· Ques (i) $(S^+)^* = (S^*)^*$

Ans $(S^+)^*$ includes $\wedge$ even is S doesnot.

so, $(S^+)^* = S^*$ and $(S^*)^* = S^*$ by Theorem 1

(ii) $(S^+)^+ = S^+$

Ans There can be no factor in $(S^+)^+$ that is not in S^+

$(S^+)^+ \subseteq S^+$ ——①

In general, any set is contained in its positive closure

$S^+ \subseteq (S^+)^+$ ——②

Therefore $(S^+)^+ = S^+$

(iii) Is $(S^*)^+ = (S^+)^*$ for all sets S?

Ans Yes. If $\wedge \in S$ then $S^* = S^+$

$(S^+)^* = (S^*)^+ = (S^*)^+$

If $\wedge \notin S$ then $\wedge \in S^*$ anyway, and

$(S^*)^+ = (S^*)^* = S^* = S^+ \cup \wedge = (S^+)^*$

case 1 When $\wedge \in S$ then $S^* = S^+$

Take Kleene's star on both sides $(S^*)^* = (S^+)^*$ ——①

Also, we know $(S^*)^* = S^*$ ——②

so, using ① and ② $(S^+)^* = S^*$ ~~then~~ $= (S^*)^+$

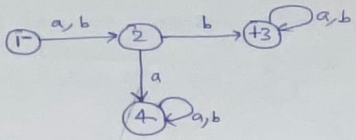
Case 2 When $\wedge \notin S$, then $\wedge \in S^*$ anyway and

$(S^*)^+ = (S^*)^* = S^* = S^+ \cup \wedge$

$= (S^+)^*$

Book

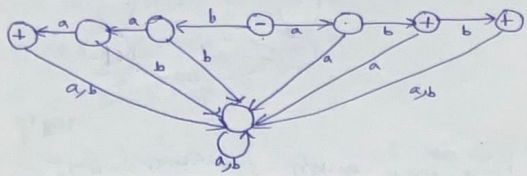
Ques2



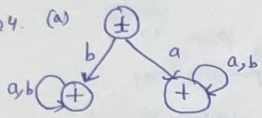
$(a+b)\,b\,(a+b)^*$

	a	b
1⁻	2	2
2	4	3
3⁺	3	3
4	4	4

Ques3

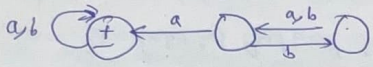


Ques4. (a)

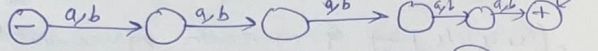


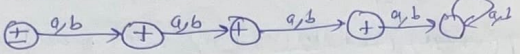
(b) If all the states in FA are final then automatically all string are accepted because no matter where the string ends its accepted

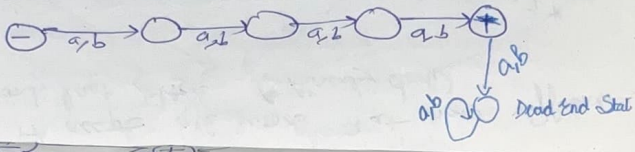
(c) Not necessarily, for the non-plus states may be unreachable as in the following FA

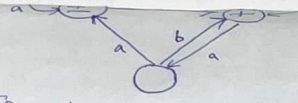


Ques5

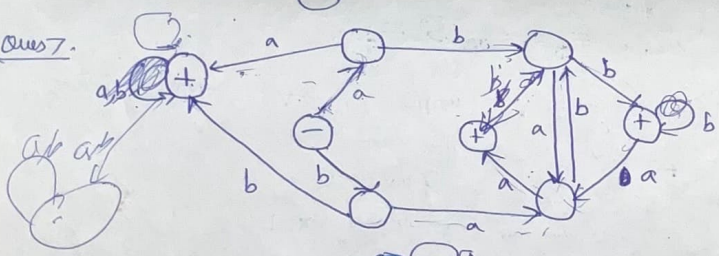
(i) 

(ii) 

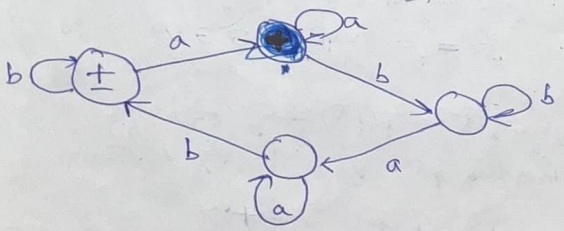
(iii) 



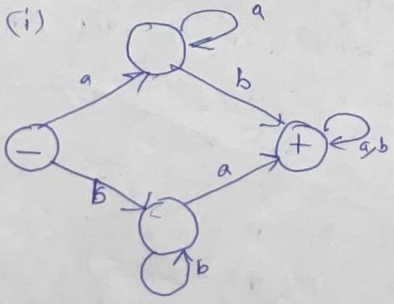
Ques7.



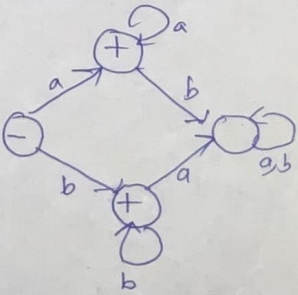
Ques8
Exam



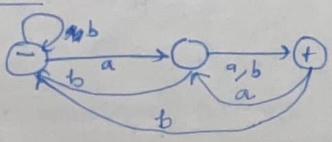
Ques9. (i)



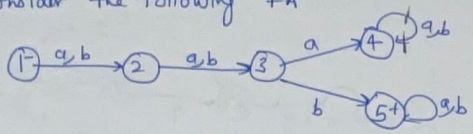
(ii) $aa^* + bb^*$



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



Ques Consider the following FA



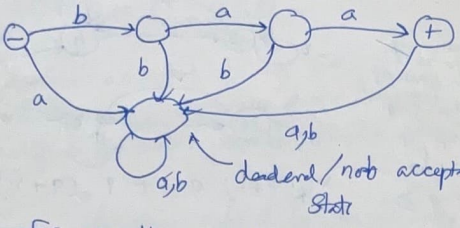
This machine accepts all words with b as the third letter and rejects all words other than this. This language defined by regular expression

$$(aab + abb + bab + bbb)(a+b)^*$$

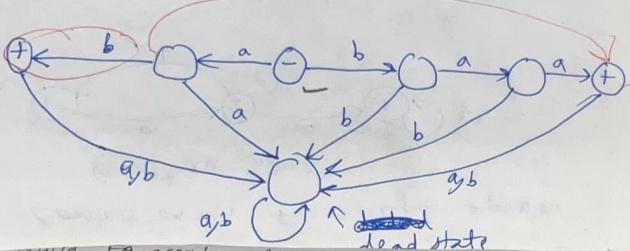
$$\Rightarrow (a+b)(a+b)\,b\,(a+b)^*$$

$$\Rightarrow (a+b)^2\,b\,(a+b)^*$$

Ques Consider specialized FA, that accepts only ~~string~~ baa



Ques FA for accepting exactly the two string baa and ab



~~Such~~ FA accepts all words that have different first and last letters (Already done)

EVEN-EVEN

Let $E = [aa + bb + (ab+ba)(aa+bb)^*(ab+ba)]^*$

Regular expression can be represented by $E = [r_1 + r_2 + r_3]^*$

where $r_1 = aa$, $r_2 = bb$

$$r_3 = (ab+ba)(aa+bb)^*(ab+ba)$$

λ_1: double a (a's are even)

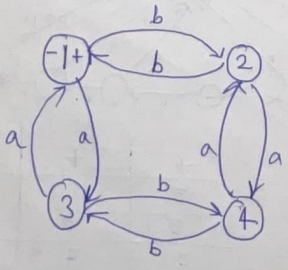
λ_2: double b (b's are even)

λ_3: (a) first choice from (ab + ba) single a and single b

(b) second " " $(aa+bb)^*$ (even no. of a's and b's)

(c) third " " (ab + ba) (single a and single b)

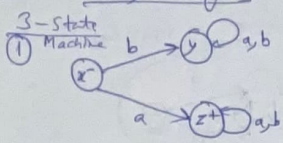
So language says, all words must have even no. of letters.



Ques Build FA that accepts all words in a language $a(a+b)^*$

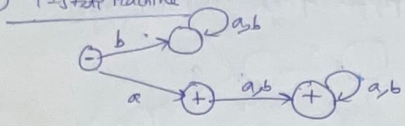
Ans Many ways

(1) 3-State Machine



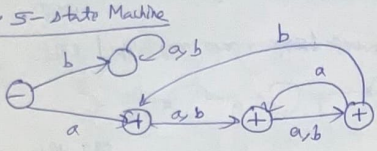
dead-end state is that state where no strings can leave once entered.

(2) 4-State Machine



i.e only word a ends with first + state and other words starting with a reach and finish 2nd + state where they are accepted.

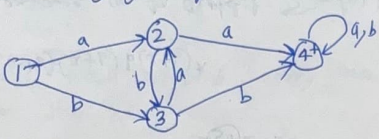
(3) by 5-state Machine



~~Note~~

Note: Every language accepted by an FA can be defined by regular expression, conversly every language that can be defined by regular expression can be accepted by some FA.

Ques Find the language defined by the following FA:

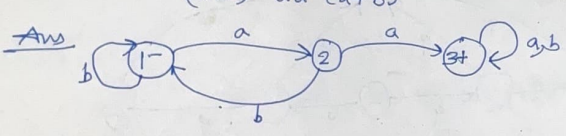


Ans $(a+b)^*(aa+bb)(a+b)^*$

Ques Construct an FA for the regular expression

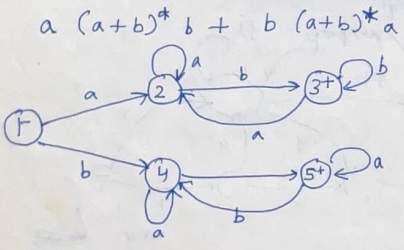
$(a+b)^*aa(a+b)^*$

Ans



Ques Construct an FA that have different first and last letter

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



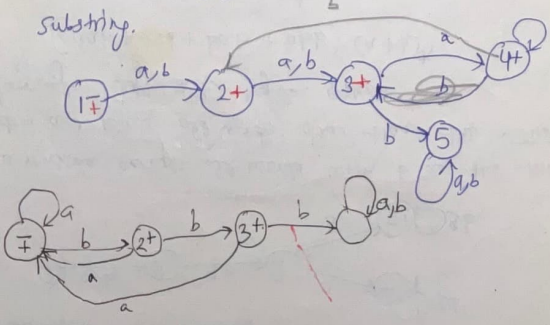
If the 2 lg. separated by + sign in e.c. are disjoint, then there should be 2 separate final states for them.

6343

Ques (5 marks) Construct an FA accepting the language L where w string belongs to:

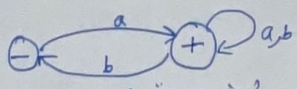
$w \in \{a,b\}^*$, w doesn't contain bbb as substring.

Ans



Non-deterministic Automata

Ques

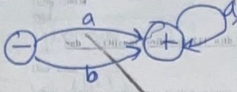


Find the regular expression?

Ans $L = \{a, aa, ab, aaa, aab, \dots\}$

L = {set of all strings of a's and b's over an alphabet that all strings begins with a}

Regular Expression $r = a(a+b)^*$



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

$= (a+b)^+$

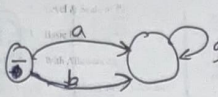
Ques

Find the regular expression?

$r = (a+b)^*$

$L = \{\wedge, a, b, aa, ab, bb, ba, \dots\}$

e.g

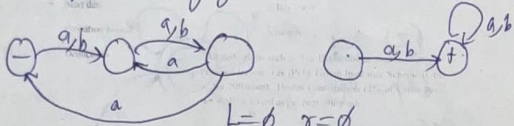


$L = \phi$

$r = \phi$

Since there is no final state. Therefore, it doesn't define any language.

eg

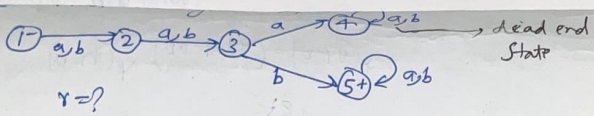


$L = \phi$ $r = \phi$

Since the graph is disconnected.

Dead End State

Any state which we can't leave, once we make it to that state.

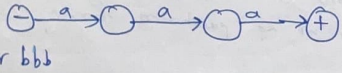


$r = ?$

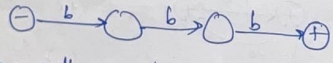
Ans $r = (a+b)(a+b)\,b\,(a+b)^*$

Ques Given a language on an alphabet $\Sigma = \{a, b\}$ which only accepts either aaa or bbb and only those words. Build an FA.

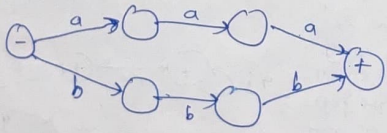
Ans To build simple word aaa.



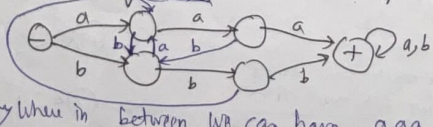
for bbb



Two path can share ~~same~~ same final and Initial state



Then, any string can be added, for simple



Anywhere in between we can have aaa / bbb for that add a-path / b-path as required

Find out if string 11010 is accepted by this FA or not?

Ans Initial State q_0

Current state q_0 Input letter 11010

q_0, 11010
q_1, 1010
q_3, 010
q_2, 10

q_3, 0
q_1, 1
q_0, ∧

Since $q_0 \in F$, since q_0 is final state, the string is accepted.

Is the string is acceptable by the above FA?
111100001010

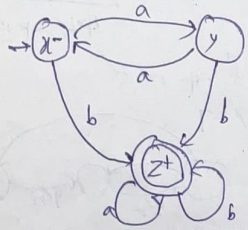
q_0, 111100001010
q_1, 11100001010
q_0, 1100001010
q_1, 100001010
q_0, 00001010
q_2, 0001010

q_0, 0010100
q_2, 010100
q_0, 10100
q_1, 0100
q_3, 100
q_2, 00
q_0, 0
q_2, ∧

Since q_2 is not the final state. Hence it is not acceptable.

Pictorial depiction of finite automata is called transition graph. It is a finite directed labelled graph in which each vertex represents a state and the directed edges indicate the transition from one state to the next when any input is applied.

Example



$Q_1 = \{x, y, z\}$
$\Sigma = \{a, b\}$
$q_0 = \{x\}$
$F = \{z\}$

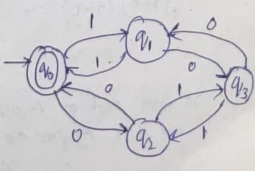
State	a	b
→ x	y	z
y	x	z
(z)	z	z

Ques Draw the transition Graph for the following:

$Q = \{q_0, q_1, q_2, q_3\}$ $q_0 = \{q_0\}$
$\Sigma = \{0, 1\}$ $F = \{q_0\}$

Transition table

State	I/p 0	1
→ q_0	q_2	q_1
q_1	q_3	q_0
q_2	q_0	q_3
q_3	q_1	q_2



Exam Questions

Ques1 Define the Kleene's closure with an example.

Ques2 Consider the language having set
$S=\{a, ab, ba\}$
Is string "abbba" a word of language

Ans No because by clubbing any combination of this finite language, we won't be able to get this string.

Ques3 find the regular expression over the alphabet $\Sigma=\{0,1\}$ to describe the set of all binary numerals without leading 0's (except 0 itself)

Ans ~~$r=0+1(0+1)^*$~~
$r=0+1(0+1)^*$

Ques4. construct the regular expression over an alphabet $\Sigma=\{a,b\}$
(i) all words that doesn't have the substring ab
(ii) all words that end with a double letter.

Ans (i) b^*a^*
(ii) $(a+b)^*(aa+bb)$
or $(a+b)^*aa + (a+b)^*bb$

CHAPTER

FINITE AUTOMATA

ABSTRACT DEFINITION OF AN FA

1) A finite set of states $\Theta_1=\{q_0, q_1, q_2, \ldots\}$ of which q_0 is the start state. $\longrightarrow q_0 \in \Theta_1$
2) A subset of Θ_1 called the final states. $\longrightarrow F \subseteq \Theta_1$
3) An alphabet $\Sigma=\{x_1, x_2, x_3 \ldots\}$
4) A transition function δ associated with each pair of state and letter with a state.

$$\delta(q_i, x_j) = q_K$$

q_K ↳ next state $\longrightarrow \delta : \Theta_1 \times \Sigma \to \Theta_1$,

δ is a transition function whose input is pair of state and alphabet letter and whose output is single state

Example Consider FA where

$\Theta_1=\{q_0, q_1, q_2, q_3\}$
$\Sigma=\{0,1\}$ $F=\{q_0\}$

Transition table δ is given as follows:

Transition States	Input 0	Input 1
→ q_0	q_2	q_1
q_1	q_3	q_0
q_2	q_0	q_3
q_3	q_1	q_2

Then, by using identity (ix) we get

$r = (r_1)^*$

by putting back again the value of r_1 we get

$r = (1^*(011)^*)^*$

$= (1+011)^*$ (by using identity (xi))

Therefore,

$L = \{$ any zero will be succeeded by two number of 1's$\}$

$= \{$ set of strings of 0's and 1's such that each 0 is followed by atleast two 1's$\}$

Formal definition of Regular ~~Expression~~ Language

Regular language is a language associated with any regular expressions.

(i) The language associated with the regular expression that is just a single letter is the one letter word alone i.e if the regular expression is $\wedge$

then $L = \text{Language}\{\wedge\}$

(ii) If r_1 is a r.e associated with Language L_1 and r_2 is r.e associated with Language L_2 then

(a) the regular expression $r_1 r_2$ is associated with language $L_1 \cdot L_2$

$\text{Lang}(r_1 \cdot r_2) = L_1 \cdot L_2$

(b) $\text{lang}(r_1 + r_2) = L_1 + L_2$

(c) $\text{Lang}(r_1^*) = L_1^*$

Ques Consider the regular expression.

$E = (a+b)^* a (a+b)^* (a+\wedge) (a+b)^* a (a+b)^*$

Give the simplified language. $L = ?$

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

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

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

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

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

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

$=$ {atleast 3 a's} + {atleast 2 a's}

$=$ {atleast 2 a's}

Therefore, $L = \{$ string of a's and b's that have atleast two a's.$\}$

Ques. Consider the regular expression

$E = [aa + bb + (ab+ba)(aa+bb)^*(ab+ba)]^*$

Give the corresponding language.

Ans. Rewrite E as follows:

$E = [r_1 + r_2 + r_3]^*$

where, $r_1 = aa$, $r_2 = bb$, $r_3 = (ab+ba)(aa+bb)(ab+ba)$

$\text{lang}(r_1) = \text{lang}\{$ contains even no. of a's$\}$

$\text{lang}(r_2) = \text{Lang}\{$ contains even no. of b's$\}$

$\text{Lang}(r_3) = \text{Lang}\{$ contains even no. of a's and b's$\}$

Therefore, $\text{Lang}\{E\} = \{$ set of all strings of a's and b's in which there is even number of a's and even number of b's$\}$

Example. The language of all words that have atleast one a and atleast one b.

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

$$= (\text{arbitrary}) a (\text{arbitrary}) b (\text{arbitrary}) + (\text{arbitrary}) b (\text{arbitrary}) a (\text{arbitrary})$$

distribution (or) $(a+b)^* (ab+ba)(a+b)^*$

$$= a a^* b b^* + b b^* a a^*$$

Note: Every regular expression defines some language to mean that the associated language has a simple English description.

Freq $(\wedge + ba^*)(ab^*a + ba^*) b (a^* + b^* a) bab^*$ probably has no cute concise alternate characterization.

Note: In algebra, $ab = ba$ but in formal languages $ab \neq ba$.

Definition

If S and T are sets of strings of letters, we define the product set

$ST = \{$all combinations of a string from S concatenated with a string from T in that order$\}$

Freq If $S = \{a, aa, aaa\}$, $T = \{bb, bbb\}$

then $ST = \{abb, abbb, aabb, aabbb, aaabb, aaabbb\}$

Ques. If $S = \{a, bb, bab\}$ $T = \{a, ab\}$

$ST = \{aa, aab, bba, bbab, baba, babab\}$

Ques If $P = \{a, bb, bab\}$ $Q = \{\wedge, bbb\}$

$PQ = \{a, bb, bab, abbb, bbbbb, babbbb\}$

Note If L is any language, then

$L\wedge = \wedge L = L$

Identities for regular expressions

(i) $\phi + r = r$

(ii) $\phi r = r\phi = \phi$

(iii) $\wedge r = r\wedge = r$

(iv) $\wedge^* = \wedge$, $\phi^* = \wedge$

(v) $r + r = r$

(vi) $r^* r^* = r^*$

(vii) $rr^* = r^* r$

(viii) $(r^*)^* = r^*$

(ix) $\wedge + rr^* = r^*$

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

(xi) $(P+Q)^* = (P^* \cdot Q^*)^* = (P^* + Q^*)^*$

Ques Using above identities, simplify the given expression.

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

$r = \wedge + 1^* (011)^* (1^* (011)^*)^*$

Ans Consider the regular expression

$r = \wedge + 1^* (011)^* (1^* (011)^*)^*$

Put $r_1 = 1^* (011)^*$

Then r can be written as

$r = \wedge + r_1 r_1^*$

Regular Expressions

are a means of representing certain set of strings over an alphabet Σ in an algebraic fashion

Recursive definition of Regular Expressions:-

Let Σ be given alphabet. Then:

(i) $\emptyset$, $\wedge$ are primitive regular expression

(ii) If r_1 and r_2 are regular expressions. Then so are:
- (a) $r_1 + r_2$
- (b) $r_1 . r_2$
- (c) r_1^*

(iii) A string is a regular expression iff it can be derived from primitive regular expressions by finite application of rules in (ii)

Regular expression	Regular Set
a	{a}
a+b	{a, b}
ab	{ab}
a^*	{∧, a, aa, aaa, ...}

Ques Give the regular expressions for the following regular set -

(i) {101} → 101

(ii) {abba} → abba

(iii) {10, 01} → 10 + 01

(iv) {∧, ab} → ∧ + ab

(v) {∧, 0, 00, 000, ...} → 0^*

(vi) {1, 11, 111, 1...} → 1^+ or 1^*1

Ques Consider the language

$L = \{x^{odd}\} = \{x, xxx, xxxxx, ...\}$

Give the regular expression.

Ans The regular expression can be either defined:

(a) $x(xx)^*$ or,

(b) $(xx)^* x$

but it can't be expressed as

$x^*(x)x^*$, since it will give even number of x.

Example The language of all words that have atleast two a's can be described by the expression

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

= (some beginning) (the first important a) (some middle) (the second important a) (some end)

where the arbitrary parts can have as many a's (or b's) as they want.

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

or = Language $(b^* a b^* a (a+b)^*)$ → We scan through some jungle of b's (or no b's) until we find the first a then more b's (or no b's), then the second a, then we finish up with anything.

or = Language $((a+b)^* a b^* a b^*)$ (last a, next to last a)

or = Language $(b^* a (a+b)^* a b^*)$ (first a, last a)

But, if we wanted all the words with exactly two a's, we could use the expression $b^* a b^* a b^*$

(ii) Let $S = \{ab, bb\}$ and $T = \{ab, bb, bbb\}$. Show that $S^* \neq T^*$, but that $S^* \subset T^*$.

Ans $S \subset T$, so $S^* \subset T^*$. However, there is no way to generate bbb with the elements of S. - so $S^* \neq T^*$

Note: $S^* = T^*$ even though $S \neq T$ only when all the words in the symmetric difference can be generated by words in the intersection.

CHAPTER 4
REGULAR EXPRESSIONS

The language defined by the expression ab^*a is the set of all strings of a's and b's that have atleast two letters, that begin and end with a's, and that have nothing but b's inside (if anything at all)

Language $(ab^*a) = \{aa, aba, abba, abbba, abbbba, \dots\}$

The language of the expression

a^*b^*

contains all the strings of a's and b's in which all the a's (if any) come before all the b's (if any)

Language $(a^*b^*) = \{\Lambda, a, b, aa, ab, bb, aaa, aab, abb, bbb, aaaa, \dots\}$

Notice ba and aba are not in the language. There need not be the same no. of a's and b's.

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

↓ Doesn't contain abab ↓ Contains abab

Example Consider the language T defined over the alphabet

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

$T = \{a, c, ab, cb, abb, cbb, abbb, cbbb, abbbb, cbbbb, \dots\}$

All the words in T begin with an a or a c then followed by some number of b's.

Infinite

Symbolically, T = language $((a+c)b^*)$

= language (either a or c then Zero or more some b's)

↓ Some always means "some or no"

Example Let us consider a finite language L that contains all the strings of a's and b's of length three exactly

$L = \{aaa, aab, aba, abb, baa, bab, bba, bbb\}$

Ans. L = language $((a+b)(a+b)(a+b))$

= Language $((a+b)^3)$

$a(a+b)^*b$ → All words that begin with an 'a' and end with a 'b'.

Positive Closure '+'

1) S is the strings that doesn't include ^ then
$S^+ = S^*$

2) S is the strings that does contain ^ then
$S^+ = S^* - \{\wedge\}$

$S = \{a, b\}$
$S^* = \{\wedge, a, b, ab, ba, aa, bb, \ldots\}$
$S^{**} = \{\wedge, a, b, ab, ba, aa, bb, \ldots\}$

Ex: $S = \{a, b\}$
$S^* = \{\wedge, a, b, aa, bb, \ldots\}$
$S^+ = \{a, b, aa, bb, \ldots\}$

Theorem topic: $(S^*)^* = S^*$

Proof: $(S^*)^*$ is made up of all the factors made up of S^*
S^* is made up of all the factors made up of S.
Therefore, transitively $(S^*)^*$ is made up of all the factors made of S
Hence, $(S^*)^* \subseteq S^*$ -- (1)

for any set A,
$A \subset A^*$
Replace A by S^*
$S^* \subseteq (S^*)^*$ --- (2)

from (1) and (2)
$(S^*)^* = S^*$

Theory of Formal Language

The word "formal" refers to the fact that all the rules for the language are explicitly stated in terms of what strings of symbols can occur.
The term "formal" used here emphasizes that it is the form of the string of symbols we are interested in, not the meaning.

$(S^+)^*$ includes ^ even if S doesn't, so $(S^+)^* = S^*$ and $S^* = S^{**}$ by theorem 1

Exercise.

Ques consider the language S^*, where $S = \{a, b\}$
How many words does this language have of length 2? of length 3? of length n?

Ans S^* has 4 2-letter words
8 3-letter words
2^n n-letter words

Ques Consider the language S^*, where $S = \{aa, b\}$
How many words does this language have of length 4? of length 5? of length 6? What can be said in general?

Ans. $S^* = \{\wedge, aa, b, aab, baa, aabb, bbaa, baab, aaaab, bbaaa, baaaa, bbbbb, aabaa, aabbb, \ldots\}$

5 words of length 4; 8 words of length 5; 13 words of length 6. In general, the number of words of length n is the number of words of length (n-1) and the number of words of length (n-2), a fibonacci sequence

Ques (i) Let $S = \{ab, bb\}$ and let $T = \{ab, bb, bbbb\}$
Show that $S^* = T^*$.

Ans $S \subset T$, so $S^* \subset T^*$. bbbb is the only word in T but not in S. However, bb $\in S$ so bbbb $\in S^*$ and $T^* \subset S^*$. Therefore $S^* = T^*$

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

Define a language called PALINDROME.

$= \{x$ such ~~such~~ that $reverse(x) = x\}$ (Set Builder Form)

$= \{a, b, aa, bb, bab, aba, abba, baab, \ldots\}$

Empty String / Null string — $\wedge$ or ϵ

A string with zero occurrences of symbols

Length $|w|$ of string w : The number of symbols in w

eg $|0111| = 4$ $\quad |\wedge| = 0$

Power of an alphabet Σ^k

a set of all strings of length k

Given $\Sigma = \{0, 1\}$, we have

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

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

$\Sigma^2 = \{00, 01, 10, 11\}$

and so on.

Set of All strings over Σ → denoted as Σ^*

$\Sigma^* = \Sigma^0 \cup \Sigma^1 \cup \Sigma^2 \cup \ldots$

Similarly, Σ^+ = set of Non-empty strings from Σ

$\Sigma^+ = \Sigma^* - \{\wedge\}$

ϕ = Empty language (not same as empty string $\wedge$)

$L \neq L + \{\wedge\}$ $\quad L = \{ab, aa, bb, \ldots\}$

and $L = L + \phi$ $\quad L' = \{\wedge, a, b, aa, bb, \ldots\}$

L and L' are not equal

KLEENE's CLOSURE

Given an alphabet Σ we define a language in which any string of letters from Σ is a word even the null string. This language is called the closure of the language and is denoted by the symbol Σ^*

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

$\Sigma^* = \{\wedge, 0, 1, 00, 01, 10, 11, 000, 001, 010, \ldots\}$

This operation makes an infinite language of string of letters from the alphabet

eg $S = \{aa, b\}$

$S^* = ?$

$S^* = \{\wedge, aa, b, aab, baa, aaaab, baab, aabaa, bbaa, \ldots\}$

$= \{\wedge$ plus words/strings of a's and b's where a's occur in even clumps$\}$

Intuition

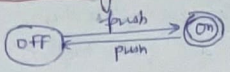
eg $S = \{a, ab\}$

$S^* = ?$ $\quad S = \{\wedge, a, ab, aab, aba, \ldots\}$

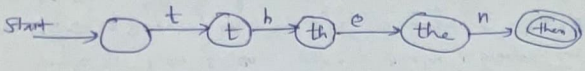
$= \{\wedge$ plus all the strings of a's and b's that can't begin with b and b's can't occur twice together$\}$

Why to Study Automata theory?

1) An FA modeling an on/off switch



2) An FA modeling recognition of the keyword "then" in a lexical analyzer.



→ final or accepting state

GRAMMAR Processing data with a recursive structure.

Eg Production rule $E \to E+E$ for generating arithmetic expressions.

REGULAR EXPRESSIONS Describing text strings like $[A-Z][a-z]^*$

3) Pattern Maching can be used in Twitter Sentiment analysis

4) A finite state machine is an FA together with actions on the arcs. Semantics

5) Lexical Analyzer

Finite Automata :- Another Method for defining language.

Finite:- Number of possible states and number of letters in the word are both finite.

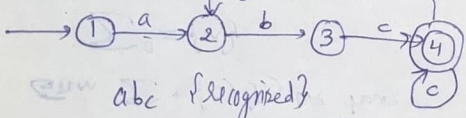
A FA is a collection of 3 things:-

1) A finite set of states [position of machine at a particular time] one of which is designated as Initial state / start state, and some are final state.
2) An alphabet of possible input letters.
3) A finite set of transitions that tell for each

It works by presenting input string of letters that it reads letter by letter starting at leftmost letter. Beginning from start state, the letter determine the sequence of states. It stops when there is no transition from its current state that matches the next character to be scanned.

If the automata stops in a final state, we say that it recognizes (or accepts) the string being scanned.

If it stops in a non-final state, it fails to recognize (or rejects) the string.



abc {Recognized}
abccabc {Recognized}
ac {Not recognized}

State	a	b	c
1	2	—	—
2	—	3	—
3	—	—	4
4	2	—	4

Definition of the GRAMMAR:

$G = (V_n, \Sigma, P, S)$

V_n = finite Non-empty set of Non-Terminals

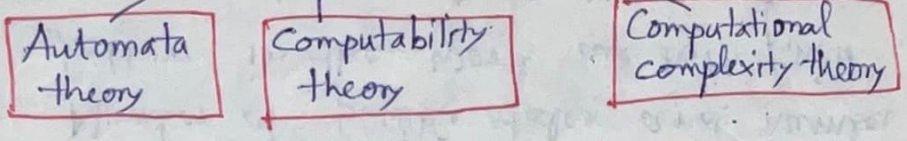
Σ = finite non-empty set of letter called Terminals. It is known as alphabet

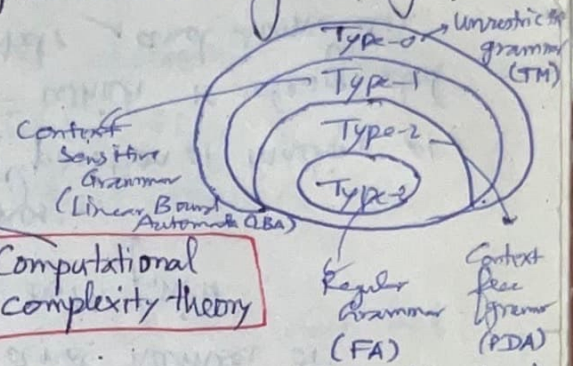
P = set of production rules

CS THEORY OF COMPUTATION

It is the branch of theoretical computer Science and mathematics that deals with how efficiently problems can be solved on a model of computation using an algorithm.

This Field is further divided into:





One popular model which is commonly used is Turing Machine (TM)

In principle, any problem that can be solved (decided) by a TM can be solved by a computer that has a bounded amount of memory.

Some pioneers of TOC are Alonzo church, Alan Turing, Stephen Kleene, John Von Neumann.

① Automata Theory

Automata Theory is the study of Abstract machines and the Computational problems that can be solved using these machines. These abstract machines are called Automata. It means, that something is doing something by itself.

③ Computational Complexity theory

complexity theory considers not only whether a problem can be solved at all on a computer, but also how efficiently the problem can be solved. Time complexity and space complexity expressed in Big-Oh Notation

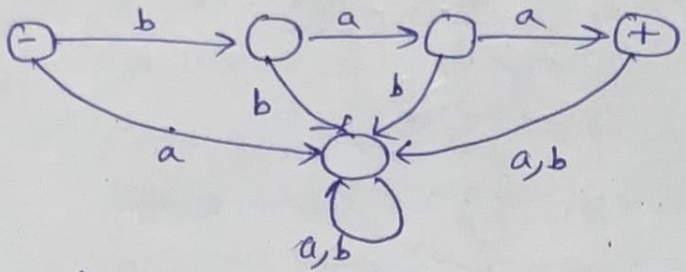
② Computability Theory

It deals with the question of the extent to which a problem is solvable on a computer. The statement that the halting problem can't be solved using Turing Machine is the result of this theory

Halting problem Given a computer program, decide whether the program finishes running or continues to run forever.

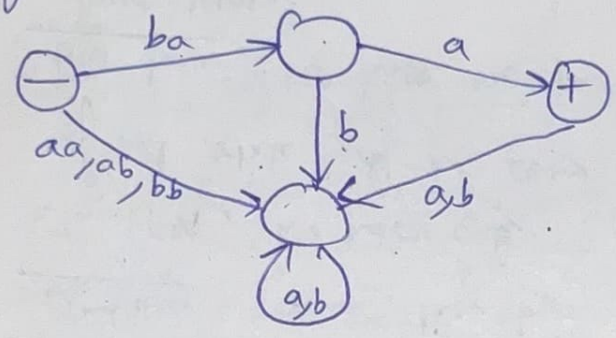
CHAPTER-6 TRANSITION GRAPHS

FA that accepts only the word baa

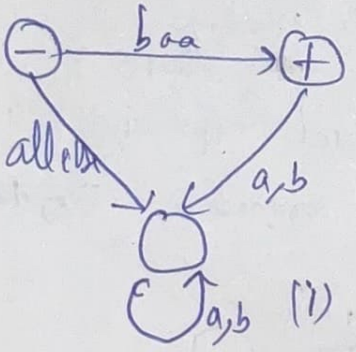


requires only 5 states because an FA can read only one letter from the input string at a time

we can design a more powerful machine that could read either ~~one or two~~ one or two letters of input string at a time and could change its state based on this input information

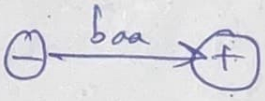


If machine can read upto three letters at a time from input string could be built with even fewer states.



(i)

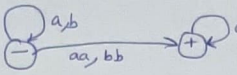
or even



(ii)

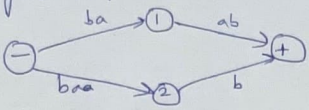
In (2), directions, when no matching of letter like if a is there, are not included.

IF A machine can read one or two letters at a time then, to recognize all words that contain a double letter



just using two states

Example. Consider the machine in which any edge in the picture can be labeled by any string of alphabet letters



1st way: ba, then ab, each final state

2nd way: baa, then b, " " "

In FA, we have only one path to reach final state, & for any input string.

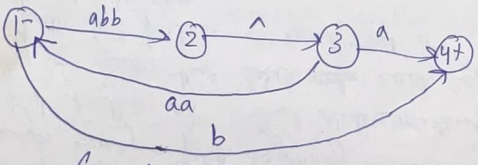
but here, we have no path for strings; while some have several.

These new machines are called "transition graphs" they are easily understood when defined as graph

Definition. A Transition graph TG, is a collection of three things,

1) A finite states set, atleast one of which is designated as the start state (–) and some (may be none) of which are designated as final states (+)
2) An alphabet Σ of possible input letters from which input strings are formed.
3) A finite set of transitions (edge labels) that show how to go from some state to some other state, based on reading specified substrings of input letters (even null Λ)

Example: consider TG

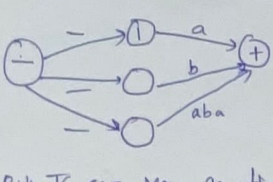


Strings: (abb)(^)(aa)(b)

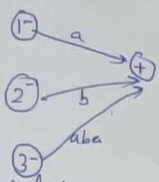
: abbaab

other words accepted: abba, abbaaabba and b

- Edges labeled Λ, free edge, not consuming any letter
- String abbab → Rejected



or



Both TG are same as they are equivalent as language acceptors, different in only no. of states.

→ Every FA is also a TG not every TG satisfies definition of FA.

Strings acceptable to TG

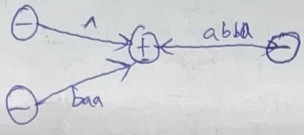
A successful path through a TG is a series of edges forming a path beginning at some state and ending at a final state. Concatenating the string of letters that label each edge in the path will produce a word that is acceptable to the TG.

Examples of TG

1) TG accepts nothing not even ∧ — L = ∅

2) TG accepts everything

3) TG accepts only ∧ — L = ∧

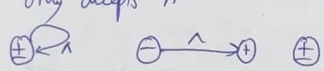
4) 

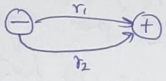
→ It accepts ∧ word, abba, baa

→ Anything read at + state will cause a crash

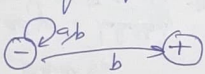
Note: When some start state is final state always accepts ∧

5) TG only accepts ∧



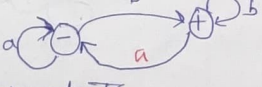
Regular Expression	Transition Graph
1) $r_1 + r_2$	
2) $r_1 \bullet r_2$	
3) r_1^*	

Example: Find the corresponding regular expression for the following TG.



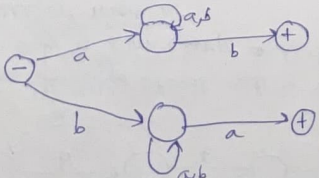
Ans $r = (a+b)^* b$

Corresponding FA



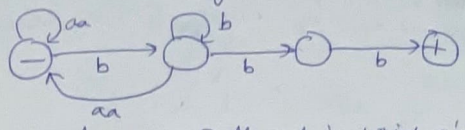
Can also represent TG

Example Consider the TG



Language $L = \{$ It contains the strings of all a's and b's such that they begin and end with different letters $\}$

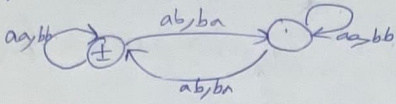
Example: The following TG



→ Accepts language of all words in which a's occur only in even clumps and that end in three or more b's.

→ There is never an edge that takes single a and it takes bbb at the end to get to +.

Example consider TG

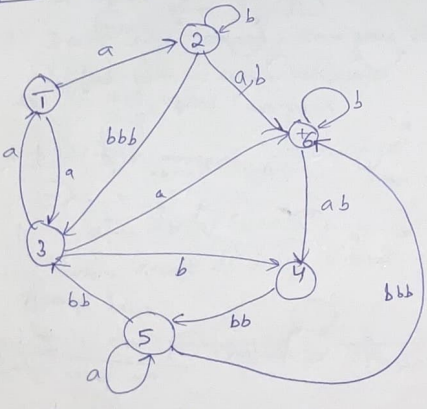


Since every edge is labelled with pair of letters. This means that for the string to be accepted is must have an even no. of letter that are read and processed in groups of two's.

→ The TG shown is for EVEN-EVEN language (signifying ± state as balanced (even state) and ○ state on right is unbalanced state (odd state)

Concluded, that TG has many practical problems There are many possible ways of grouping the letters of the input string that we must examine many possibilities before we know whether a given string is accepted or rejected.

Example Consider TG



Is the word abbbabbbabba accepted by this machine

Ans Yes, in three different ways

q_0, abbbabbbabba
q_2, bbbabbbabba
q_3, bbabbbabba
q_2, babbbabba
q_6, abbbabba
q_4, bbabba
q_5, abba
q_5, bba
q_3, a
q_6, ^ (Accepted)

Actually There are two other ways to do that

Note: If we allow $\wedge$-edges, we have infinite no. of ways of grouping the letter of an input string.

eg ab string, no of ways.

→ (a)(b)

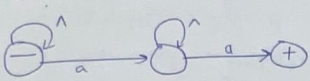
→ (a)(∧)(b)

→ (a)(∧)(∧)(b)

→ (a)(∧)(∧)(∧)(b)

. . . . .

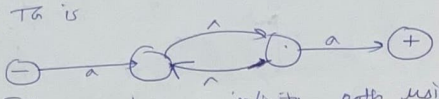
eg Consider TG



Simple word aa

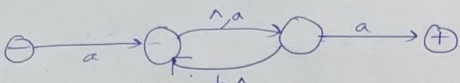
but it can be accepted by infinitely many different paths

If TG is



finite path becomes infinite path using ∧-loops.

With this above machine,



If, ∧ option erased, language is changed.

```
// Note
and
Just like
if (....
    Stat
}
else {
    Stat
}
or
if (....
    Stat
} else {
    Stat
}
and
do {
    stat
    stat
} while
and
switch (
case 1:
    ...
    brea
}
case 2:
    ...
    brea
}
default:
    Writ
    abor
}
}//switch
and
Function
```

Generalized Transition Graphs

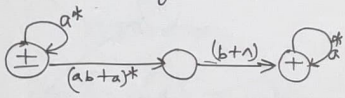
Definition A generalized transition graph is a collection of three things:

1) A finite set of states of which atleast one is a start state and some (may be none) are final state

2) An alphabet Σ of input letters

3) Directed edges connecting some pairs of states, each labelled with a regular expression.

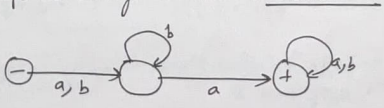
Example



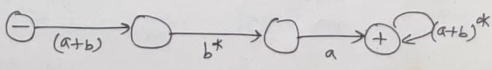
$$a^* + (ab+a)^*(b+\wedge)a^*$$

This machine accepts all strings without a double b.

Compare (1)



(2)



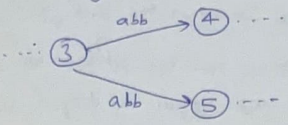
In (1), loop in middle state as many time as we want or go straight to the third state.

In (2), b^* gives the choice of taking $\wedge$ or no. of b's.

Nondeterminism

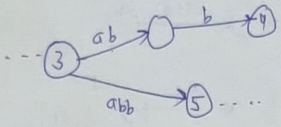
As we know, in a TG a particular string of input letters may trace through the machine on different paths, depending on our choice of grouping

eg



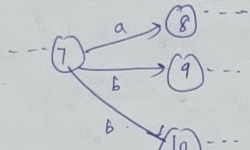
fragment of TG

or

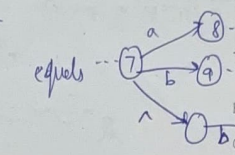


Machine with two letter, or three letter or one letter

or with one letter only



equals



The ultimate path through the machine is not determined by input alone. Therefore, we say this machine is non-deterministic.

Human choice becomes a factor in selecting the path through the machine doesn't make all its own deterministic

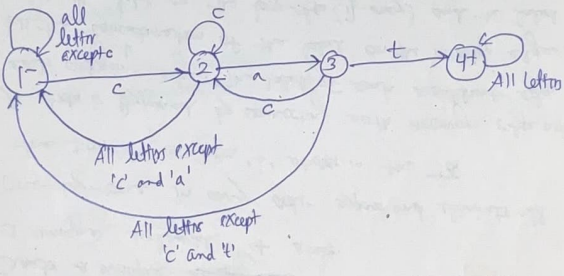
```
----------
FILENAME: Codi
PURPOSE:  This
          codi
          cont
          help
          dist
          and
          are
          anno
AUTHORS:  Jay
          Loke
----------
CHANGES:
Version 1.0 (9
-- The guideli
-- GloMoSim le
   no longer n
----------
Here are the c
General:
    In general,
file you are m
(or modify it
source and sou
Also, the purp
and informativ
Awards".

Naming:
e long, d
Try not to
in variable
are highly

-- Use "Ya
```

~~Ass~~

Ques Build an FA to accept all words that contain the substring 'cat' where

$$\Sigma = \{a, b, c, d, \ldots, z\}$$

Ans



CHAPTER - 7 KLEENE'S THEOREM

We have already studied the seperate ways of defining a language:

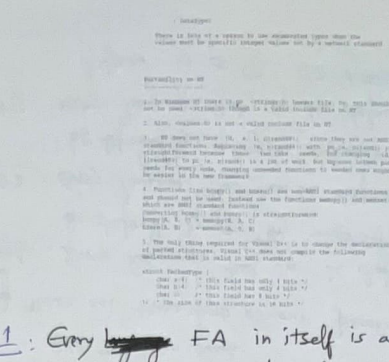
(i) regular expression, or
(ii) finite automaton, or
(iii) Transition Graph

~~Theorem~~ There are three parts of our proof:

part 1: Every language that can be defined by a FA can also be defined by a TG. ~~Already done~~

part 2: Every language that can be defined by a TG can also be defined by a regular expression ~~How to be done~~

part 3: Every language that can be defined by a RE can also be defined by a FA. ~~Already done~~ ~~How to be done~~



part 1: Every ~~language~~ FA in itself is a TG. Therefore, any language that has been defined by FA has already defined by TG.

part 2: Algorithm:

Turning TGs into regular expressions:

Step 1. Create a unique ~~transformable~~ non-reentrable '-' state and a unique unleavable '+' state.

Step 2 One-by-one in any order bypass and eliminate all the non '-' or non '+' states in the TG.

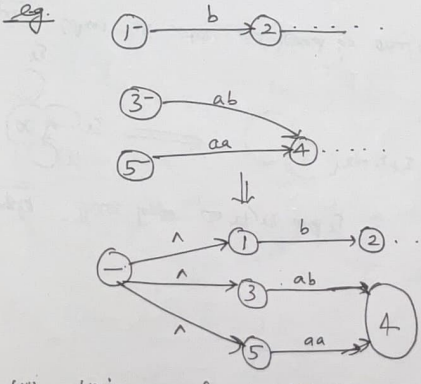
"A state is Bypassed by connecting both incoming edge with each outgoing edge. The label of each resultant edge is the concatenation of the label on the incoming edge with the label on the loop-edge (if any) and the label on the outgoing edge."

Step 3) When two states are joined by more than one edge going in the same direction unify them by adding their labels.

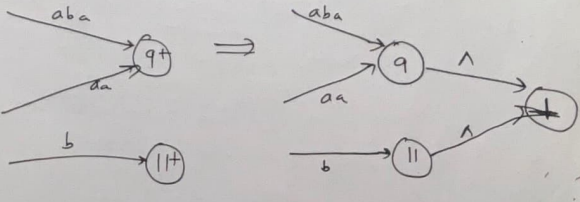
Step 4) Finally, when all that is left is one edge from '-' to '+' the label on that edge is the regular expression that generates the same language as recognized by the original machine.

Step 1 (i) Unique non-reentrable '-' state

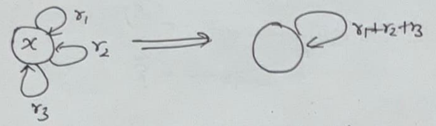
eg.



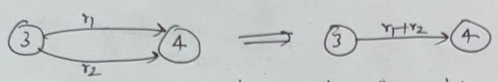
Step 2 (ii) Unique unleavable '+' state



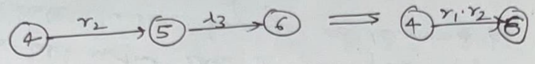
example Three loops as r_1, r_2 and r_3



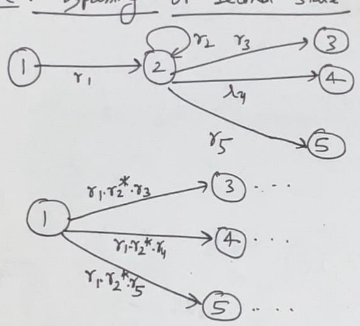
example When two states are joined by one or more edges



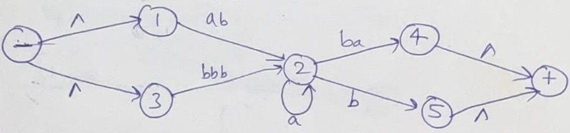
example When one state is shared between two other states



example. Bypassing of second state

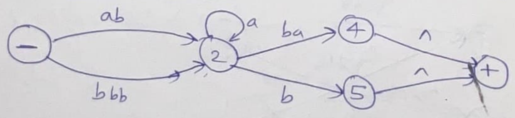


Example 1

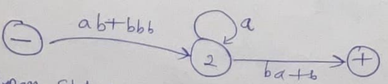


(i) Bypass states 1 and 3

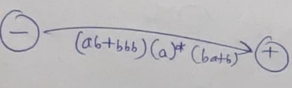
Ans



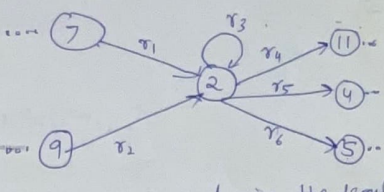
(ii) Bypass states 4 and 5



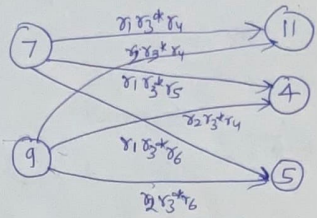
(iii) Bypass state 2



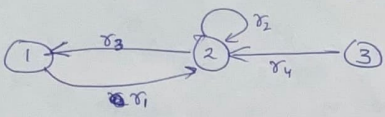
Example 2



Bypass state '2' and give the regular expression

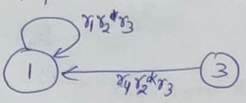


Example 3.

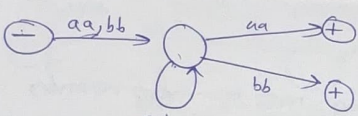


Bypass state '2' ?

Ans

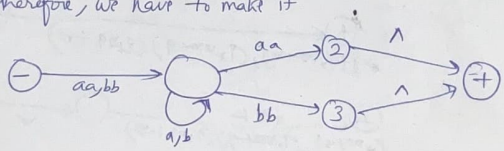


Ques 1.

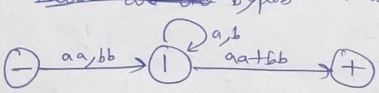


Find the regular expressions

Ans Step1 Since there is no unique unreachable final state. Therefore, we have to make it



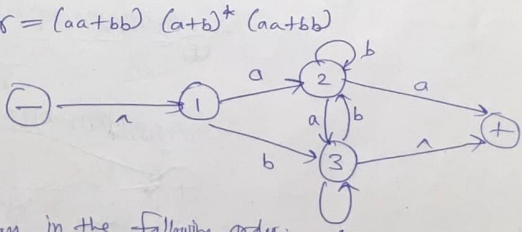
Step2 ~~Delete one one~~ Bypass '2' and '3'



Therefore, regular expression is given by

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

Ques2



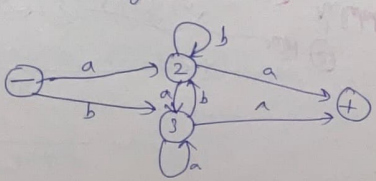
Bypass in the following order.

(i) 1, 2, 3

(ii) 3, 2, 1

find the regular expressions.

(i) Bypass (1)

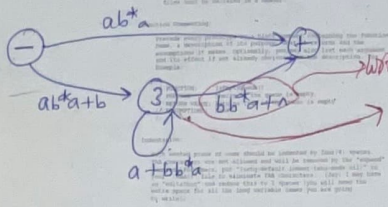


TWO ways to reach from (2)

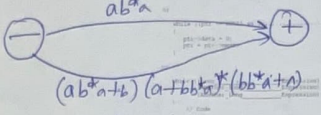
① from (1) directly b

② from (3) ab^*a

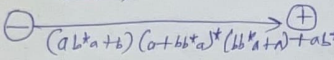
Bypassing state 2



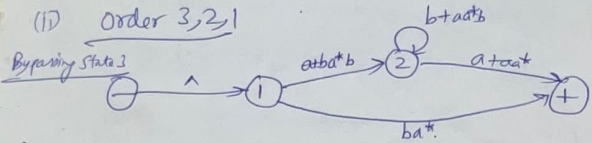
Bypassing state 3



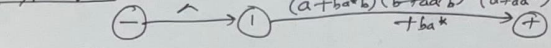
finally,



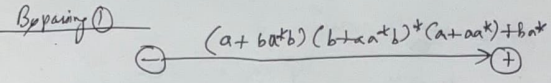
(ii) Order 3, 2, 1



Bypassing 2

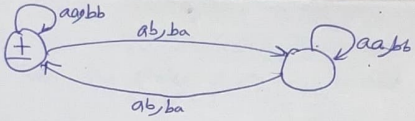


Bypassing 1

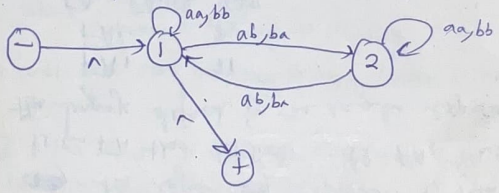


If we had not seen how they were derived, we might have no clue as to whether these regular expressions define the same language

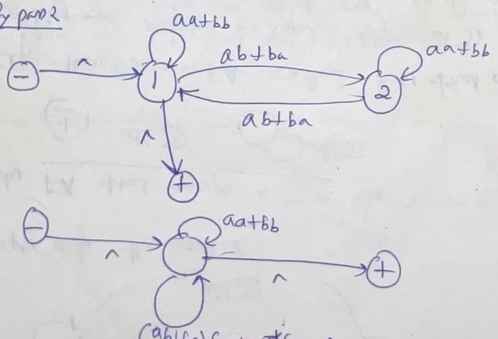
Ques

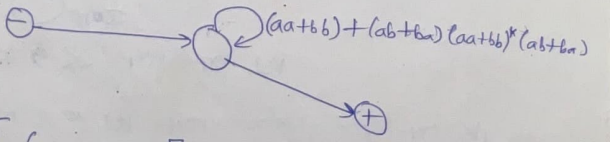


Step 1 Make unique initial state and unique final state



Step 2 By para 2





$$\left[(aa+bb)+\left[(ab+ba)(aa+bb)^*(ab+ba)\right]\right]^*$$

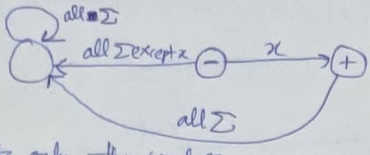
Part 3: Converting a regular expression into FA

Rule 1)

(i) There is an FA that accepts any particular letter of the alphabet and

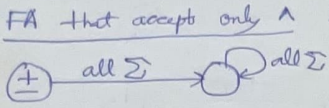
(ii) There is an FA that accepts only the 'Λ'

$\Rightarrow$ If $x \in \Sigma$, then the FA



accepts only the word x.

$\Rightarrow$ only FA that accept only Λ is



Rule 2) If there is an FA called FA_1 that accepts a language defined by regular expression r_1 and there is an FA called FA_2 that accepts the regular expression r_2 then there is an FA that we shall call FA_3 that accepts the language defined by the regular expression $r_1 + r_2$

FA_1 r_1

FA_2 r_2

$FA_3 = FA_1 + FA_2$ $r_1 + r_2$

Algorithm

Let FA_1 have the states $x_1, x_2, x_3, \ldots$ and FA_2 have the states $y_1, y_2, y_3, \ldots$

1) Build a new machine FA_3 with states $z_1, z_2, z_3 \ldots$

Where $z_{something} = x_{something}$ or $y_{something}$

2) The combination state of x_{start} or y_{start} is the initial state of FA_3.

3) If either x_{part} or the y_{part} is the final state then the corresponding z is the final state.

4) Table for transition → To go from one z to another by reading a letter from the input string we see what happens to the x_{part} or the y_{part}.

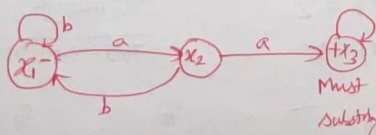
$[z_{new} \text{ after reading letter } p] = [x_{new} \text{ after reading letter } p \text{ in } FA_1]$

or

$[y_{new} \text{ after reading letter } p \text{ in } FA_2]$

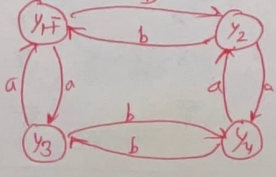
Ques 1 Consider the following two FA's FA_1 & FA_2. Find $FA_1 + FA_2$.

FA_1:



Must have aa as substring in any string

FA_2:



EVEN-EVEN

Transition Table For FA_1

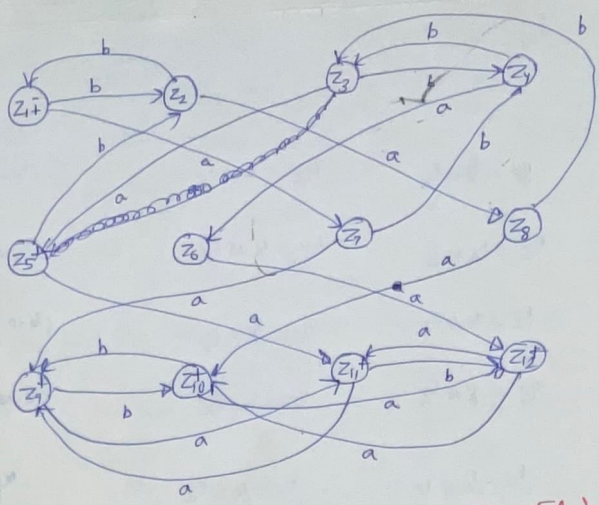
State	Transitions a	b
→ x_1	x_2	x_1
x_2	x_3	x_1
(*) x_3	x_3	x_3

Transition Table For FA_2

State	Transitions a	b
→ y_1	y_3	y_2
y_2	y_4	y_1
y_3	y_1	y_4
y_4	y_2	y_3

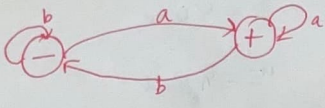
$Z_1 = (x_1 \text{ or } y_1)$
$Z_2 = (x_1 \text{ or } y_2)$
$Z_3 = (x_1 \text{ or } y_3)$
$Z_4 = (x_1 \text{ or } y_4)$
$Z_5 = (x_2 \text{ or } y_1)$
$Z_6 = (x_2 \text{ or } y_2)$
$Z_7 = (x_2 \text{ or } y_3)$
$Z_8 = (x_2 \text{ or } y_4)$
$Z_9 = (x_3 \quad y_1)$
$Z_{10} = (x_3 \quad y_2)$
$Z_{11} = (x_3 \quad y_3)$
$Z_{12} = (x_3 \quad y_4)$

State	a	b
± Z_1 $(x_1 \text{ or } y_1)$	$x_2 \text{ or } y_3 = Z_7$	$x_1 \text{ or } y_2 = Z_2$
Z_2 $(x_1 \text{ or } y_2)$	$x_2 \text{ or } y_4 = Z_8$	$x_1 \text{ or } y_1 = Z_1$
Z_3 $(x_1 \text{ or } y_3)$	$x_2 \text{ or } y_1 = Z_5$	$x_1 \text{ or } y_4 = Z_4$
Z_4 $(x_1 \text{ or } y_4)$	$x_2 \text{ or } y_2 = Z_6$	$x_1 \text{ or } y_3 = Z_3$
+ Z_5 $x_2 \text{ or } y_1$	$x_3 \text{ or } y_3 = Z_{11}$	$x_1 \text{ or } y_2 = Z_2$
Z_6 $x_2 \text{ or } y_2$	$x_3 \text{ or } y_4 = Z_{12}$	$x_1 \text{ or } y_1 = Z_1$
Z_7 $x_2 \text{ or } y_3$	$x_3 \text{ or } y_1 = Z_9$	$x_1 \text{ or } y_4 = Z_4$
Z_8 $x_2 \text{ or } y_4$	$x_3 \text{ or } y_2 = Z_{10}$	$x_1 \text{ or } y_3 = Z_3$
+ Z_9 $(x_3 \text{ or } y_1)$	$x_3 \text{ or } y_3 = Z_{11}$	$x_3 \text{ or } y_2 = Z_{10}$
+ Z_{10} $(x_3 \text{ or } y_2)$	$x_3 \text{ or } y_4 = Z_{12}$	$x_3 \text{ or } y_1 = Z_9$
+ Z_{11} $(x_3 \text{ or } y_3)$	$x_3 \text{ or } y_1 = Z_9$	$x_3 \text{ or } y_4 = Z_{12}$
+ Z_{12} $(x_3 \text{ or } y_4)$	$x_3 \text{ or } y_2 = Z_{10}$	$x_3 \text{ or } y_3 = Z_{11}$

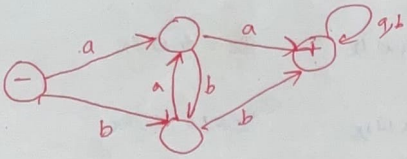


Ques Construct FA for the language $(FA_1 + FA_2)$ Where FA_1 and FA_2 are given below:

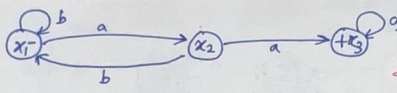
FA_1:



FA_2:

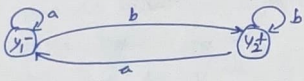


Ques FA_1:



End with double a

FA_2:



End with a b

Construct $FA_1 + FA_2$.

Ans.

FA_1

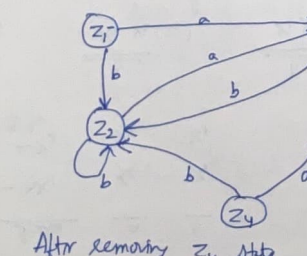
States	a	b
$-x_1$	x_2	x_1
x_2	x_3	x_1
$+x_3$	x_3	x_3

FA_2:

States	a	b
$-y_1$	y_1	y_2
$+y_2$	y_1	y_2

$\rightarrow Z_1$:	x_1	y_1
Z_2:	x_1	(y_2)
Z_3:	x_2	y_1
Z_4:	x_2	(y_2)
Z_5:	(x_3)	y_1
Z_6:	(x_3)	y_2

States	a	b
$-Z_1$	x_2 or $y_1 = Z_3$	x_1 or $y_2 = Z_2$
$+Z_2$	x_2 or $y_1 = Z_3$	x_1 or $y_2 = Z_2$
Z_3	x_3 or $y_1 = Z_5$	x_1 or $y_2 = Z_2$
$+Z_4$	x_3 or $y_1 = Z_5$	x_1 or $y_2 = Z_2$
$+Z_5$	x_3 or $y_1 = Z_5$	x_3 or $y_2 = Z_6$
$+Z_6$	x_3 or $y_1 = Z_5$	x_3 or $y_2 = Z_6$



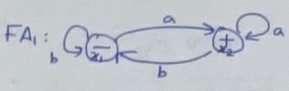
After removing Z_4 state final answer

Z_4 state is never reachable

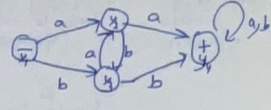
Z_4 (x_2 or y_2)

reachable from x_1 when last letter read is 'a'

reachable from y_1 when last letter read is 'b'

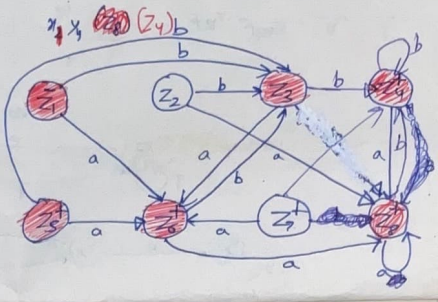
FA_1: 

States	a	b
$\rightarrow x_1$	x_2	x_1
$+x_2$	x_2	x_1

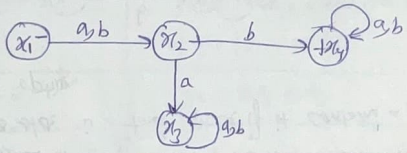
FA_2: 

States	a	b
$\rightarrow y_1$	y_2	y_3
y_2	y_4	y_3
y_3	y_2	y_4
$+y_4$	y_4	y_4

	a	b
$\rightarrow z_1$ (x_1, y_1) ✓	$x_2\ y_2\ (z_6)$	$x_1\ y_3\ (z_3)$
z_2 (x_1, y_2)	$x_2\ y_4\ (z_8)$	$x_1\ y_3\ (z_3)$
z_3 (x_1, y_3) ✓	$x_2\ y_2\ (z_6)$	$x_1\ y_4\ (z_4)$
$+z_4$ (x_1, y_4) ✓	$x_2\ y_4\ (z_8)$	$x_1\ y_4\ (z_4)$
$+z_5$ (x_2, y_1)	$x_2\ y_2\ (z_6)$	$x_1\ y_3\ (z_3)$
$+z_6$ (x_2, y_2) ✓	$x_2\ y_4\ (z_8)$	$x_1\ y_3\ (z_3)$
$+z_7$ (x_2, y_3)	$x_2\ y_2\ (z_6)$	$x_1\ y_4\ (z_4)$
$+z_8$ (x_2, y_4) ✓	$x_2\ y_4\ (z_8)$	$x_1\ y_4\ (z_4)$

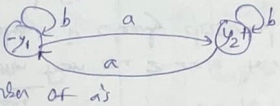


THEOREM 2: If there is an FA_1 that accepts the language defined by the regular expression r_1 and an FA_2 that accepts the language defined by the regular expression r_2. Then there is an FA_3 that accepts the language defined by the concatenation of $r_1 \cdot r_2$.

Example: FA_1: 

Second letter must always be a letter b

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

FA_2: 

odd number of a's

$L = \{$ strings of all a's and b's such that it contains odd number of a's $\}$

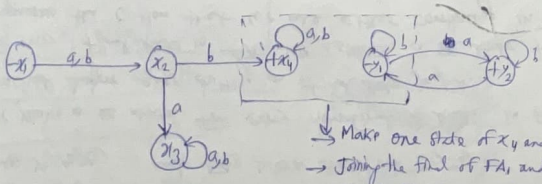
$FA_3 = FA_1 \cdot FA_2$.

First part of the word should be accepted by FA_1 and the latter half should be accepted by FA_2.

eg A word like aabab won't be accepted

A word like abbab is accepted.

abaab



→ Make one state of x_4 and y_1.

→ Joining the final of FA_1 and Initial of FA_2

Drawback

If the word is 'ababbab' it must be rejected but by clubbing them it is accepted.

ALGORITHM FOR $FA_1 \cdot FA_2$

Let us suppose FA_1 whose states are $x_1, x_2, \ldots, x_r \ldots$ and FA_2 whose states are $y_1, y_2, y_3, \ldots$ and FA_3 whose states are $z_1, z_2, z_3 \ldots$

(a) We make a Z state for every non-final X-state in FA_1 reached before ever hitting a final state in FA_1

(b) For every final state in FA_1, we establish a Z state that expresses the option that we are either continuing in FA_1 or beginning on FA_2 i.e.

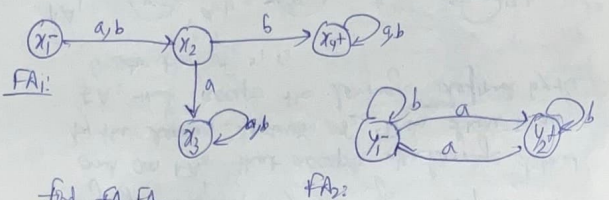
$Z_{something} =$ { are in X something which is + state and still continuing on FA_1 **or** have finished with FA_1 part of input string and have jumped to FA_2 to commence tracing the remainder of string.

(c) Transition table

The transition from one Z-state to another for each letter of an alphabet is clearly determined by the transition rules of FA_1 and FA_2.

(d) The final state of FA_3 means the final state of FA_2. So, any Z-state is a final state if it contains a y-final as a possible option

Ques



FA_1:

find $FA_1 \cdot FA_2$

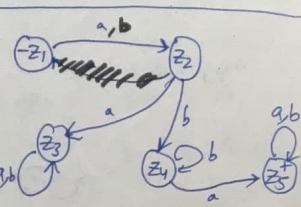
FA_2:

Ans $\quad z_1 = x_1, \; z_2 = x_2, \; z_3 = x_3$

	a	b
z_1	goes to $x_2 = z_2$	goes to $x_2 = z_2$
z_2	goes to $x_3 = z_3$	goes to x_4, remain on FA_1 **or** jumps to y_1, starts with FA_2 $= x_4 \mid y_1 = z_4$
z_3	goes to $x_3 = z_3$	goes to $x_3 = z_3$
z_4 $(x_4 \mid y_1)$	x_4 loops x_4, remains of FA_1 **or** x_4 jumps y_1, starts FA_2 **or** y_1 goes y_2 $\Rightarrow x_4 \mid y_1 \mid y_2 = z_5$	x_4 loops x_4, remains on FA_1 **or** x_4 jumps to y_1, starts with FA_2 **or** y_1 loops to y_1 $\Rightarrow x_4 \mid y_1 \mid y_1 = z_4$
z_5^+ $(x_4 \mid y_1 \mid y_2)$	x_4 loops to x_4, remains on FA_1 **or** x_4 jumps to y_1, starts with FA_2 or y_1 goes to y_2 y_2 goes to y_1 $x_4 \mid y_1 \mid y_2 \mid x = z_5$	x_4 loops to x_4, remains on FA_1 **or** x_4 jumps to y_1, starts with FA_2 **or** y_1 loops to y_1 y_2 loops to y_2 $= x_4 \mid y_1 \mid y_1 \mid y_2 = z_5$

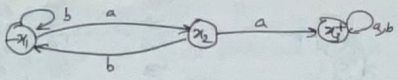
No, new state, then stop

	a	b
$-z_1$	z_2	z_2
z_2	z_3	z_4
z_3	z_3	z_3
z_4	z_5	z_4
$+z_5$	z_5	z_5

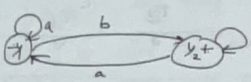


Ques2.

FA1:



FA2:

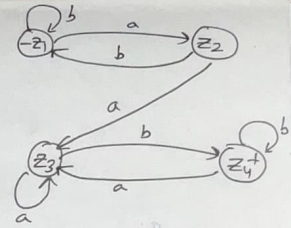


~~find~~ FA1 · FA2.

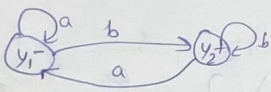
Ans $z_1 = x_1$ $z_2 = x_2$

	a	b
z_1	goes to $x_2 = z_2$	loops to $x_1 = z_1$
z_2	goes to x_3 and remain on FA_1 or jumps to y_1, starts with FA_2. $x_3 \mid y_1 = z_3$	goes to $x_1 = z_1$
z_3	x_3 loops to x_3, remain on FA_1 or jumps to y_1, starts with FA_2. or y_1 loops to y_1 $x_3 \mid y_1 = z_3$	x_3 loops to x_3. remain on FA_1 or jumps to y_1, starts with FA_2 or y_1 goes to y_2 $x_3 \mid y_1 \mid y_2 = z_4$
z_4^+	x_3 loops to x_3, remain on FA_1 or jumps to y_1, starts with FA_2 or y_1 loops to y_1, y_2 goes to y_1 $x_3 \mid y_1 \mid y_1 \mid y_1 = z_3$	x_3 loops to x_3, or jumps to y_1, starts with FA_2. y_1 goes to y_2 y_2 loops to y_2 $= x_3 \mid y_1 \mid y_2 \mid y_2 = z_4$

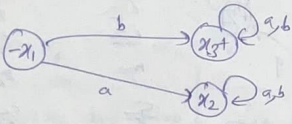
	a	b
z_1	z_2	z_1
z_2	z_3	z_1
z_3	z_3	z_4
z_4	z_3	z_4



Ques3. FA1:



FA2:



Calculate FA1 · FA2.

$z_1 = y_1$

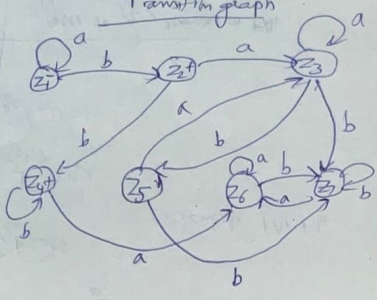
	a	b
z_1	loops to $y_1 = z_1$	goes to y_2, remains on FA_1 or jumps to x_1, starts with FA_2 $y_2 \mid x_1 = z_2$
z_2 $y_2 \mid x_1$	goes to y_1 or x_1 goes to x_2 ie $y_1 \mid x_2 = z_3$	y_2 loops to y_2 or jumps to x_1 and remains on FA_1 starts with FA_2 x_1 goes to x_3 $y_2 \mid x_1 \mid x_3 = z_4$
z_3 $y_1 \mid x_2$	loops to x_1 or x_2 goes loops to x_2 $y_1 \mid x_2 = z_3$	goes to y_2 or jumps to x_1 or x_2 loops to x_2 $y_2 \mid x_1 \mid x_2 = z_5$
z_4 + $y_2 \mid x_1 \mid x_3$	goes to y_1 or x_1 goes to x_2 or x_3 loops to x_3 $y_1 \mid x_2 \mid x_3 = z_5$	loops to y_2 stays at FA_1 or jumps to x_1 starts with FA_2 or x_1 goes to x_3 or x_3 loops to x_3 $y_2 \mid x_1 \mid x_3$ $= z_4$

z_5 $y_2\|x_1\|x_2$	goes to x_1, goes to x_2, goes to x_2, $y_1\|x_2\|x_2 = z_3$	loops to x_2, goes to x_1, goes to x_3, loops to x_2 $y_2\|x_1\|x_3\|x_2 = z_7$
z_6+ $y_1\|x_2\|x_3$	loops to y_1, loops to x_2, loops to x_3 $y_1\|x_2\|x_3 = z_6$	goes to x_1 or goes to x_1, loops to x_2, loops to x_3 $y_2\|x_1\|x_2\|x_3 = z_7$
z_7 + $y_2\|x_1\|x_1\|x_3$	goes to x_1, goes to x_2, loops to x_2, loops to x_3 $x_1\|x_2\|x_3 = z_8$	loops to x_2, goes to x_1, goes to x_3, loops to x_2, loops to x_3 $y_2\|x_1\|x_3\|x_2\|x_3 = z_7$

Transition table

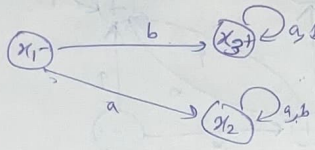
States	a	b
$-z_1$	z_1	z_2
z_2	z_3	z_4
z_3	z_3	z_5
$+z_4$	z_6	z_4
z_5	z_3	z_7
$+z_6$	z_6	z_7
$+z_7$	z_6	z_7

Transition graph

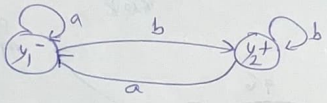


Ques. Construct FA_1. FA_2

FA_1 :



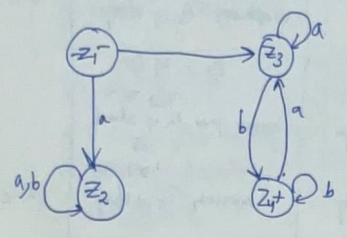
FA_2 :



Ans. $z_1 = x_1$ $z_2 = x_2$

	a	b
z_1 (x_1)	goes to x_2 and stop $= z_2$	goes to x_3 remains on FA_1 or jumps to y_1 and starts with FA_2 $x_3\|y_1 = z_3$
z_2 (x_2)	loops to $x_2 = z_2$	loops to $x_2 = z_2$
z_3 $(x_3\|y_1)$	loops to x_3, remains on FA_1 or jumps to y_1 and starts with FA_2 $x_3\|y_1 = z_3$	loops to x_3 and remains on FA_1 or jumps to y_1 and then to y_2 $x_3\|y_1\|y_2 = z_4$
+ z_4 $(x_3\|y_1\|y_2)$	loops to x_3, remains on FA_1, starts with FA_2, loops to y_1 from y_2 to y_1 $= x_3\|y_1 = z_3$	loops to x_3, jumps to y_1 and then y_2, from y_2 loops to y_2 $= x_3\|y_1\|y_2 = z_4$

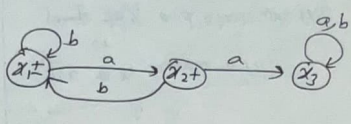
States	a	b
$-Z_1$	Z_2	Z_3
Z_2	Z_2	Z_2
Z_3	Z_3	Z_4
$+Z_4$	Z_3	Z_4



Note: If you will try to find out $FA_2 \cdot FA_1$, result is gonna be different (Pg 123)

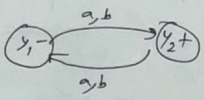
Ques. Pg 104.

FA_1:

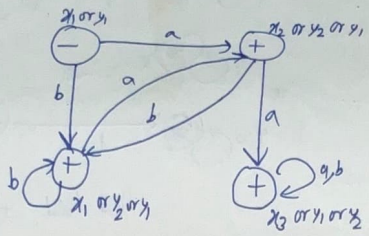


Don't contain the substring aa

FA_2:



All words with odd number of letters



Theorem: If r is a regular expression and FA_1 is an FA that accepts the language defined by r then there is an FA called FA_2 that will accept exactly the same language as defined by r.

ALGORITHM

Incomplete — Real

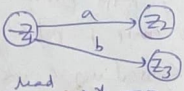
Build FA for r:

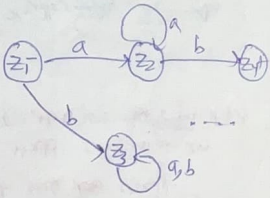
Start with incomplete Algo.

1) $-Z_1 = x_1$

2) $Z_1 \xrightarrow[a]{\text{read}} x_2 = Z_2$

$Z_1 \xrightarrow[\text{read}]{b} x_3 = Z_3$





3) $Z_2 \xrightarrow[a]{\text{read}} x_2 = Z_2$

$Z_2 \xrightarrow[b]{\text{read}} \begin{Bmatrix} x_4 \\ \text{or} \\ x_1 \end{Bmatrix} = Z_4(+)$ (∵ of x_4)

4) $Z_3 \xrightarrow[a \text{ or } b]{\text{read}} x_3 = Z_3$ (a bondon state)

5) $+Z_4 \xrightarrow[a]{\text{read}} \begin{Bmatrix} x_3 \\ \text{or} \\ x_2 \end{Bmatrix} = Z_5$

$Z_4 \xrightarrow[b]{\text{read}} \begin{Bmatrix} x_3 \\ \text{or} \\ x_4 \\ \text{or} \\ x_1 \end{Bmatrix} = Z_6$

Algorithm (Real)

Given an FA_1 whose states are $x_1, x_2, \ldots$ any FA_1 that accepts the Kleene's Closure of the language of the original machine can be build as follows.

Step1: Create a state for every subset of x's. Cancel any subset that contains a final x-state, but don't contain the start state.

Step2: For all the remaining nonempty states, draw an a-edge and a b-edge to the collection of x-states reachable in the original FA from the component x's by a and b-edge, respectively.

Step3: Call the null subset a $\pm$ state and connect it to whatever states the original start state is connected to by a and b edges, even possibly the start state itself.

Step4: Finally, put $+$ signs in every state containing an x-component that is a final state of the original FA.

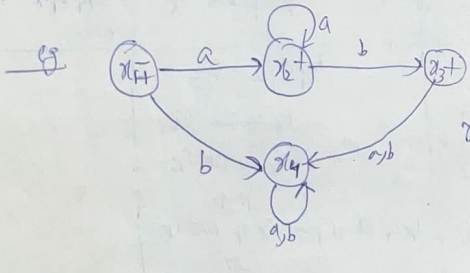
Incomplete (Algorithm)

Given an FA_1 whose states are $x_1, x_2, x_3, x_4, \ldots$ An FA_2 that accepts the Kleene's Closure of the language of the original machine be build as follows.

Step1) We make the start state of FA_1 as the z_{start} of FA_2 and mark z_{start} as the initial as well as the final state.

Step2) On hitting a final state in FA_1, we establish a z-state that expresses the option that we can remain on a final state of FA_1 or can start all over again at the initial state of FA_1.

$z_{something}$ = { x_{final} and continuing processing the string of type r / or / We have accepted a section of input string and start with x_{start} for the next section of input string }



$r = a^* + aa^*b$

Step3 Mark the z_{final} whichever state contains the x_{final}.

In our example,

the FA_1 can accept the string aaaa also it can accept the string aab

Concatenating aaaa | aab we get a new string which must be accepted by r^*

$r^* = (a^* + aab)^*$

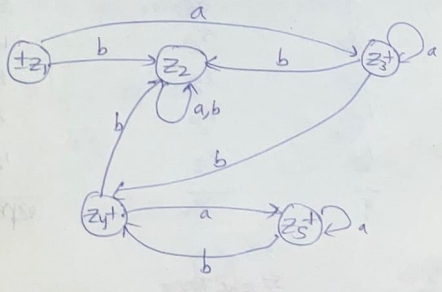
because it can be rejected by FA_1 but it can be accepted by FA_1^*

$FA_1 \longrightarrow r$

$(FA_1)^* \longrightarrow r^*$

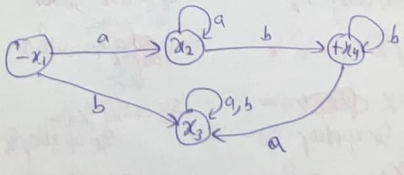
State	a	b
$\rightarrow z_1 = x_1$	x_1 goes to x_2 or x_1 goes x_2 & jumps to x_1 $\Rightarrow x_2 \mid x_1 = z_3$	x_1 goes x_4 $z_2 = x_4$
$z_2 = x_4$	x_4 remains x_4 $z_2 = x_4$	x_4 remain x_4 $z_2 = x_4$
$z_3 = x_1 \mid x_2$	x_1 goes to x_2 or x_1 goes x_2 & jumps to x_1 and x_2 goes x_2, x_2 jumps to x_1 $z_3 = x_1 \mid x_2$	x_1 goes x_4 x_2 goes x_3 or x_2 goes x_3 and jumps to x_1 $z_4 = x_1 \mid x_3 \mid x_4$
$z_4 = x_1 \mid x_3 \mid x_4$	x_1 goes to x_2 or x_1 goes to x_2 and jumps to x_1 and x_3 goes to x_4 x_4 goes to x_4 $z_5 = x_1 \mid x_2 \mid x_4$	x_1 goes to x_4 x_3 goes to x_4 x_4 goes to x_4 $z_2 = x_4$
$z_5 = x_1 \mid x_2 \mid x_4$	x_1 goes to x_2 or x_1 goes to x_2 and jumps to x_1 and x_2 goes to x_2 & x_4 loops to x_4 $x_1 \mid x_2 \mid x_4 = z_5$	x_1 goes to x_4 x_2 goes to x_3 & x_2 goes to x_3 jumps to x_1 & x_4 loops to x_4 $= x_4 \mid x_3 \mid x_1 = z_4$

States	a	b
$\rightarrow z_1$	z_3	z_2
z_2	z_2	z_2
$+z_3$	z_3	z_4
$+z_4$	z_5	z_2
$+z_5$	z_5	z_4



Ques Construct an $(FA_1)^*$

FA1



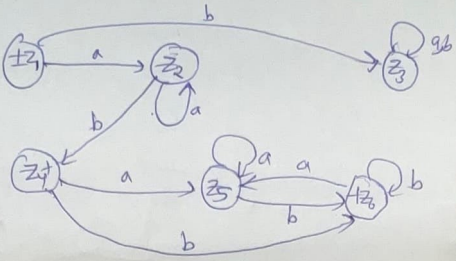
Ans

State	a	b
$\rightarrow z_1 = x_1$	x_1 goes to x_2 $= z_2$	x_1 goes to x_3 $= z_3$
$z_2 = x_2$	x_2 loops to x_2 $= z_2$	x_2 goes x_4 or x_2 goes to x_4 & jumps to x_1 $\Rightarrow x_1 \mid x_4 = z_{\&}$

	a	b
$Z_3 = x_3$	loops to $x_3 = Z_3$	loops to $x_3 = Z_3$
$+Z_4 = x_1 \mid x_4$	x_1 goes to x_2 and x_4 goes to x_3 $\Rightarrow x_2 \mid x_3 = Z_5$	x_1 goes to x_3 x_4 loops to x_4 goes to x_1 $\Rightarrow x_3 \mid x_4 \mid x_1 = Z_6$
$Z_5 = x_2 \mid x_3$	x_2 loops to x_2 x_3 loops to x_3 $\Rightarrow x_2 \mid x_3 = Z_5$	x_2 goes to x_4 jumps to x_1 x_3 loops to x_3 $\Rightarrow x_4 \mid x_1 \mid x_3 = Z_6$
$+Z_2 = x_1 \mid x_3 \mid x_4$	x_1 goes to x_2 x_3 loop to x_3 x_4 goes to x_3 $\Rightarrow x_2 \mid x_3 = Z_5$	x_1 goes to x_3 x_3 loops to x_3 x_4 loops to x_4 or jumps to x_1 $\Rightarrow x_1 \mid x_3 \mid x_4 = Z_6$

States	a	b
$+Z_1$	Z_2	Z_3
Z_2	Z_2	Z_4
Z_3	Z_3	Z_3
$+Z_4$	Z_5	Z_6
Z_5	Z_5	Z_6
$+Z_6$	Z_5	Z_6

TG for FA*

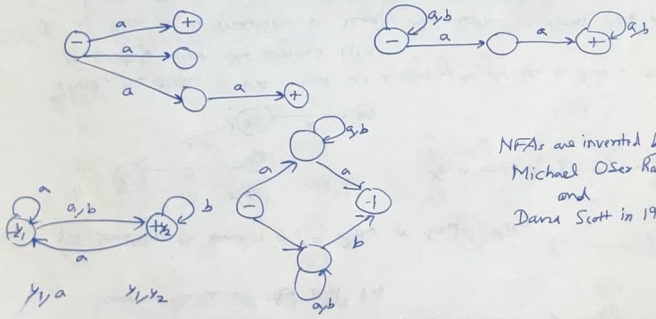


Pg 135 Nondeterministic Finite Automata

Conversion from nondeterministic machine like a TG to a deterministic machine, an FA

A NDFA or NFA is a TG with a unique start state with the property that each of its edge labels is a single alphabet letter.

Example These are all NFAs:



NFAs are invented by Michael Oser Rabin and Dana Scott in 1959

Y_1, a	Y_1, Y_2
Y_1, b	Y_2
Y_2, a	Y_1
Y_2, b	Y_2

An NDFA is defined by a quintuple or 5-tuple $(Q, \Sigma, \delta, q_0, F)$

Where (i) Q is the finite non-empty set of states
(ii) Σ is a finite non-empty set of letters called the alphabet
(iii) δ is the transition function from $Q \times \Sigma$ into 2^Q which is a power set of Q (The set of all subsets of Q)
(iv) $q_0 \in Q$ which is the initial state and
(v) F is the subset of Q which is a set of final states.

→ In simple words for NFA we can't uniquely determine which states to go next when at a given state any input is applied

→ A string $w \in \Sigma^*$ is accepted by an NFA if $\delta(q_0, w)$ contains some final states

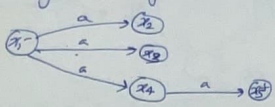
THEOREM for every NFA there exist a DFA that simulates the behaviour of DFA

Algorithm : NFA to DFA

1.) The initial state of new FA is the same as that of old FA.

2.) We construct the transition table of a new FA by using the transition table of the old FA.

{x something on reading 'a'} = x_{this} or x_{that} ... x_{other}



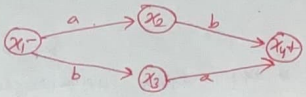
$x_1, a = x_2$ or x_3 or x_4

Thus, the states in FA that we are going to produce will be a collection of x-states from the original NFA.

3.) If there is no transitions on reading the letter 'a' from the x-state in the NFA then we can say that, on reading the letter 'a' the new state reached is a Null state $\emptyset$. This state has a loop for the letter a and b.

4.) The final state is a collection of x-state from the original machine that includes the old final state in them.

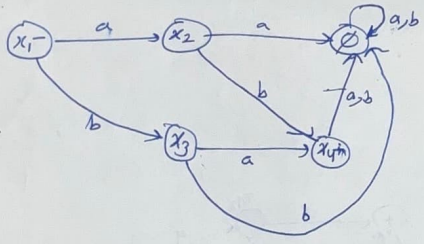
Eg NFA to DFA



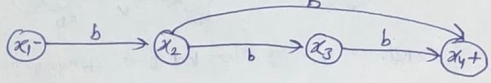
States	a	b
→ x_1	x_2	x_3
x_2	$\emptyset$	x_4
x_3	x_4	$\emptyset$
+ x_4	$\emptyset$	$\emptyset$
$\emptyset$	$\emptyset$	$\emptyset$

5 states in the DFA

$x_1, x_2, x_3, x_4, \emptyset$

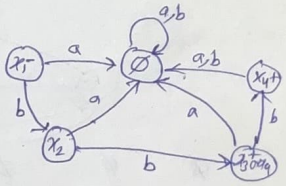


Exam Ques Consider an NFA and construct the DFA



Ans Transition table for NFA

States	a	b
→ x_1	$\emptyset$	x_2
x_2	$\emptyset$	x_3, x_4
x_3	$\emptyset$	x_4
+ x_4	$\emptyset$	$\emptyset$



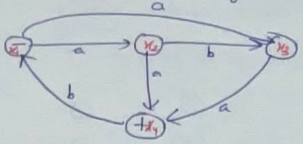
Transition Table for DFA

	a	b
→ $[x_1]$	$\emptyset$	x_2
$[x_2]$	$\emptyset$	x_3, x_4
+ $[x_3, x_4]$	$\emptyset$	x_4
+ $[x_4]$	$\emptyset$	$\emptyset$
$\emptyset$	$\emptyset$	$\emptyset$

1997

Ques 4) (a) Convert DFA for the following NFA

chap 7 Ex 14) (80 part (5 marks)

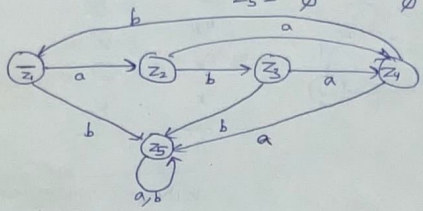


Ans Transition Table for NFA

State	a	b
$-x_1$	x_2, x_3	$\emptyset$
x_2	x_4	x_3
x_3	x_4	$\emptyset$
$+x_4$	$\emptyset$	x_1

Transition Table for DFA

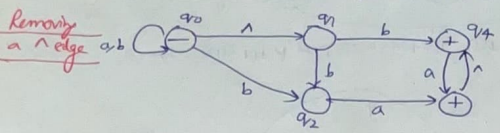
States	a	b
$z_1 = -[x_1]$	x_2, x_3	$\emptyset$
$z_2 = [x_2, x_3]$	x_4	x_3
$z_3 = [x_3]$	x_4	$\emptyset$
$z_4 = +[x_4]$	$\emptyset$	x_1
$z_5 = \emptyset$	$\emptyset$	$\emptyset$



1788

Ques (6 marks) Construct a DFA equivalent to the NDFA

Removing a $\wedge$ edge



Algorithm for '$\wedge$' occurring in a Transition Graph (Mishra and Chandrashekaran)

To replace a '$\wedge$' (Null string) from state q_1 to q_2 we proceed as follows

Step1 Find all the edges starting from q_2 and say to q_3 and q_4

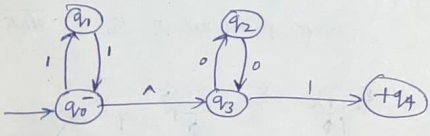
Step2 Duplicate all ~~these~~ these edges starting from q_1 without changing the edge labels i.e edges from q_1 to q_3 and q_1 to q_4 with edge labels same as from q_2 to q_3 and q_2 to q_4

Step3 If q_1 is an initial state, make q_2 also as a initial state

Step4 If q_2 is the final state, make q_1 also as a final state

Step5 Remove the '$\wedge$' between q_1 and q_2

Ques

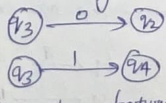


Remove Null edge between q_0 and q_3

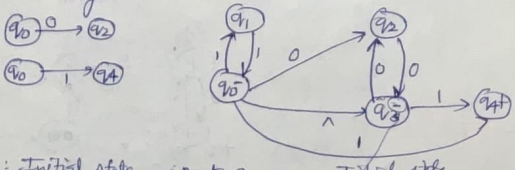
Ans

Initial state	q_0
Final state	q_4

Step1 Find all the edges starting from q_3



Step2 Make new edges between q_0 to q_2 and q_0 to q_4

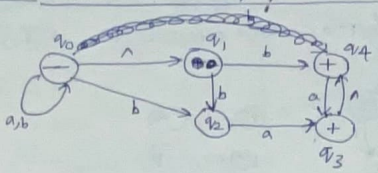


Step3: q_0 : Initial state ; make q_3 as a Initial state

Step4: Not applicable

Step 5 Remove '$\wedge$' edge

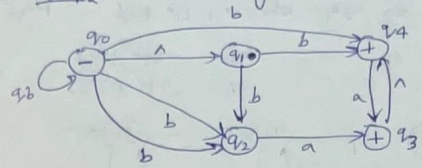
Q. Construct the DFA equivalent to NDFA



Ans Step1 $q_0 \xrightarrow{b} q_2$

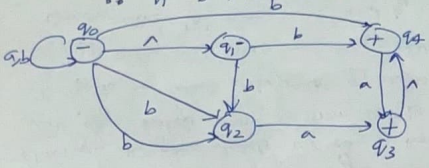
$q_1 \xrightarrow{b} q_4$

Step2 Make new edges between $q_0 \xrightarrow{b} q_2$, $q_0 \xrightarrow{b} q_4$



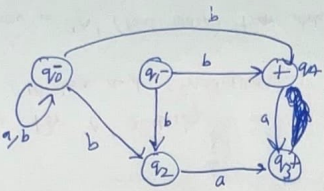
Step3 q_0 is the initial state

$\therefore$ q_1 is the initial state



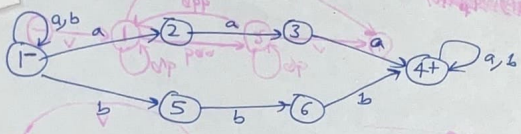
Step4 Not applicable

Step5 Remove '^' edge



Now convert NFA to DFA

Construct the DFA



Ans Transition Table for NFA

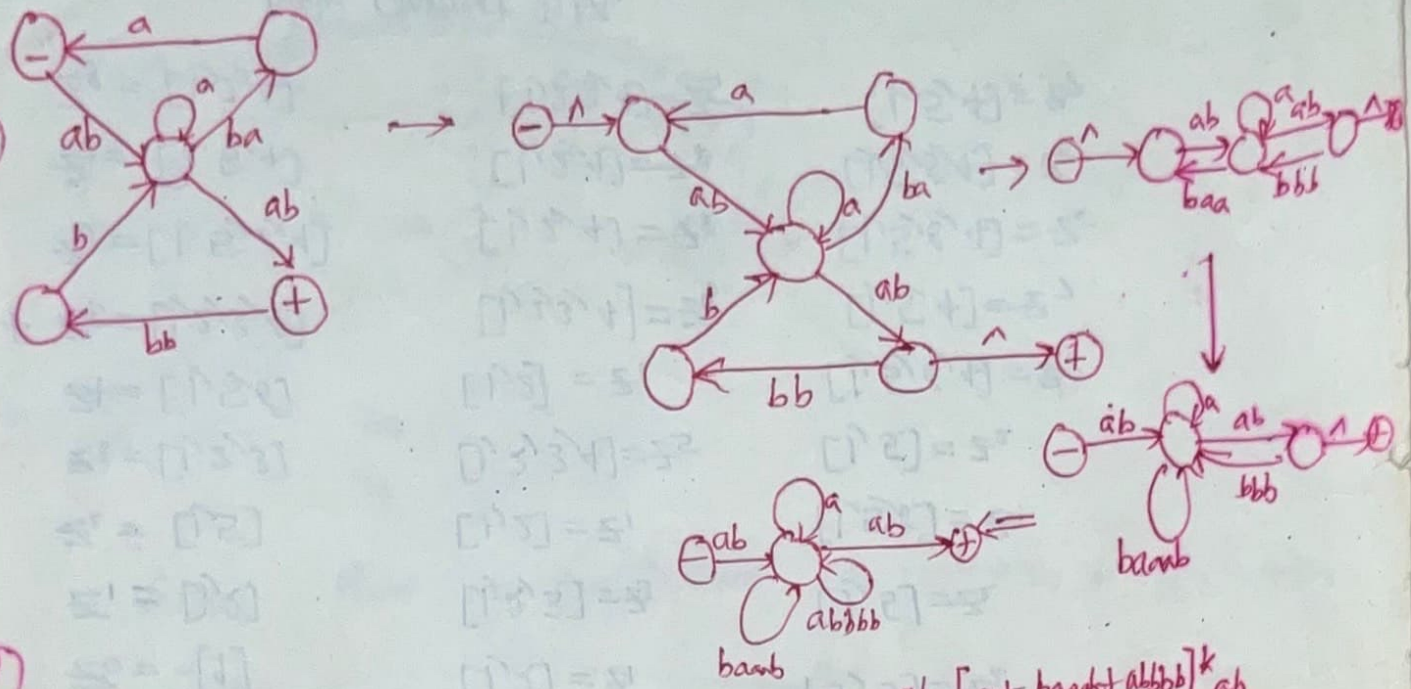
States	Transitions a	Transitions b
→1	[1,2]	[1,5]
2	3	∅
3	4	∅
+4	4	4
5	∅	6
6	∅	4

Transition Table for DFA

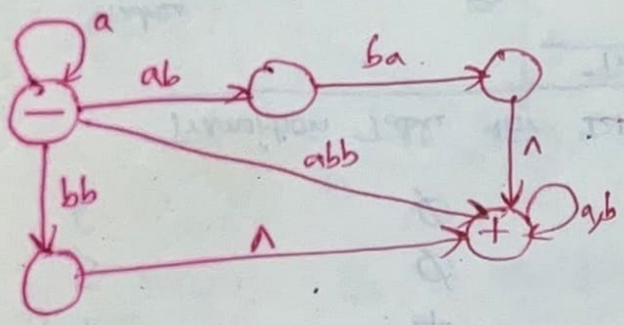
States	a	b
$Z_0 = [1]$	$[1,2] = Z_1$	$[1,5] = Z_2$
$Z_1 = [1,2]$	$[1,2,3] = Z_3$	$[1,5] = Z_2$
$Z_2 = [1,5]$	$[1,2] = Z_1$	$[1,5,6] = Z_4$
$Z_3 = [1,2,3]$	$[1,2,3,4] = Z_5$	$[1,5] = Z_2$
$Z_4 = [1,5,6]$	$[1,2] = Z_1$	$[1,5,6,4] = Z_6$
+ $Z_5 = [1,2,3,4]$	$[1,2,3,4] = Z_5$	$[1,5,4] = Z_7$
$Z_6 = [1,5,6,4]$	$[1,2,4] = Z_8$	$[1,5,6,4] = Z_6$
$Z_7 = [1,5,4]$	$[1,2,4] = Z_8$	$[1,5,6,4] = Z_8$
$Z_8 = [1,2,4]$	$[1,2,3,4] = Z_5$	$[1,5,4] = Z_7$

Now construct DFA.

(1)

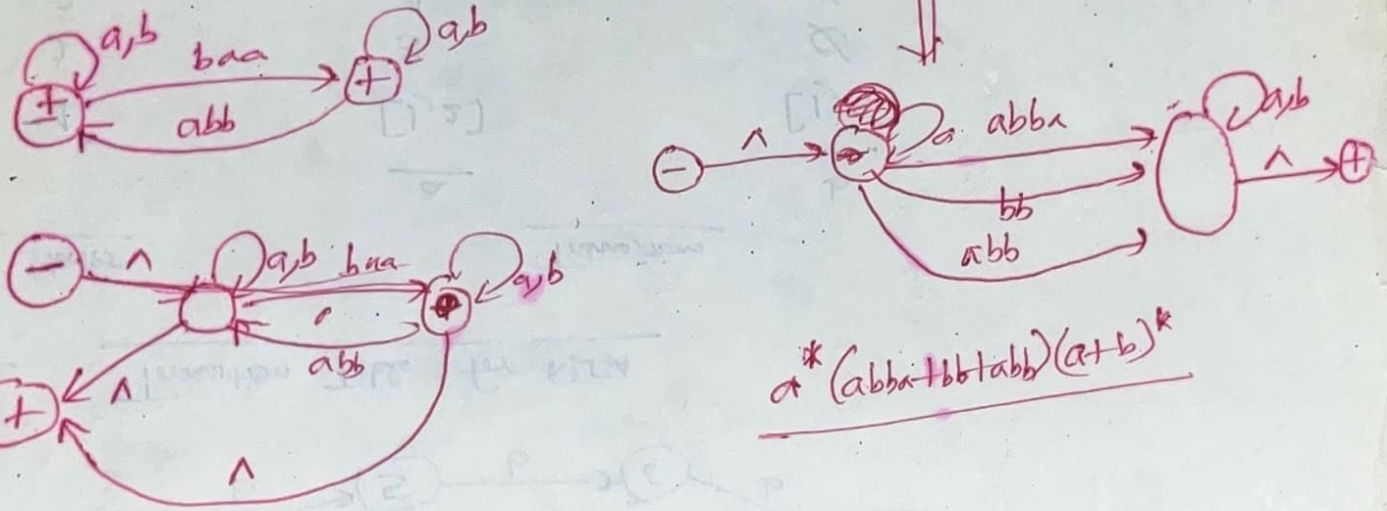


(2)

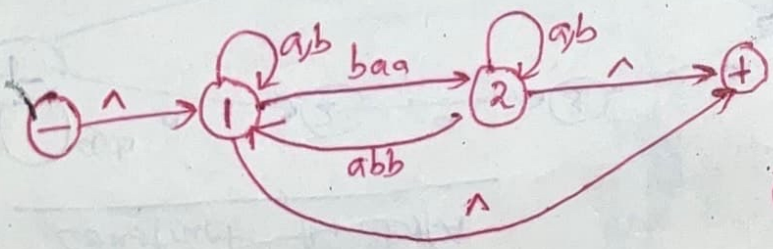


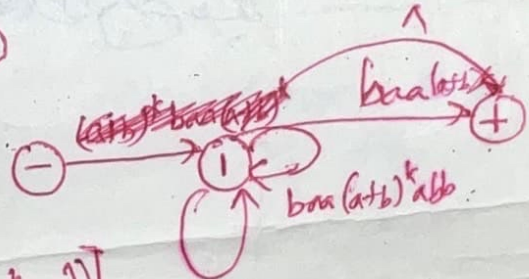
$ab\,[a + baaab + abbbb]^* ab$

(3)



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





$[(a+b) + baa(a+b)^*abb]^*((baa(a+b)^*abb)^*+\wedge))]$