ^s .
$\emptyset$	S, A	$S \rightarrow a$ $A \rightarrow a$
S, A	S, A	$S \rightarrow a$ $A \rightarrow a$

The resulting grammar, from which we get only string of terminals is given by.

$$G_1 = (V_1, T_1, P_1, S)$$

where

$$V_1 = \{ S, A \}$$

$$T_1 = \{ a \}$$

$$P_1 = \{ S \rightarrow a, A \rightarrow a \}$$

Step 12

Eliminate the symbol & production which are not reachable from the start symbol.

P'	T'	V'
-	-	S
$S \rightarrow a$	a	S

So the final grammar obtained after eliminating useless symbols & productions is given by

$$G' = (V', T', P', S)$$

where

$$V' = \{ S \}$$

$$T' = \{ a \}$$

$$P' = \{ S \rightarrow a \}$$

S is the start symbol.

eliminate the useless symbols in the grammar

$$S \rightarrow aA | bB$$

$$A \rightarrow aA | a$$

$$B \rightarrow bB$$

$$D \rightarrow ab | Ea$$

$$E \rightarrow aC | d$$

ov	nv	prod ^s
$\emptyset$	A, E, D	$A \rightarrow a$ $E \rightarrow d$ $D \rightarrow ab$
A, E, D	A, E, D, S	$S \rightarrow aA$ $A \rightarrow aA a$ $D \rightarrow ab Ea$
A, D, E, S	A, D, E, S	-

The resulting grammar $G_1 = (V_1, T_1, P_1, S)$ where

$$V_1 = \{A, D, E, S\}$$

$$T_1 = \{a, b, d\}$$

$$P_1 = \{$$

$$S \rightarrow aA, A \rightarrow aA | a, D \rightarrow ab | Ea, E \rightarrow d\}$$

S is the start symbol.

P'	T'	V'
-	-	S
$S \rightarrow aA$	a	S, A
$A \rightarrow aA a$	a	S, A

The resulting grammar $G' = (V', T', P', S)$ where

$$V' = \{S, A\}$$

$$T' = \{a\}$$

$$P' = \{S \rightarrow aA, A \rightarrow aA|a\}$$

S is the start symbol.

(Q) Eliminate useless symbols from the following grammar

$$S \rightarrow aAB$$

$$A \rightarrow aB|B$$

$$B \rightarrow aB|b|bC$$

$$D \rightarrow Ea$$

$$E \rightarrow a|aE|bC$$

Soln

OV	nv	Production.
$\emptyset$	B, E	$B \rightarrow b$ $E \rightarrow a$
B, E	A, B, D, E	$A \rightarrow aB B$ $B \rightarrow aB$ $D \rightarrow Ea$ $E \rightarrow aE$

OV	nv	prod ⁿ .
A, B, D, E	S, A, B, D, E	$S \rightarrow aAB$
S, A, B, D, E	-	-

$$G_1 = (V_1, T, P_1, S)$$

$$V_1 = \{S, A, B, D, E\}$$

$$T_1 = \{a, b\}$$

$$P_1 = \{S \rightarrow aAB, A \rightarrow aB/b, B \rightarrow aB/b, D \rightarrow Ea, E \rightarrow aE/a\}$$

prod ⁿ (p')	T'	V'
-	-	S
$S \rightarrow aAB$	a	S, A, B
$A \rightarrow aB/b$	a	S, A, B
$B \rightarrow aB/b$	a, b	S, A, B

$$G' = (V', T', P', S)$$

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

$$T' = \{a, b\}$$

$$P' = \{S \rightarrow aAB, A \rightarrow aB/b, B \rightarrow aB/b\}$$

S is the start symbol.

(Eg) Eliminate useless symbols from the following grammar

$$S \rightarrow aA \mid a \mid Bb \mid cC$$

$$A \rightarrow aB$$

$$B \rightarrow a \mid Aa$$

$$C \rightarrow cCD$$

$$D \rightarrow ddd$$

Sol: Step 1:

OV	nv	prod 2.
$\emptyset$	S, B, D	$S \rightarrow a$ $B \rightarrow a$ $D \rightarrow ddd$
S, B, D	S, A, B, D	$S \rightarrow Bb$ $A \rightarrow aB$
S, A, B, D	S, A, B, D	$S \rightarrow aA$ $A \rightarrow aB$ $B \rightarrow Aa$

$$G_1 = (V_1, T_1, P_1, S)$$

$$V_1 = \{S, A, B, D\}$$

$$T_1 = \{a, b, d\}$$

$$P_1 = \{S \rightarrow a \mid Bb \mid aA, A \rightarrow aB, B \rightarrow a \mid Aa, D \rightarrow ddd\}$$

Production (P')	T'	V'
-	-	S
$S \rightarrow a Bb aA$	a, b	S, A, B
$A \rightarrow aB$	a, b	S, A, B
$B \rightarrow a Aa$	a, b	S, A, B

The resulting grammar is

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

$$T' = \{a, b\}$$

$$P' = \{S \rightarrow a | Bb | aA, A \rightarrow aB, B \rightarrow a | Aa\}$$

S is the start symbol.

Eliminating ϵ -productions.

A production of the form $A \rightarrow \epsilon$ is undesirable in a CFG, unless an empty string is derived from the start symbol. Suppose, the language generated from a grammar G does not derive any empty string & the grammar consist of ϵ -productions. Such ϵ -prod^s can be removed.

Defn: Let $G = (V, T, P, S)$ be a CFG. A production in P of the form

$$A \rightarrow \epsilon$$

is called an ϵ -prodⁿ or null prodⁿ. After applying the production the variable A is erased. For each A in V , if there is a derivation of the form

$$A \xRightarrow{*} \epsilon$$

then A is a nullable variable.

(Eg) In grammar if there is ϵ -prod^s of the form

$$B \rightarrow \epsilon$$

$$C \rightarrow \epsilon$$

are ϵ -prod^s and the variable B, C are nullable variables. If there is a production

$$A \rightarrow BC$$

as both B, C are nullable variables, then A is also a nullable variable.

Eliminate all ϵ -productions from the grammar

$$S \rightarrow ABCa/bD$$

$$A \rightarrow BC/b$$

$$B \rightarrow b/\epsilon$$

$$C \rightarrow c/\epsilon$$

$$D \rightarrow d$$

$$B \rightarrow \epsilon$$

$$C \rightarrow \epsilon$$

$$A \rightarrow BC$$

$V = \{B, c, A\}$ are all nullable variables.

productions	resulting prodns
$S \rightarrow ABCa$	$S \rightarrow ABCa BCa ACa ABa ca Aa Ba a$
$S \rightarrow bD$	$S \rightarrow bD$
$A \rightarrow BC/b$	$A \rightarrow BC B c b$
$B \rightarrow b/\epsilon$	$B \rightarrow b$
$C \rightarrow c/\epsilon$	$C \rightarrow c$
$D \rightarrow d$	$D \rightarrow d$

The grammar $G' = (V', T', P', S)$ where

$$V' = \{S, A, B, C, D\}$$

$$T' = \{a, b, c, d\}$$

$$P' = \{ S \rightarrow ABCa|BCa|ACa|ABa|ca|Aa|Ba|a|bD$$

$$A \rightarrow BC|B|c|b$$

$$B \rightarrow b$$

$$C \rightarrow c$$

$$D \rightarrow d$$

S is the start symbol.

(Eq) Eliminate all ϵ -productions from the grammar

$$S \rightarrow BAAB$$

$$A \rightarrow 0A2 \mid 2A0 \mid \epsilon$$

$$B \rightarrow AB \mid 1B \mid \epsilon$$

Solⁿ

$$A \rightarrow \epsilon$$

$$B \rightarrow \epsilon$$

$$S \rightarrow BAAB$$

So S, A, B are all nullable variables.

Productions	Resulting prod ⁿ (P)
$S \rightarrow BAAB$	$S \rightarrow BAAB \mid AAB \mid BAB \mid BAA \mid AB \mid BB \mid BA \mid AA \mid A \mid B$
$A \rightarrow 0A2$	$A \rightarrow 0A2 \mid 02$
$A \rightarrow 2A0$	$A \rightarrow 2A0 \mid 20$
$B \rightarrow AB$	$B \rightarrow AB \mid B \mid A$
$B \rightarrow 1B$	$B \rightarrow 1B \mid 1$

The grammar $G' = (V', T', P', S)$ where

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

$$T' = \{0, 1, 2\}$$

$$P' = \{ S \rightarrow BAAB \mid AAB \mid BAB \mid BAA \mid AB \mid BB \mid BA \mid AA \mid A \mid B \\ A \rightarrow 0A2 \mid 02 \mid 2A0 \mid 20 \\ B \rightarrow AB \mid B \mid A \mid 1B \mid 1 \}$$

S is the start symbol.

Eliminating unit productions

A unit production is a prodⁿ of the form $A \rightarrow B$, where both A & B are variables.

Eliminate all unit prodⁿs from the grammar

$$S \rightarrow AB$$

$$A \rightarrow a$$

$$B \rightarrow C|b$$

$$C \rightarrow D$$

$$D \rightarrow E|bc$$

$$E \rightarrow d|Ab$$

Non unit prodⁿs

$$S \rightarrow AB$$

$$A \rightarrow a$$

$$B \rightarrow b$$

$$D \rightarrow bc$$

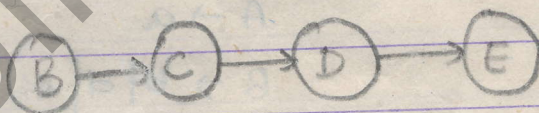
$$E \rightarrow d|Ab$$

unit prodⁿs

$$B \rightarrow C$$

$$C \rightarrow D$$

$$D \rightarrow E$$



It is clear from the dependency graph that $D \rightarrow E$, so the non-unit prodⁿs generated from 'E' can also be generated from 'D'.

$$E \rightarrow d|Ab$$

Since $D \xRightarrow{*} E$

$$D \rightarrow d|Ab$$

The resulting D prodⁿs are

$$D \rightarrow d|Ab|bc$$

Since $C \xRightarrow{*} D \xRightarrow{*} E$

$$C \rightarrow d | Ab | bC$$

Since $B \xRightarrow{*} C, C \xRightarrow{*} D, D \xRightarrow{*} E$

$$B \rightarrow b | d | Ab | bC$$

The final grammar obtained after eliminating unit productions is shown below:

$$V = \{S, A, B, C, D, E\}$$

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

$$P = \{ S \rightarrow AB$$

$$A \rightarrow a$$

$$B \rightarrow b | d | Ab | bC$$

$$C \rightarrow bC | d | Ab$$

$$D \rightarrow bC | d | Ab$$

$$E \rightarrow d | Ab$$

}

S is the start symbol.

(Eg)

Eliminate unit productions from the grammar

$$S \rightarrow A0/B$$

$$B \rightarrow A/11$$

$$A \rightarrow 0/12/B$$

Non-unit prodⁿ

$$S \rightarrow A0$$

$$B \rightarrow 11$$

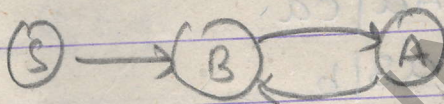
$$A \rightarrow 0|12$$

Unit prodⁿ

$$S \rightarrow B$$

$$B \rightarrow A$$

$$A \rightarrow B$$



It is clear from the dependence graph that $S \xRightarrow{*} B$, $S \xRightarrow{*} A$, $B \xRightarrow{*} A$ & $A \xRightarrow{*} B$ so the new prodⁿ for $S, A \& B$ are

$$S \rightarrow A0|11|0|12$$

$$B \rightarrow 0|12|11$$

$$A \rightarrow 0|12|11$$

The resulting grammar without unit prodⁿ

$$G' = (V', T', P', S)$$

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

$$T' = \{0, 1, 2\}$$

$$P' = \{ S \rightarrow A0|11|0|12,$$

$$B \rightarrow 0|12|11,$$

$$A \rightarrow 0|12|11 \}$$

S is the start symbol.

(Eq)

eliminate unit prodⁿ from the grammar

$$S \rightarrow Aa|B|Ca$$

$$B \rightarrow aB|b$$

$$C \rightarrow Db|D$$

$$D \rightarrow E|d$$

$$E \rightarrow ab$$

Solⁿ:

Nm-unit prodⁿ

$$S \rightarrow Aa|Ca$$

$$B \rightarrow aB|b$$

$$C \rightarrow Db$$

$$D \rightarrow d$$

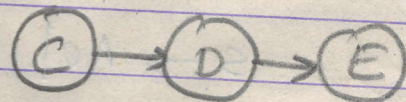
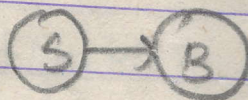
$$E \rightarrow ab$$

Unit prodⁿ

$$S \rightarrow B$$

$$C \rightarrow D$$

$$D \rightarrow E$$



$$S \xRightarrow{*} B$$

$$S \rightarrow Aa|aB|b|Ca$$

$$~~C \rightarrow D~~ \quad D \xRightarrow{*} E$$

$$D \rightarrow ab|d$$

$$c \xrightarrow{*} D \quad D \xrightarrow{*} E$$

$$c \rightarrow Db | ab | d$$

The resulting grammar

$$G' = (V', T', P', S)$$

$$V' = \{ S, A, B, C, D, E \}$$

$$T' = \{ a, b, d \}$$

$$P' = \{$$

$$S \rightarrow Aa | aB | b | ca$$

$$B \rightarrow aB | b$$

$$C \rightarrow Db | ab | d$$

$$D \rightarrow d | ab$$

$$E \rightarrow ab \}$$

S is the start symbol.

Chomsky Normal form.

Every nonempty CFL with ϵ has a grammar G in which all productions are in one of two forms, either

- 1) $A \rightarrow BC$, where A, B & C are variable
- 2) $A \rightarrow a$, where A is a variable & a is a terminal

Further, G has no useless symbols. Such a grammar is said to be in Chomsky Normal form, or CNF.

(Eg) Convert the following grammar to CNF

- ① Remove ϵ -prod
- ② Remove unit-p
- ③ Remove useless

$$S \rightarrow 0A \mid 1B$$

$$A \rightarrow 0AA \mid 1S \mid 1$$

$$B \rightarrow 1BB \mid 0S \mid 0$$

Soln

Std form

$$A \rightarrow 1$$

$$B \rightarrow 0$$

$$S \rightarrow 0A \mid 1B$$

$$S \rightarrow B_0 A \mid B_1 B$$

$$B_0 \rightarrow 0$$

$$B_1 \rightarrow 1$$

$$A \rightarrow B_0 AA \mid B_1 S \mid 1$$

$$B_0 \rightarrow 0$$

$$B_1 \rightarrow 1$$

$$B \rightarrow B_1 BB \mid B_0 S$$

$$B_0 \rightarrow 0$$

$$B_1 \rightarrow 1$$

Grammar obtained

$$V_1 = \{S, A, B, B_0, B_1\}$$

$$T_1 = \{0, 1\}$$

$$P_1 = \{ S \rightarrow B_0 A \mid B_1 B, A \rightarrow B_0 AA \mid B_1 S \mid 1, B \rightarrow B_1 BB \mid B_0 S$$

$$B_0 \rightarrow 0$$

$$B_1 \rightarrow 1 \}$$

S is the start symbol.

$$S \rightarrow B_0 A \mid B_1 B$$

$$A \rightarrow B_0 A A \mid B_1 S \mid 1$$

$$B \rightarrow B_1 B B \mid B_0 S \mid 0$$

$$B_0 \rightarrow 0$$

$$B_1 \rightarrow 1$$

CNF prodⁿ are

$$S \rightarrow B_0 A \mid B_1 S$$

$$A \rightarrow B_1 S \mid 1$$

$$B \rightarrow B_0 S \mid 0$$

$$B_0 \rightarrow 0$$

$$B_1 \rightarrow 1$$

productions which are not in CNF form.

$$A \rightarrow B_0 A A$$

$$B \rightarrow B_1 B B$$

$$A \rightarrow B_0 A A, A \rightarrow B_0 D_1$$

$$D_1 = A A$$

$$~~B \rightarrow B_1 B B~~ B \rightarrow B_1 B B$$

$$B \rightarrow B_1 D_2$$

$$D_2 \Rightarrow B B$$

$$G = \{S, A, B, B_0, B_1, D_1, D_2\}$$

$$T_1 = \{0, 1\}$$

$$P_1 = \{S \rightarrow B_0 A \mid B_1 S, A \rightarrow B_0 D_1 \mid B_1 S \mid 1, B \rightarrow B_1 D_2 \mid B_0 S \mid 0\}$$

$$B_0 \rightarrow 0, D_1 \rightarrow A A$$

$$B_1 \rightarrow 1, D_2 \rightarrow B B \}$$

S is the start symbol

(Eq) Convert the following grammar to CNF

$$I \rightarrow C|d|I_a|I_b|I_o|I_i$$

$$F \rightarrow I|(E)$$

$$T \rightarrow F|T * F$$

$$E \rightarrow T|E + T$$

(Eq) Find a grammar equivalent to

$$S \rightarrow AB|CA$$

$$A \rightarrow a$$

$$B \rightarrow BC|AB$$

$$C \rightarrow aB|b$$

with no useless symbols

(Eq) Begin with the grammar

$$S \rightarrow ASB|E$$

$$A \rightarrow aAS|a$$

$$B \rightarrow SbS|A|bb$$

(a) Are there any useless symbols? Eliminate them if so.

(b) Eliminate ϵ -productions

(c) Eliminate unit-productions

(d) put the grammar into CNF.

Greibach Normal Form (GNF).

In GNF there is no restriction on the number of symbols on the right hand side, but there is restriction on the terminals & variables appear on the right hand side of the prodⁿ.

Defⁿ: Let $G = (V, \Sigma, P, S)$ be a CFG. The CFG G is said to be in GNF if all the productions are of the form

$$A \rightarrow a\alpha$$

where $a \in \Sigma$ and $\alpha \in V^*$, i.e. the first symbol on the right hand side of the prodⁿ must be a terminal & it can be followed by zero or more variables.

eg

Convert the following grammar to GNF.

$$S \rightarrow AB1 \mid 0$$

$$A \rightarrow 00A \mid 1A1$$

$$B \rightarrow 1A1$$

$$S \rightarrow ABA_1 \mid 0$$

$$A \rightarrow A_0 A_0 A \mid A_1 A A_1$$

$$B \rightarrow A_1 A A_1$$

$$A_1 \rightarrow 1$$

$$A_0 \rightarrow 0$$

Now restrict the no of variables on the right hand side of the prodⁿ to two & the resulting grammar in CNF notation is

$$S \rightarrow AD_1 | 0$$

$$A \rightarrow A_0 D_2 | A_1 D_3$$

$$B \rightarrow A_1 D_3$$

$$A_1 \rightarrow 1$$

$$A_0 \rightarrow 0$$

$$D_1 \rightarrow BA_1$$

$$D_2 \rightarrow A_0 A$$

$$D_3 \rightarrow AA_1$$

Now let us rename all the variables as shown below:

$$\text{Let } S = A_1, A = A_2, B = A_3, A_0 = A_4, A_1 = A_5,$$

$$D_1 = A_6, D_2 = A_7, D_3 = A_8$$

Now the grammar can be rewritten as:

$$A_1 \rightarrow A_2 A_6 | 0$$

$$A_2 \rightarrow A_4 A_7 | A_5 A_8$$

$$A_3 \rightarrow A_5 A_8$$

$$A_5 \rightarrow 1$$

$$A_4 \rightarrow 0$$

$$A_6 \rightarrow A_3 A_5$$

$$A_7 \rightarrow A_4 A_2$$

$$A_8 \rightarrow A_2 A_5$$

In the above prodⁿ, note that A_4 & A_5 -prodⁿ are in GNF

Consider A_3 prodⁿ: Substituting for A_5 in A_3 -prodⁿ we get

$$A_3 \rightarrow A_5 A_2 = 1 A_2 \rightarrow \text{GNF.}$$

Consider A_2 prodⁿ

$$A_2 \rightarrow A_4 A_7 \mid A_5 A_8 = 0 A_7 \mid 1 A_8 \rightarrow \text{GNF.}$$

Consider A_1 prodⁿ

$$A_1 \rightarrow A_2 A_6 \mid 0 = (0 A_7 \mid 1 A_8) A_6 \mid 0 = 0 A_7 A_6 \mid 1 A_8 A_6 \mid 0 \rightarrow \text{GNF}$$

Consider A_6 prodⁿ

$$A_6 \rightarrow A_3 A_5 = \cancel{1 A_8} (A_5 A_2) A_5$$

$$A_6 \rightarrow A_5 A_8 A_5 = 1 A_8 A_5 \rightarrow \text{GNF}$$

Consider A_7 prodⁿ

$$A_7 \rightarrow A_4 A_2 = 0 A_2 \rightarrow \text{GNF.}$$

Consider A_8 prodⁿ

$$\begin{aligned} A_8 \rightarrow A_2 A_5 &= (0 A_7 \mid 1 A_8) A_5 \\ &= 0 A_7 A_5 \mid 1 A_8 A_5 \rightarrow \text{GNF} \end{aligned}$$

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

$$V = \{A_1, A_2, A_3, A_4, A_5, A_6, A_7, A_8\}$$

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

$$P = \{$$

$$A_1 \rightarrow 0A_7A_6 \mid 1A_8A_6 \mid 0$$

$$A_2 \rightarrow 0A_7 \mid 1A_8$$

$$A_3 \rightarrow 1A_8$$

$$A_4 \rightarrow 0$$

$$A_5 \rightarrow 1$$

$$A_6 \rightarrow 1A_8A_5$$

$$A_7 \rightarrow 0A_2$$

$$A_8 \rightarrow 0A_7A_5 \mid 1A_8A_5$$

}

S is start symbol.

(Eg)

Convert the following grammar

$$A \rightarrow BC$$

$$B \rightarrow CA \mid b$$

$$C \rightarrow AB \mid a$$

into CNF.