Theory of Computation Notes

Contributor: Deepali

[Bhaskaracharya (DU)]

Computer Science Notes

Page No.
Date

Languages (ch-2) – Cohen

Languages :-

finite set of non-empty string.

Or

Given an alphabet Σ, we wish to define a language in which any string of letters from Σ is a word, even the null string.

Or

If Σ is an alphabet & $L \subseteq \Sigma^*$ then L is a language

Operations on languages :-

- Complementation → $(\bar{L})$
  $\bar{L} = \Sigma^* - L$
- Union → $L_1 \cup L_2$
- Intersection → $L_1 \cap L_2$
- Concatenation → $L_1 \cdot L_2$
- Reversal → L^r
- Kleene's closure → L^*
- Positive closure → $L^+ = L^* - \wedge$

Length of string →

No of letters in a string.

e.g.

If $C = 428$, length $(c) = 3$

length $(\wedge) = 0$

Reverse →

If a is word in some language L then reverse (a) is the same string of letters spelled backward, called the reverse of a, even if this backward string is not a word in L.

e.g. reverse (542) = 245

reverse (140) = 041 ∉ L

Page No.
Date

Theorem :-

$$S^* = S^{**}$$

Proof :-

As we know that every word in S^{**} is made up of factors from S^*. And every word in S^* is made up of factors from S. Thus, every word in S^{**} is made up of factors from S. Therefore, every word in S^{**} is also a word in S^*.
We can write this as

$$S^{**} \subseteq S^* \quad \text{— (1)}$$

Now, in general it is true that for any set A we know that $A \subseteq A^*$, since in A^*, we can choose a word any one factor from A. So, if we consider A to be our set S^*, then

$$S^* \subseteq S^{**} \quad \text{— (2)}$$

From (1) & (2), we get

$$S^* = S^{**}$$

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 ?

Sol ⇒ Words of length 2 :

(aa, bb, ab, ba) i.e., 4 words (2^2)

Words of length 3 :

(aaa, bbb, aba, abb, bab, bba, aab, baa)

i.e., 8 words. (2^3)

∴, words in length n = 2^n words.

Page No.
Date

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?

Sol ⇒ Words of length 2: aa, bb
(Add aa to all words of length 0)
(Add b to all words of length 1)

Words of length 3: aab, baa, bbb
(Add b to all words of length 2)
(Add aa to all words of length 1)

Words of length 4:
(Add aa to all words of length 2)
(Add b to all words of length 3)

(aaaa, bbaa, aabb, baab, bbbb) ~~bbbb) baa~~

Total words in length 4 = 5 words

Words of length 5:
(Add aa to all words of length 3)
(Add b to all words of length 4)

(aaaab, aabaa, baaaa, aabbb, bbbaa, bbaab, baabb, bbbbb)

Total words in length 5 = 8 words

Words of length 6:
(Add aa to all words of length 4)

Page No.
Date

(Add b to all words of length 5)

(aaaaaa, aaaabb, bbaaaa, aabaab, baabaa, aabbaa, bbaabb, aabbbb, bbbbaa, baaaab, bbbbbb, baabbb, bbbaab)

Total words = 13 words

In general, words of length n can be formed by adding aa to words of length (n-2) and b to words of length (n-1). This is the Fibonacci Sequence.

Question → Exercise – Q-1 to 6, 9, 11

Page No.
Date

Regular Expression (Ch-4)

Regular Expression :-

Regular Expression can be thought of as the algebric description of a regular language. Regular languages are the languages that can be generated from one-element languages by applying certain standard operations a finite no of times.

The set of regular expression is defined by the following rules :-

Rule 1 :- Every letter of Σ can be made into a regular expression by writing it in boldface; $\wedge$ is a regular expression.

Rule 2 :- If r_1 and r_2 are regular expression then (r_1), $r_1 r_2$, $r_1 + r_2$, r_1^* are also regular expression.

Rule 3 :- Nothing else is a regular expression.

Q ⇒ L = { aa, aba, abba, abbba, ... }

R.E = ab^*a

Q ⇒ L = { ∧, a, b, aa, bb, aaa, aab, abb, bbb, aaaa, ... }

R.E = a^*b^* [≠ $(ab)^*$]

Q ⇒ L = { a, c, ab, cb, abb, cbb, ... }

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

Q ⇒ L = { aaa, aab, aba, abb, baa, bab, bbb, bba }

Page No.
Date

R.E ⇒ (a+b)(a+b)(a+b)

Q⇒ L = {a, ab, abb, abbb, ... }
R.E ⇒ ab*

Q⇒ L = {∧, ab, abab, ababab, ... }
R.E ⇒ (ab)*

Q⇒ L = {a, aa, aaa, aaaa, ... }
R.E ⇒ a+

Q⇒ L = {abba, baaa, bbbb} — finite
R.E ⇒ abba + baaa + bbbb

Q⇒ L = {∧, a, aa, bbb} — finite
R.E ⇒ ∧ + a + aa + bbb

Q⇒ L = {∧, a, aa, bbb, bbbb, bbbbb, ... }
R.E ⇒ ∧ + a + aa + bbb+

Q⇒ exactly 2 no. ⇒ length = 2. ← language
R.E ⇒ (a+b)(a+b)

Q⇒ exactly 3 ← lang. of
R.E ⇒ (a+b)(a+b)(a+b)

Q⇒ atleast 2 length ← lang. of
R.E ⇒ (a+b)(a+b)(a+b)*

Q⇒ atmost 2 length ← lang. of
R.E ⇒ (a+b+∧)(a+b+∧)

lang. of

Q⇒ even length string.

R.E ⇒ $[(a+b)(a+b)]^*$

lang. of

Q⇒ odd length string

R.E ⇒ $(a+b)[(a+b)(a+b)]^*$

lang. of

Q⇒ length is divisible by 3 ⇒ $|w| = 0 \pmod 3$

R.E ⇒ $[(a+b)(a+b)(a+b)]^*$

Q⇒ $|w| = 2 \pmod 3$

when length is divided by 3 & we get 2 remainder.

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

Q⇒ language of exactly 2 a's.

R.E ⇒ $b^* a\, b^* a\, b^*$

Q⇒ language of atleast 2 a's.

R.E ⇒ $(a+b)^* a\, (a+b)^* a\, (a+b)^*$

Q⇒ language of atmost 2 a's.

R.E ⇒ $b^*\ (a+\wedge)\ b^*\ (a+\wedge) b^*$

Q⇒ $|w|_b = 0 \pmod 2$ ⇒ (length of b is divide by 2).

R.E ⇒ $(a^* b\ a^* b)^* a^*$

Q⇒ $|w|_a = 1 \pmod 3$

R.E ⇒ $(b^* a\ b^* a\ b^* a)^*\ b^* a b^*$

Q⇒ $|w|_b = 2 \pmod 3$

R.E ⇒ $(a^* b\ a^* b\ a^* b)^*\ a^* b\ a^* b\ a^*$

Date

Q=> language of even no of a's.
R.E => $(b^*ab^*a)^*b^*$

Q=> language that starts with a
R.E => $a(a+b)^*$

Q=> language that ends with a
R.E => $(a+b)^*a$

Q=> language that contain a.
R.E => $(a+b)^*a(a+b)^*$

Q=> language of all words that starts & ends with diff. symbols.
R.E => $a(a+b)^*b + b(a+b)^*a$

Q=> language that starts & ends with same symbols.
R.E => $a(a+b)^*a + b(a+b)^*b$

Q=> language of all words starts with a & end with b.
R.E => $a(a+b)^*b$.

Q=> language of odd no of a's.
R.E => $(b^*ab^*a)^*b^*ab^*$

Q=> language of all words that have atleast one a & one b.
R.E => $(a+b)^*a(a+b)^*b(a+b)^* + (a+b)^*b(a+b)^*a(a+b)^*$

Q=> language that have exactly 2 consecutive a's.
R.E => $(ab)^*aa(ba)^*$

Q⇒ language that have two consecutive a's.
R.E ⇒ $(a+b)^* aa (a+b)^*$

Q⇒ L = { ∧, a, b, ab, bb, abb, bbb, ... }
R.E ⇒ $(\wedge+a)b^*$

Q⇒ lang of [Even no of a's & even no of b's [EVEN-EVEN]
R.E ⇒ $[aa+bb+(ab+ba)(aa+bb)^*(ab+ba)]^*$

Q⇒ lang. of odd no of a's & odd no of b's.
R.E ⇒ [ODD-ODD]

R.E ⇒ EVEN-EVEN (ab+ba) EVEN-EVEN. Or
(ab+ba) [Even-Even]

Q⇒ lang. of even no. of a's & odd no of b's.
R.E ⇒ b [EVEN-EVEN] + a [ODD-ODD]

Q⇒ lang. in which 3rd symbol from left end is b.
R.E ⇒ $(a+b)(a+b)\,b\,(a+b)^*$

Q⇒ lang. of all strings that do not end with double letter.
R.E ⇒ $(a+b)^*(ab+ba) + a + b + \wedge$

Q⇒ lang. of all words without double a.
R.E ⇒ $b^*(abb^*)^*(\wedge+a)$

Q⇒ lang. of all words which do not have substring ab.
R.E ⇒ b^*a^*

Page No.
Date

Product set of strings of letters.

If S & T are sets of string of letters, we define the product set of string of letters -
ST = { all combinations of a string from S concatenated with a string from T }

Q => If S = { a, aa, aaa }
T = { bb, bbb }

ST = { abb, abbb, aabb, aabbb, aaabb, aaabbb }

Exercise Question - (Q-1 to 11) - ch-4 - Cohen.

Finite Automata (Ch-5)

(more than 1 finite state possible)

Finite Automata :-

A finite automata is a collection of three things :-

1. A finite set of states, one of which is designated as the initial state, called start state, and some of which are designated as final state.
2. An alphabet Σ of possible input letters, from which are formed strings, that are to be read one letter at a time.
3. A finite set of transitions that tell for each state & for each letter of the input alphabet which state to go to next.

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

where

Q = Total no of states $\{q_0, q_1, \ldots\}$

q_0 = initial state

$\Sigma = \{a, b\}$ = (alphabets on which it works)

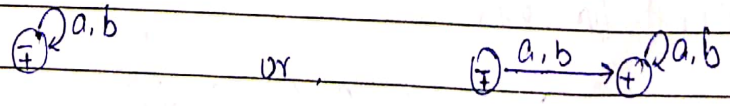
δ = transition function

$$Q \times \Sigma \to Q$$

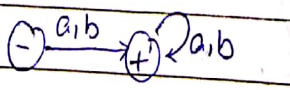
Initial state → (−), >○

Final state → (+), ◎

Q⇒ FA for $(a+b)^*$

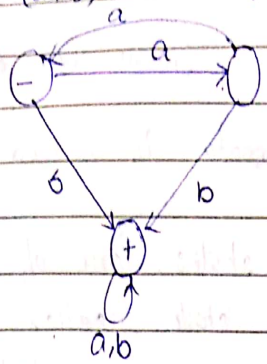


Q⇒ FA for $(a+b)^+$

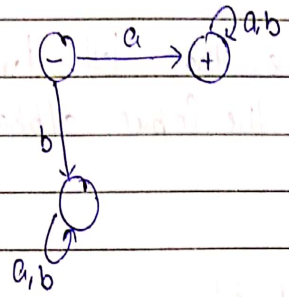


Page No.
Date

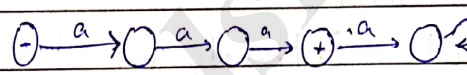
Q⇒ F.A for $(a+b)^* b (a+b)^*$



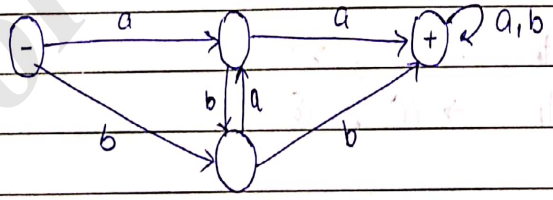
Q⇒ F.A for R.E → $a(a+b)^*$



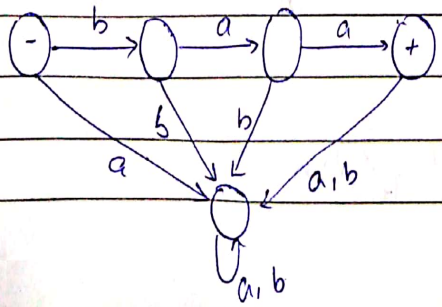
Q⇒ F.A for R.E → aaa. $\Sigma = \{a\}$



Q⇒ F.A for $(a+b)^* (aa+bb) (a+b)^*$

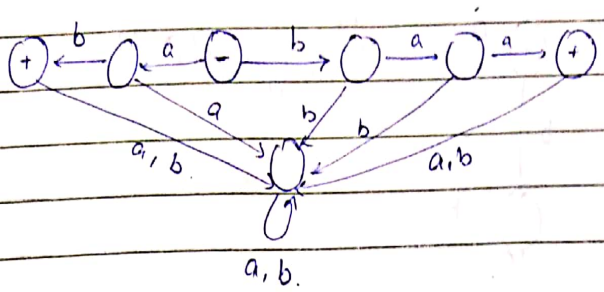


Q⇒ F.A for baa

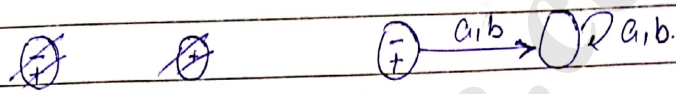


Page No.
Date

Q⇒ FA for baa, ab

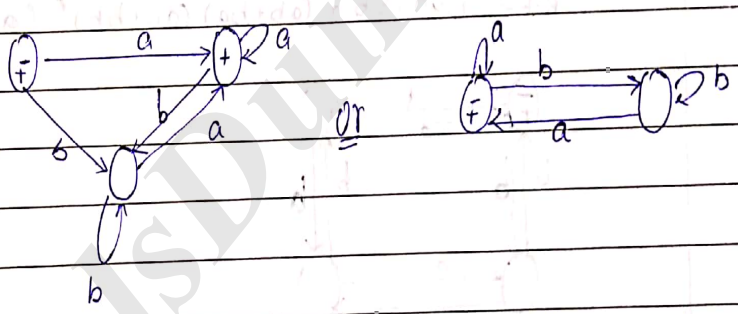


Q⇒ FA for ∩, Σ = {a, b}

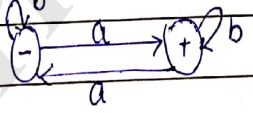


Q⇒ FA for langs ends with a

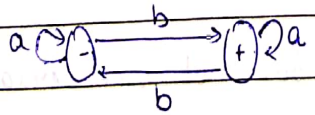
R.E = (a+b)* a.



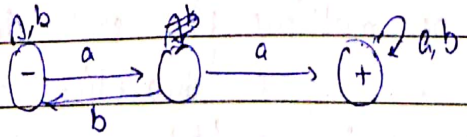
Q⇒ FA for b*a b*(ab*ab*)* — odd no of a's.



Q⇒ FA for odd no of b's.



Q⇒ FA for (a+b)* aa (a+b)*



TutorialsDuniya.com

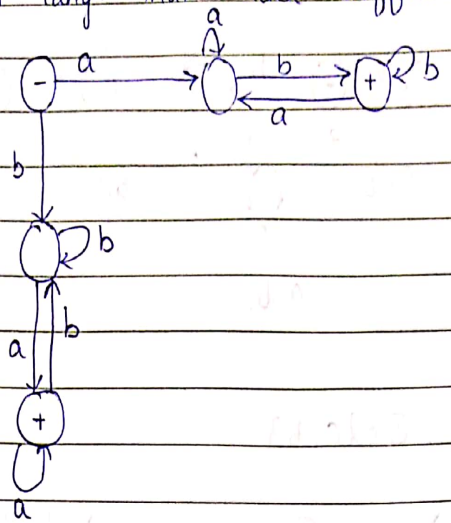
Download FREE Computer Science Notes, Programs, Projects, Books PDF for any university student of BCA, MCA, B.Sc, B.Tech CSE, M.Sc, M.Tech at https://www.tutorialsduniya.com

- Algorithms Notes
- Artificial Intelligence
- Android Programming
- C & C++ Programming
- Combinatorial Optimization
- Computer Graphics
- Computer Networks
- Computer System Architecture
- DBMS & SQL Notes
- Data Analysis & Visualization
- Data Mining
- Data Science
- Data Structures
- Deep Learning
- Digital Image Processing
- Discrete Mathematics
- Information Security
- Internet Technologies
- Java Programming
- JavaScript & jQuery
- Machine Learning
- Microprocessor
- Operating System
- Operational Research
- PHP Notes
- Python Programming
- R Programming
- Software Engineering
- System Programming
- Theory of Computation
- Unix Network Programming
- Web Design & Development

Please Share these Notes with your Friends as well

Date

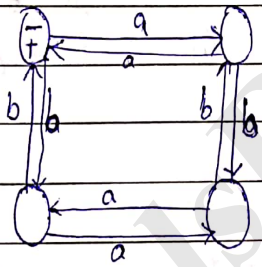
Q⇒ FA for lang. that have different first & last letter.



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

Q⇒ FA for even-even.

R.E⇒ $[aa+bb+(ab+ba)(aa+bb)^*(ab+ba)]^*$



FA

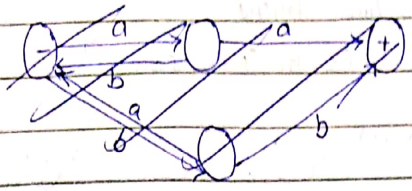
DFA NFA

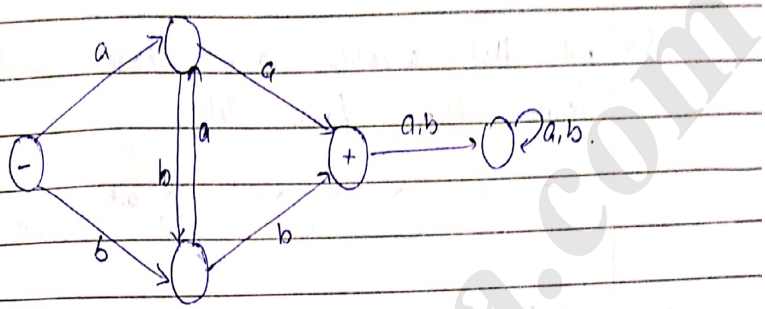
(Deterministic finite automata) (Non-deterministic finite automata)

- Every DFA is NFA but not vice-versa
- Both NFA & DFA have same power
- Each NFA can be translated into DFA.
- There can be multiple final states in both DFA & NFA.
- NFA is more of a theoretical concept.
- DFA is used in lexical analysis in compiler.

Page No.
Date

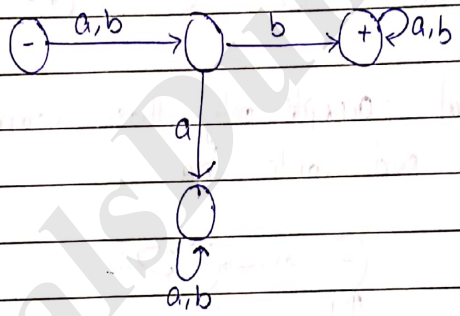
Q⇒ FA for lang. of all words that end in double letter.



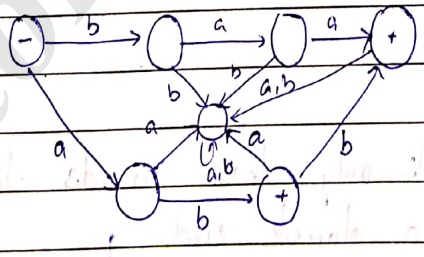


Exercise Ques

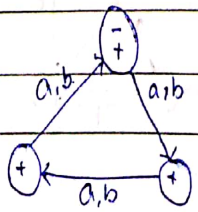
Q⇒ F.A that accepts only lang. of all words with b as second letter.



Q⇒ F.A that accepts baa, ab, abb.



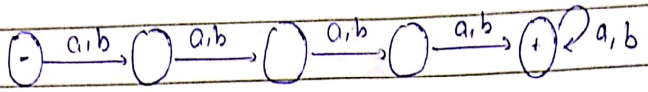
Q⇒ F.A with three states that accepts all strings.



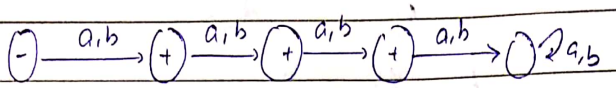
Date

Q⇒ FA that accepts only those words that have more than four letters.

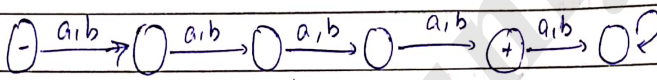
R.E ⇒ $(a+b)(a+b)(a+b)(a+b)^+$



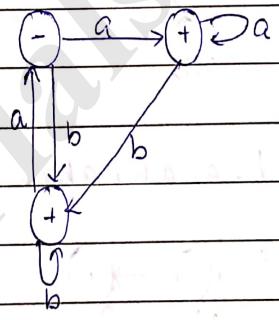
Q⇒ FA that accepts only those words that have fewer than four letters.



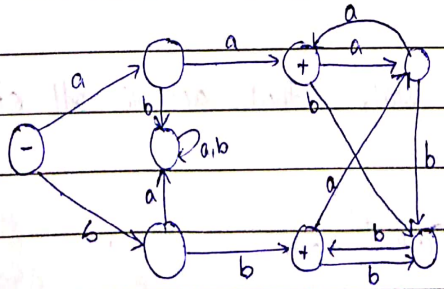
Q⇒ FA that accepts only those words with exactly four letters.



Q⇒ FA that accepts only those words that do not end with ba.

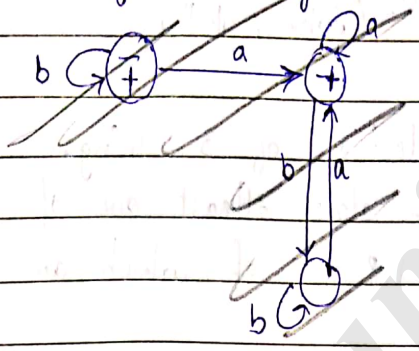


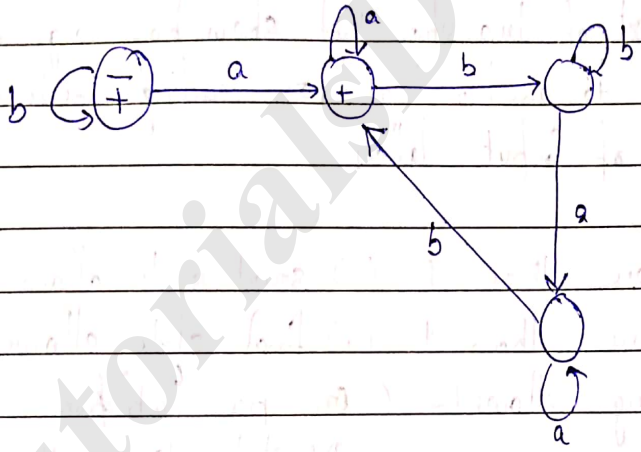
Q⇒ FA that accepts only those words that begin or end with a double end.



Page No.
Date

Q=> FA that accepts only those words that have an even no of substring ab.





Page No.
Date

Transition Graph (ch-5)

Transition Graph → (Invented by John Myhill)

A TG is a collection of 3 things:-

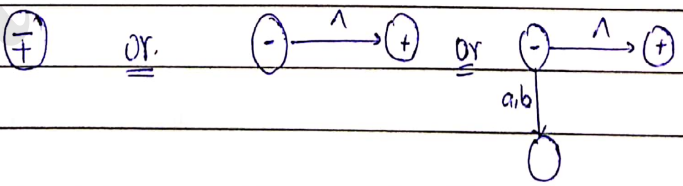
1. A finite set of states atleast one of which is designated as start state (-) & some of which are designated as final state (+).
2. An alphabet Σ.
3. A finite set of transition that show how to go from one state to another based on reading specified substrings of input letters.

TG → more than 1 (-) start state allowed
→ more than 1 (+) final state allowed
→ string allowed — (can pass 'ab' from one state to another)

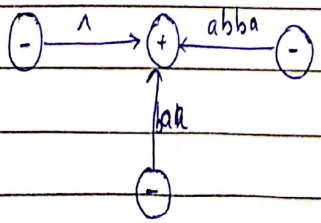
not necessary to show path of every Σ alphabet of Σ.

Q⇒ TG that do not accept 'a' not even null.

Q⇒ TG that accepts only null



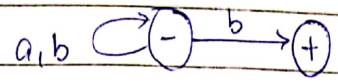
Q⇒ TG that accepts ∧, abba, baa.



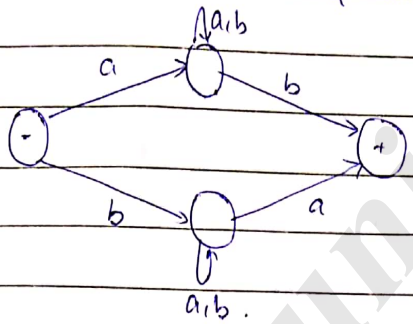
Page No.
Date

Q⇒ TG that ends with b.

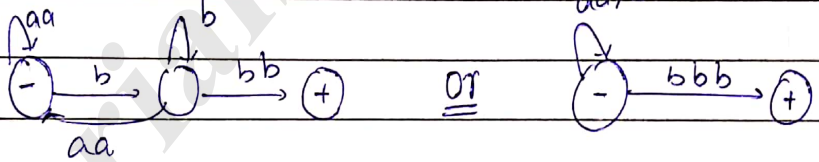
R.E ⇒ $(a+b)^* b$.



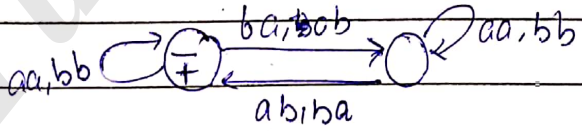
Q⇒ TG that starts with 'a' & ends with 'b' & (vice-versa)



Q⇒ TG that ~~occurs~~ accepts 'a' occurs only in even terms & ending with 3 or more 'b'



Q⇒ TG for EVEN-EVEN

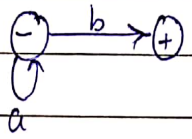


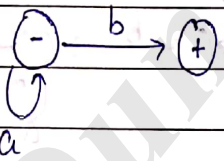
Kleene's Theorem (ch-5)

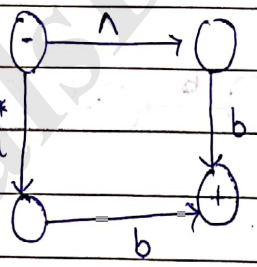
Kleene's Theorem states that :-

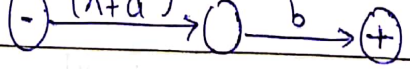
1. Every language that can be defined by FA can also be defined by TG.
2. Every language that can be defined by TG can also be defined by RE.
3. Every language that can be defined by RE can also be defined by FA.

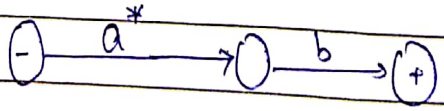
Conversion of TG to RE.

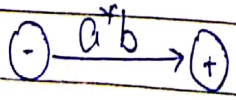
Q⇒ Convert TG  into RE.

Sol:- Step 1 :- 

Step 2 :- 

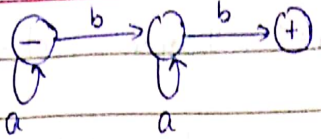
Step 3 :- 

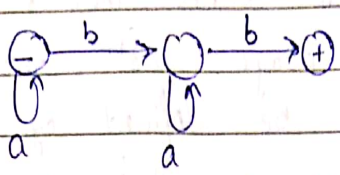
Step 4 :- 

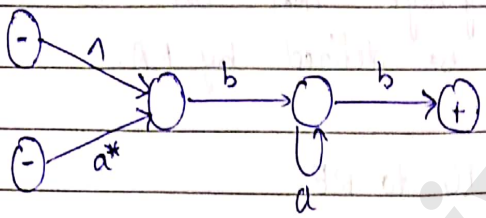
Step 5 :- 

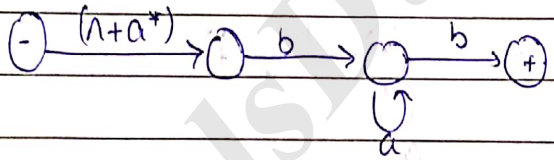
Thus, R.E ⇒ a^*b.

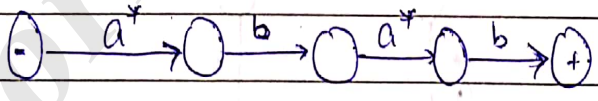
Page No.
Date

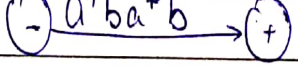
Q ⇒ Convert TG  into R.E.

Sol ⇒ Step 1:- 

Step 2:- 

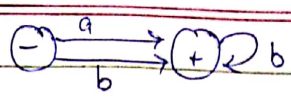
Step 3:- 

Step 4:- 

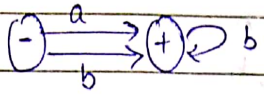
Step 5:- 

Thus, R.E ⇒ a^*ba^*b

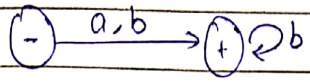
Page No.
Date

Q⇒ Convert TG  into R.E.

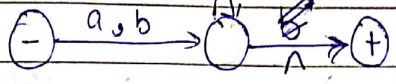
Sol. Step 1:-



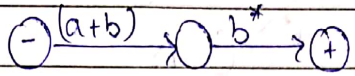
Step 2:-



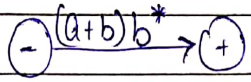
Step 3:-



Step 4:-

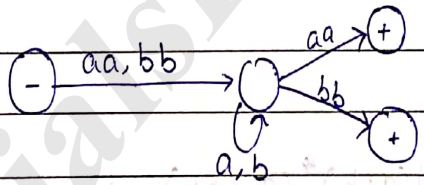


Step 5:-

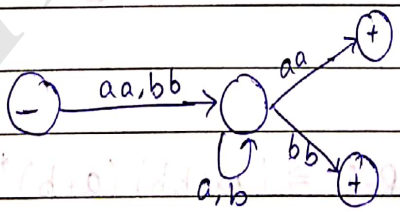


Thus, R.E ⇒ $(a+b)b^*$

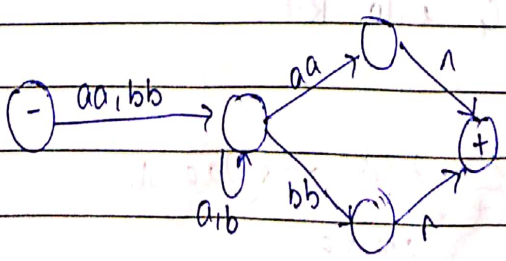
Q⇒ Convert TG into R.E.



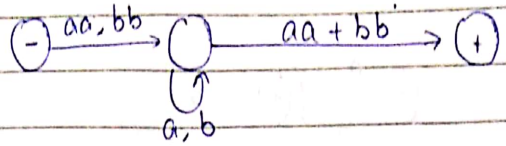
Sol⇒ Step 1:-



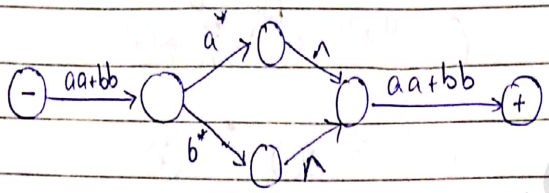
Step 2:-



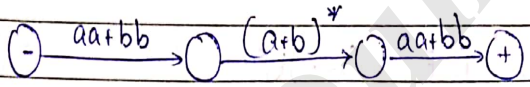
Step 3:-



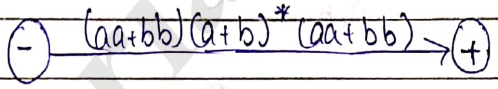
Step 4:-



Step 5:-

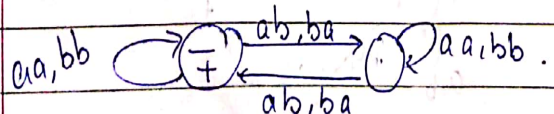


Step 6:-



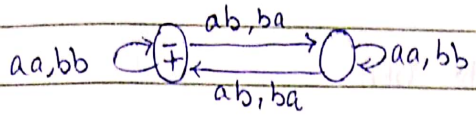
Thus, R.E ⇒ (aa+bb)(a+b)*(aa+bb)

Q⇒ Convert TG into R.E.

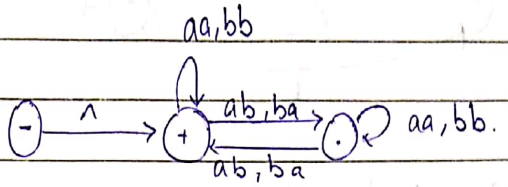


Page No.
Date

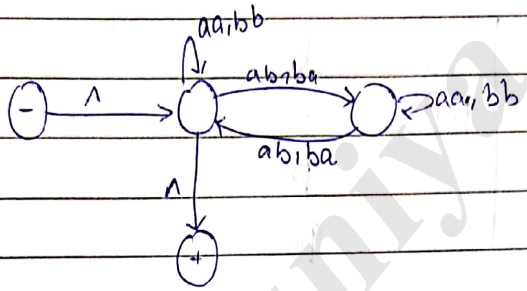
Sol Step 1:-



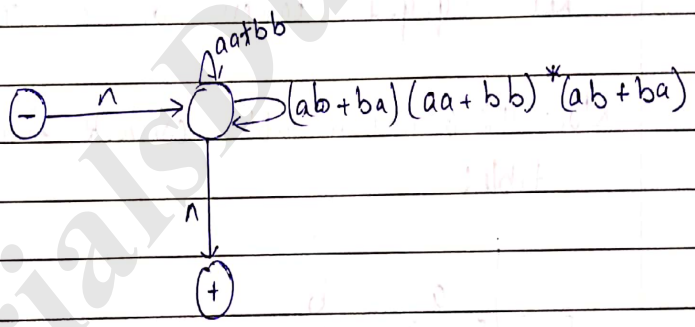
Step 2:-



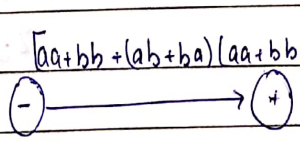
Step 3:-



Step 4:-



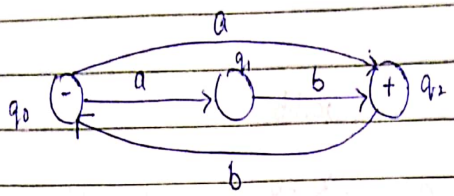
Step 5:-



Thus, R.E = $[aa + bb + (ab + ba)(aa + bb)^*(ab + ba)]^*$

Conversion of NFA to DFA.

Q⇒ convert NFA to DFA.



Sol:- Transition table of NFA :-

	a	b
(−) q_0	$\{q_1, q_2\}$	–
q_1	–	q_2
(+) q_2	–	q_0

Let $[q_1 q_2]$ taken as a single state.

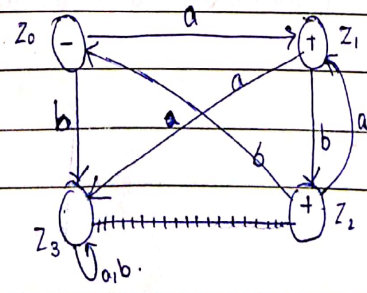
Now, this transition table converts into new transition table :-

	a	b
(−) $[q_0]$	$[q_1 q_2]$	ϕ
(+) $[q_1 q_2]$	ϕ	$[q_2 q_0]$
(+) $[q_2 q_0]$	$[q_1 q_2]$	$[q_0]$
ϕ	ϕ	ϕ

	a	b
− Z_0	Z_1	Z_3
+ Z_1	Z_3	Z_2
+ Z_2	Z_1	Z_0
Z_3	Z_3	Z_3

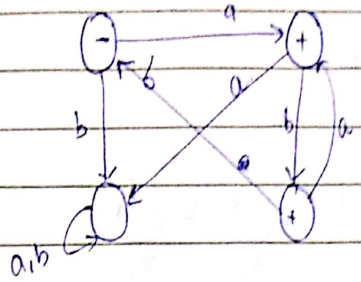
Let $[q_0]$, $[q_1 q_2]$, $[q_2 q_0]$, ϕ are Z_0, Z_1, Z_2, Z_3 respectively.

DFA :-

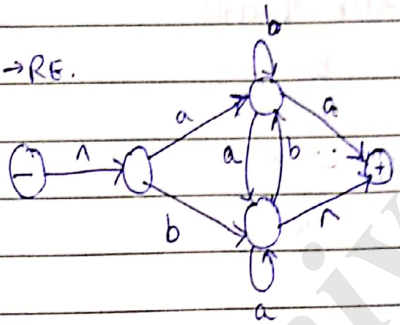


Page No.
Date

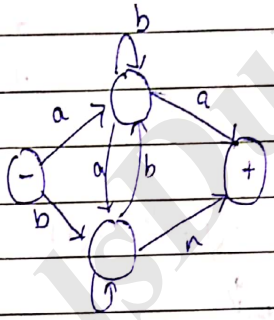
Now, the final DFA is :-



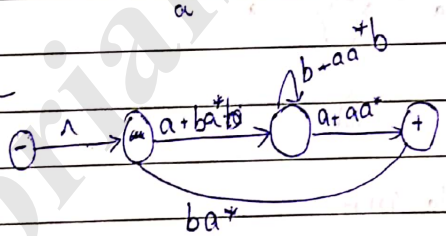
Q⇒ Convert TG → RE.



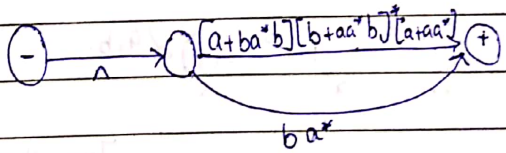
Sol⇒ Step 1:-



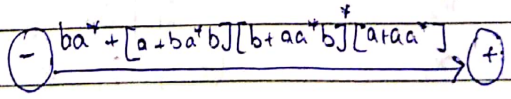
Step 2:-



Step 3:-



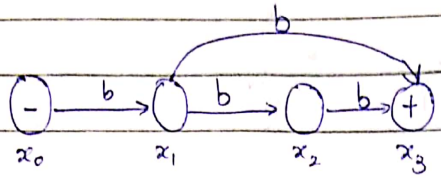
Step 4:-



Page No.
Date

Regular Language

Q⇒ Convert NFA to DFA:-



Sol. Transition table of NFA:-

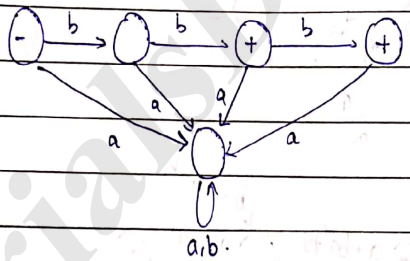
	a	b
(-) x_0	–	x_1
x_1		$[x_2, x_3]$
x_2	–	x_3
(+) x_3	–	–

New transition table

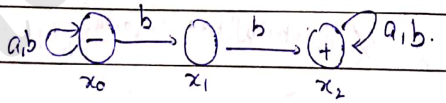
where $x_0 \to z_0$ $x_3 \to z_3$
$x_1 \to z_1$
$x_2, x_3 \to z_2$

	a	b
– z_0	–	z_1
z_1	–	z_2
+ z_2	–	z_3
+ z_3	–	–

DFA:-



Q⇒ Convert NFA to DFA



Sol. Transition table of NFA:-

	a	b
(-) x_0	x_0	$[x_0 x_1]$
x_1	–	x_2
(+) x_2	x_2	x_2

Transition table of DFA:-

	a	b
(-) z_0	z_0	z_1
z_1	z_0	z_2
(+) z_2	z_2	z_2

$z_0 \to x_0$
$z_1 \to x_0$ or x_1
$z_2 \to x_2$ or x_2

TutorialsDuniya.com

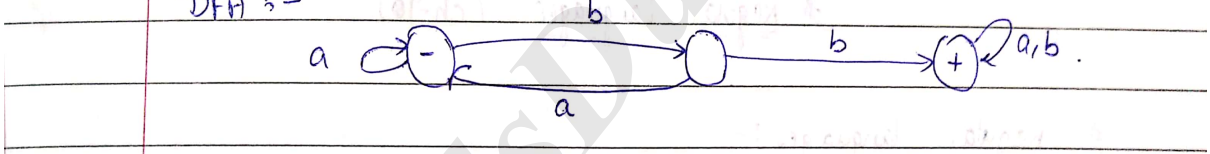
Download FREE Computer Science Notes, Programs, Projects, Books PDF for any university student of BCA, MCA, B.Sc, B.Tech CSE, M.Sc, M.Tech at https://www.tutorialsduniya.com

- Algorithms Notes
- Artificial Intelligence
- Android Programming
- C & C++ Programming
- Combinatorial Optimization
- Computer Graphics
- Computer Networks
- Computer System Architecture
- DBMS & SQL Notes
- Data Analysis & Visualization
- Data Mining
- Data Science
- Data Structures
- Deep Learning
- Digital Image Processing
- Discrete Mathematics
- Information Security
- Internet Technologies
- Java Programming
- JavaScript & jQuery
- Machine Learning
- Microprocessor
- Operating System
- Operational Research
- PHP Notes
- Python Programming
- R Programming
- Software Engineering
- System Programming
- Theory of Computation
- Unix Network Programming
- Web Design & Development

Please Share these Notes with your Friends as well

Page No.
Date

DFA :-



Date

Regular Languages (ch-9)

Regular languages :-

A language that can be defined by a regular expression is called a regular language.

Theorem :-

If L_1 & L_2 are regular languages, then $L_1 + L_2$, $L_1 \cdot L_2$ and L_1^* are also regular languages.

Proof :-

$L_1 + L_2$ means lang. of all words in either L_1 or L_2.

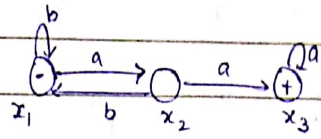
$L_1 \cdot L_2$ means lang. of all words formed by concatenating a word from L_1 with a word from L_2

L_1^* means string that are the concatenation of arbitrarily many factors from L_1.

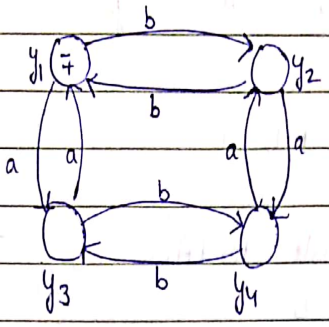
If L_1 and L_2 are regular language, there are regular expression r_1 and r_2 that define these lang. Then $(r_1 + r_2)$ is a regular expression that defines the language $L_1 + L_2$. The language $L_1 L_2$ can be defined by the regular expression $r_1 r_2$. The language L_1^* can be defined by the regular expression $(r_1)^*$. Therefore, all three of these sets of words are definable by regular expressions & so are themselves regular languages. i.e, means $L_1 + L_2$, $L_1 L_2$, L_1^* are regular languages.

Imp # Union

Q.1 ⇒ FA$_1$ =



FA$_2$ =



FA$_1$ + FA$_2$ = ?

Sol ⇒ Transition table for FA$_1$

	a	b
(−) x_1	x_2	x_1
x_2	x_3	x_1
(+) x_3	x_3	x_3

Transition table for FA$_2$

	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

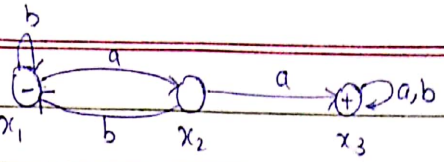
Transition table for FA$_1$ + FA$_2$:-

		a	b
(+)(−)	Z_1 (x_1, y_1)	Z_2	Z_3
	Z_2 (x_2, y_3)	Z_4	Z_5
	Z_3 (x_1, y_2)	Z_6	Z_1
(+)	Z_4 (x_3, y_1)	Z_7	Z_8
	Z_5 (x_1, y_4)	Z_9	Z_{10}
	Z_6 (x_2, y_4)	Z_8	Z_{10}
(+)	Z_7 (x_3, y_3)	Z_4	Z_{11}
(+)	Z_8 (x_3, y_2)	Z_4	Z_4
	Z_9 (x_2, y_2)	Z_{11}	Z_1
	Z_{10} (x_1, y_3)	Z_{12}	Z_5
(+)	Z_{11} (x_3, y_4)	Z_8	Z_7
(+)	Z_{12} (x_2, y_3)	Z_7	Z_3

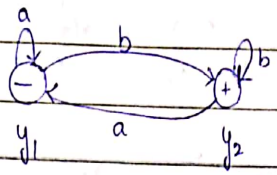
Total states = 12
final states = 6

Page No.
Date

Q = $FA_1 \Rightarrow$



$FA_2 \Rightarrow$



$FA_1 + FA_2 = ?$

Sol. Transition Table for FA_1

	a	b
(-) x_1	x_2	x_1
x_2	x_3	x_1
(+) x_3	x_3	x_3

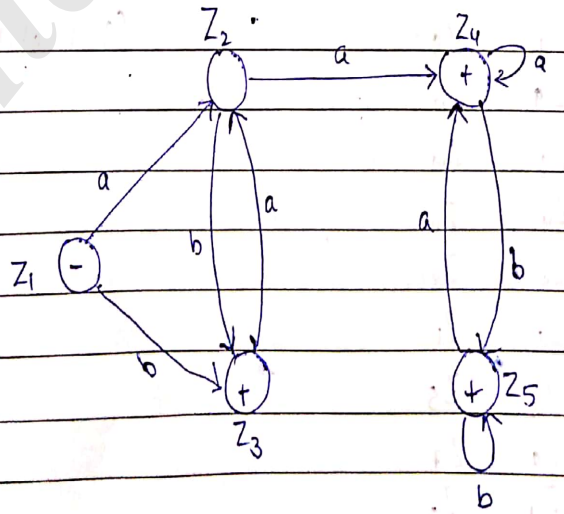
Transition table for FA_2

	a	b
(-) y_1	y_1	y_2
(+) y_2	y_1	y_2

Transition table for $FA_1 + FA_2$:-

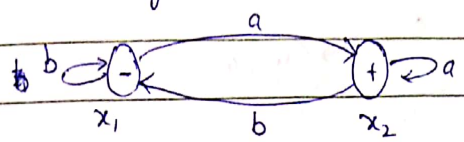
	a	b
(-) Z_1 (x_1, y_1)	Z_2	Z_3
Z_2 (x_2, y_1)	Z_4	Z_3
(+) Z_3 (x_1, y_2)	Z_2	Z_3
(+) Z_4 (x_3, y_1)	Z_4	Z_5
(+) Z_5 (x_3, y_2)	Z_4	Z_5

$FA = FA_1 + FA_2$:→

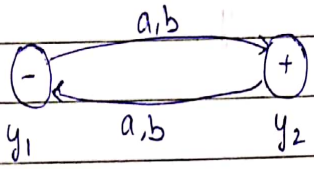


Page No.
Date

Q=> FA_1 = ending with a $(a+b)^*a$



FA_2 = odd no of letters



$FA_1 + FA_2 = ?$

Sol. Transition table for FA_1

	a	b
(-) x_1	x_2	x_1
(+) x_2	x_2	x_1

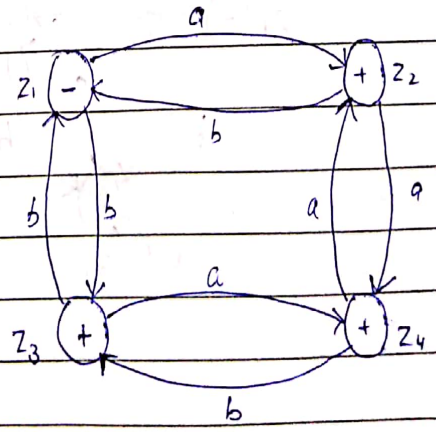
Transition table for FA_2

	a	b
(+) y_1	y_2	y_2
(+) y_2	y_1	y_1

Transition table for $FA_1 + FA_2$:-

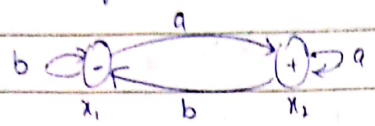
	a	b
(-) Z_1 (x_1, y_1)	Z_2	Z_3
(+) Z_2 $(x_2 y_2)$	Z_4	Z_1
(+) Z_3 (x_1, y_2)	Z_4	Z_1
(+) Z_4 (x_2, y_1)	Z_2	Z_3

FA :-

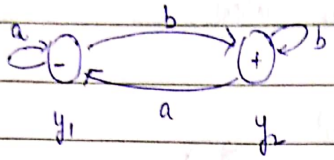


Page No.
Date

Q) FA_1 = ending with a



FA_2 = ending with b



$FA_1 + FA_2$:- ?

Sol) Transition table for FA_1

	a	b
(-) x_1	x_2	x_1
(+) x_2	x_2	x_2

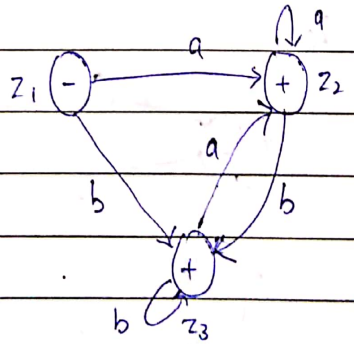
Transition table for FA_2

	a	b
(-) y_1	y_1	y_2
(+) y_2	y_1	y_2

Transition table for $FA_1 + FA_2$:-

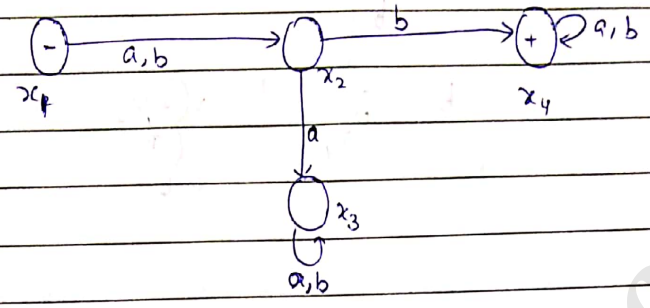
	a	b
(-) $Z_1(x_1, y_1)$	Z_2	Z_3
(+) $Z_2(x_2, y_1)$	Z_2	Z_3
(+) $Z_3(x_1, y_2)$	Z_2	Z_3

$FA_1 + FA_2$:-

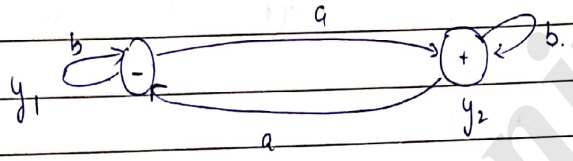


Imp # Concatenation

Q⇒ FA_1 ⇒ (all letters with b as 2nd letter)



FA_2 ⇒ odd no of a's.



$FA_1 \cdot FA_2 = ?$

Sol. Transition table for FA_1

	a	b
(-) x_1	x_2	x_2
x_2	x_3	x_4
x_3	x_3	x_3
(+) x_4	x_4	x_4

Transition table for FA_2

	a	b
(-) y_1	y_2	y_1
(+) y_2	y_1	y_2

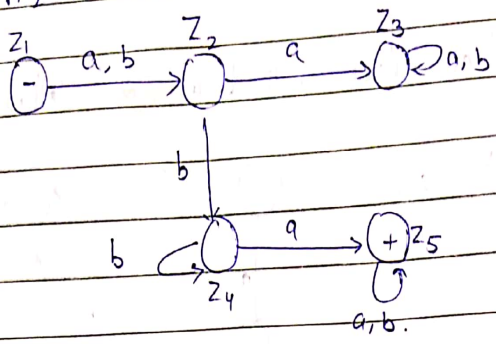
Transition Table for $FA_1 \cdot FA_2$:-

	a	b
(-) $Z_1(x_1)$	Z_2	Z_2
$Z_2(x_2)$	Z_3	Z_4
$Z_3(x_3)$	Z_3	Z_3
$Z_4(x_4 y_1)$	Z_5	Z_4
(+) $Z_5(x_4 y_2)$	~~Z_5~~	Z_5
$(x_4 y_1 y_2)$	Z_5	

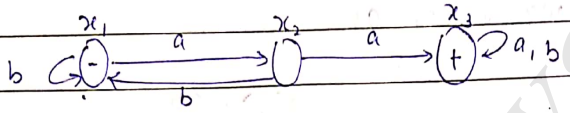
Final state → which contain final state of 2nd F.A.

Page No.
Date

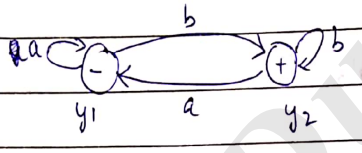
$FA_1 . FA_2 \Rightarrow$



Q ⇒ FA_1 : (double a)



FA_2 : (ending with b)



Sol: Transition table for FA_1

	a	b
(-) x_1	x_2	x_1
x_2	x_3	x_1
(+) x_3	x_3	x_3

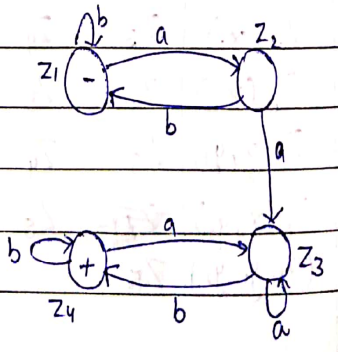
Transition table for FA_2

	a	b
(-) y_1	y_1	y_2
(+) y_2	y_1	y_2

Transition table for $FA_1 . FA_2$:-

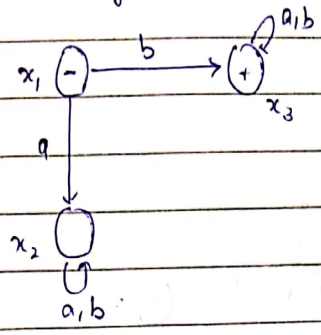
	a	b
(-) Z_1 (x_1)	Z_2	Z_1
Z_2 (x_2)	Z_3	Z_1
Z_3 $(x_3 y_1)$	Z_3	Z_4
(+) Z_4 $(x_3 y_1 y_2)$	Z_3	Z_4

FA :- $(FA_1 . FA_2)$

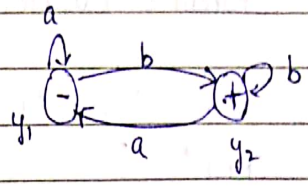


Page No.
Date

Q-> FA_1 = (Starting with b)



FA_2 = (ending with b)



$FA_1 \cdot FA_2$ = ?

Sol. Transition table for FA_1

	a	b
(-) x_1	x_2	x_3
x_2	x_2	x_2
(+) x_3	x_3	x_3

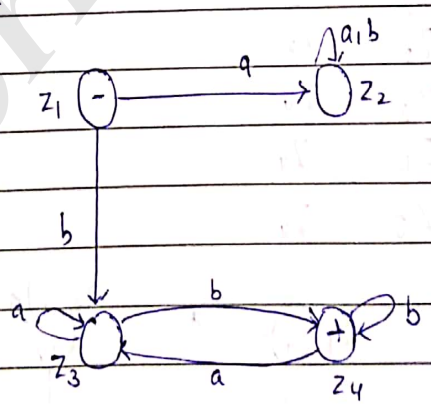
Transition table for FA_2.

	a	b
(-) y_1	y_1	y_2
(+) y_2	y_1	y_2

Transition table for $FA_1 \cdot FA_2$:-

	a	b
(-) $Z_1(x_1)$	Z_2	Z_3
$Z_2(x_2)$	Z_2	Z_2
$Z_3(x_3 y_1)$	Z_3	Z_4
(+) $Z_4(x_3 y_1 y_2)$	Z_3	Z_4

$FA_1 \cdot FA_2$:-



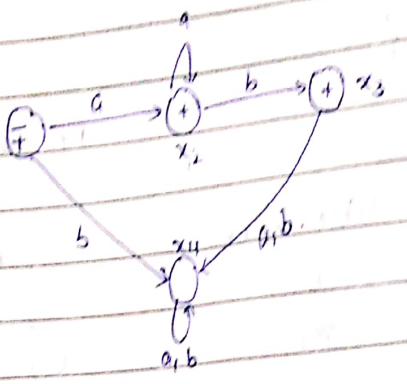
Unit 2

Kleene's closure :-

Q ⇒ $r = a^* + aa^*b$

$(FA)^* = ?$

Sol ⇒ FA =



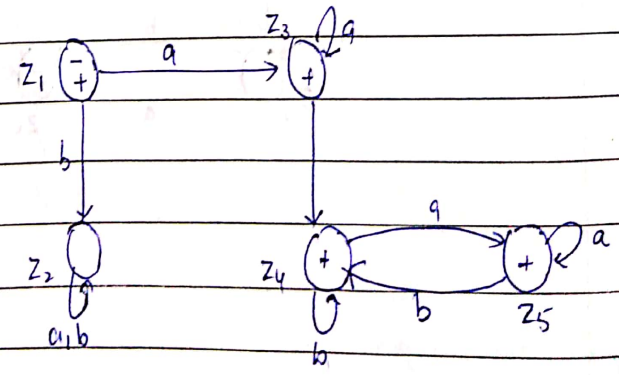
Transition table for FA :-

		a	b
(+)(−)	x_1	x_2	x_4
(+)	x_2	x_2	x_3
(+)	x_3	x_4	x_4
	x_4	x_4	x_4

Transition table for FA^* :-

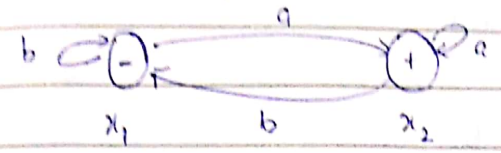
		a	b
(+)(−)	Z_1 (x_1)	Z_3	Z_2
	Z_2 (x_4)	Z_2	Z_2
(+)	Z_3 $(x_1 x_2)$	Z_3	Z_4
(+)	Z_4 $(x_2 x_3 x_4)$	Z_5	Z_4
(+)	Z_5 $(x_2 x_4)$	Z_5	Z_4

FA^* ⇒



Page No.
Date

Q. FA : (ending with a)



$(FA)^* = ?$

Sol Transition table for FA:-

		a	b
(–)	x_1	x_2	x_1
(+)	x_2	x_2	x_1

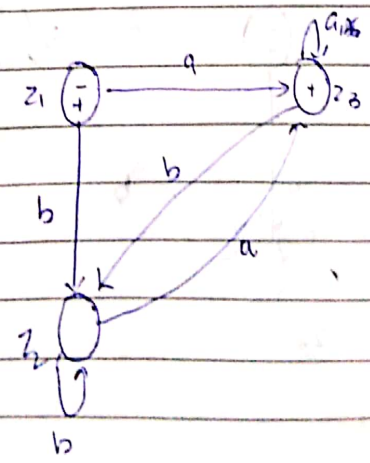
Because in this case Kleene's closure case, there are 2 states in its FA.
Therefore, we take

$Z_1 = x_1$ & a final state
$Z_2 = x_1$ & non-final state
$Z_3 = x_1$ or x_2

Transition Table for FA^* :-

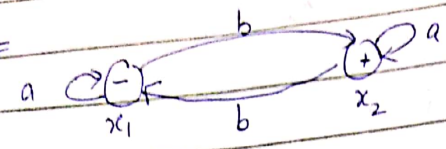
		a	b
(+) (–)	Z_1	Z_3	Z_2
	Z_2	Z_3	Z_2
(+)	Z_3	Z_3	Z_2

$FA^* \Rightarrow$



Q = FA = odd no of b's

FA =



$FA^* = ?$

Sol Transition table for FA:-

	a	b
(−) x_1	x_1	x_2
(+) x_2	x_2	x_1

Let

Z_1 = x_1 & a final state

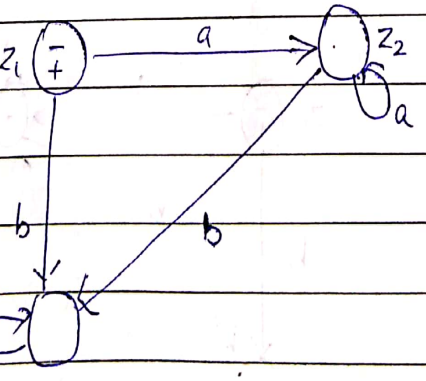
Z_2 = x_1 & non-final state

Z_3 = x_1 or x_2

Transition table for FA^*

	a	b
(±) Z_1	Z_2	Z_3
Z_2	Z_2	Z_3
(+) Z_3	Z_3	Z_3

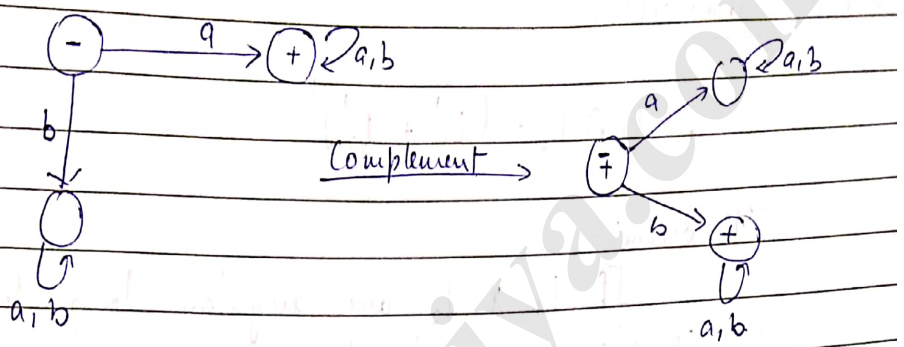
FA^* ⇒



Sub #

Complement

If L is a language over the alphabet Σ, we define its complement, L' to be the language of all strings of letter from Σ that are not words in L.



Theorem:-

If L is a regular language, then L' is also a regular language.

OR

The set of regular languages is closed under complementation.

Proof:-

If L is a regular lang., we know from kleene's theorem that there is some FA that accepts the lang. L. Some of states of this FA are final states, & most likely, some are not. Let us reverse the final states of each state; that is, if it was a final state, make it a non-final state. if it ~~was now ends~~ was a ~~non~~ final state, ~~& vice versa~~ make it ~~non~~ final state. If an input string formerly ended in a nonfinal state, it now ends in final state & vice-versa. This new machine we

have built accepts all input string that were not accept by original FA & rejects all input strings that FA used to accept.
Therefore, this machine accepts exactly the lang. L'. So by Kleen's theorem, L' is regular

Imp # Intersection

$$L_1 \cap L_2 = (L_1' + L_2')'$$

Theorem:-
If L_1 & L_2 are regular lang. then $L_1 \cap L_2$ is also a regular lang.

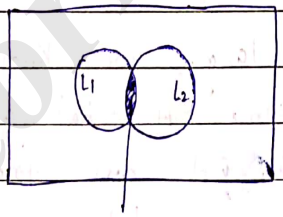
or

The set of all regular languages is closed under Intersection.

Proof:-
By DeMorgan's law for sets of any kind (reg. lang. or not)

$$L_1 \cap L_2 = (L_1' + L_2')'$$

This is illustrated by Venn dia gram.



$$L_1 \cap L_2 = (L_1' + L_2')'$$

This means that the lang. $L_1 \cap L_2$ consists of all words that are not either in L_1' or L_2'.
Because L_1 & L_2 are regular, then so are L_1' & L_2'. Since L_1' & L_2' are regular,

TutorialsDuniya.com

Download FREE Computer Science Notes, Programs, Projects, Books PDF for any university student of BCA, MCA, B.Sc, B.Tech CSE, M.Sc, M.Tech at https://www.tutorialsduniya.com

- Algorithms Notes
- Artificial Intelligence
- Android Programming
- C & C++ Programming
- Combinatorial Optimization
- Computer Graphics
- Computer Networks
- Computer System Architecture
- DBMS & SQL Notes
- Data Analysis & Visualization
- Data Mining
- Data Science
- Data Structures
- Deep Learning
- Digital Image Processing
- Discrete Mathematics
- Information Security
- Internet Technologies
- Java Programming
- JavaScript & jQuery
- Machine Learning
- Microprocessor
- Operating System
- Operational Research
- PHP Notes
- Python Programming
- R Programming
- Software Engineering
- System Programming
- Theory of Computation
- Unix Network Programming
- Web Design & Development

Please Share these Notes with your Friends as well

Page No.
Date

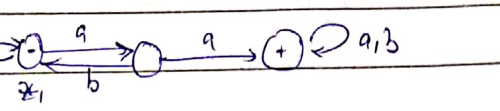
So is $L_1' + L_2'$. And because $L_1' + L_2'$ is regular, then so is $(L_1' + L_2')'$, which means $L_1 \cap L_2$ is regular.

Q ⇒ L_1 = all strings with double a

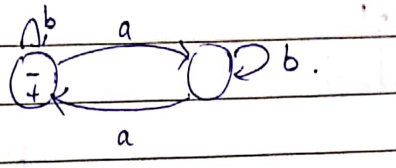
L_2 = even no of a

find complement intersection of L_1 & L_2 & draw FA

Sol. FA_1 =

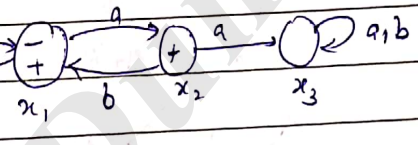


FA_2 =

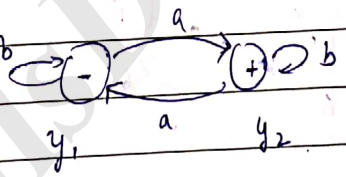


FA_1 to. $(FA_1) \cap (FA_2) = [(FA_1)' + (FA_2)']'$

$\therefore$ FA_1' =

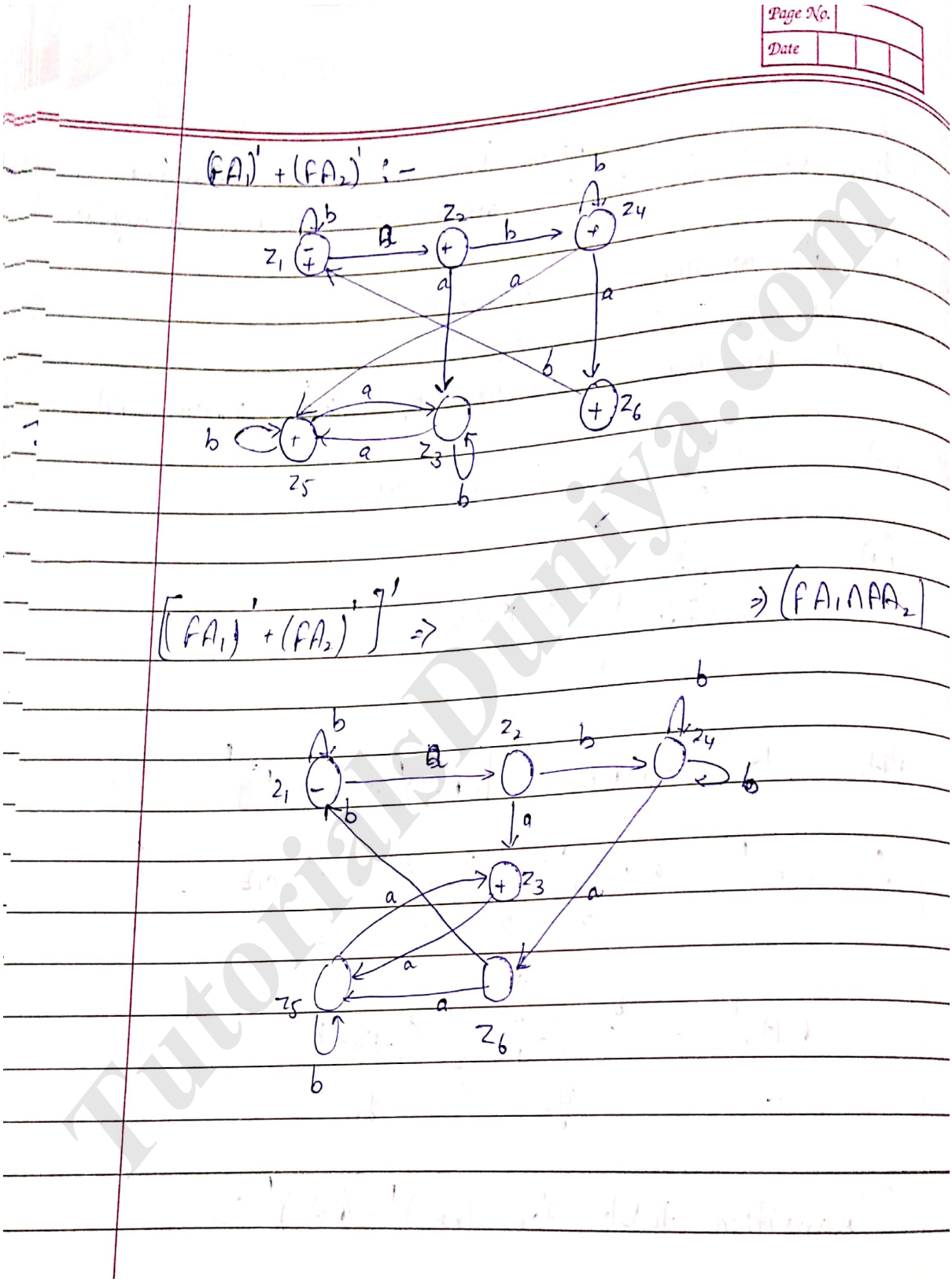


FA_2' =



Transition table for $(FA_1)' + (FA_2)'$:-

	a	b
(+)(-) Z_1 (x_1, y_1)	Z_2	Z_1
(+) Z_2 (x_2, y_2)	Z_3	Z_4
Z_3 (x_3, y_1)	Z_5	Z_3
(+) Z_4 (x_1, y_2)	Z_6	Z_4
(+) Z_5 (x_2, y_2)	Z_3	Z_5
(+) Z_6 (x_2, y_1)	Z_5	Z_1



Page No.
Date

Non-regular languages (ch-10)

Non-regular lang.:-

A language that can't be defined by regular expression is called non-regular languages.

Non-regular languages can't be accepted by FA or TG.

Theorem:- (Pumming Lemma Theorem)

Let L be any regular language that has infinitely many words. Then there exist some three strings x, y, and z (where y is not the null string) such that all strings of form

$$xy^nz \text{ for } n = 1, 2, 3, \ldots$$

are words in L.

Pumming lemma Theorem is used to check whether the language is regular or not.

e.g. $\Sigma = \{a\}$

Use pumming lemma, prove that $L = \{a^n \text{ for } n = 0, 1, 2, \ldots\}$ is regular.

Sol. The pumming lemma says that there must be 3 strings x, y, and z such that all words of the form xy^nz are in L.

Let us take a typical word of L ⇒

$w = aaaaaaa$

Observation 1:- $w = a\,a\,a\,a\,a\,a\,a$

If y string contained aa which

is underlined, then xyyz would contain 4 a's, i.e.

$\Rightarrow xy^n z$

$\Rightarrow aaa(aa)^n aa$ (put n=2)

$\Rightarrow aaaaaaaa$

which is also a word in L.

Observation 2:- w = aaaaaaa

If string y contained 3 a's which are underlined, then acc. to pumming lemma,

$xy^n z \Rightarrow aa(aaa)^n aa$ put n=2

$\Rightarrow aaaaaaaaa$

which is also a word in L.

Thus, a^* is a regular language.

Q⇒ $L = \{a^n : n \text{ is prime}\}$

Use pumming lemma, and check whether L is regular or not.

Sol⇒ The pumming lemma says that there must be strings x, y, z such that all words of the form $xy^n z$ are in L.

let us take a typical string.

w = aaaaa.

Observation 1:- w = aaaaa

If y string contained a, which

is underlined, then $x = aa$ & $z = aa$.

Now,

$xy^nz \Rightarrow aa(a)^n aa$ (put $n=2$)

$\Rightarrow aa\,aa\,aa$

which is not a prime number.

Observation 2:- ~~If~~ $w = aaaaa$

If y string contained 'aaa' which is underlind in above string and

$x = a$, $z = a$

then,

xy^nz

$\Rightarrow a(aaa)^n a$ (put $n=2$)

$\Rightarrow a\,aaaaaa\,a$

which is not a prime number.

Conclusion :- After solving a string with pumming lemma, there exists a string which is not a prime no.

Therefore, this is not a regular language.

Q) $L = \{a^n b^n ; n \geq 1\}$

Use pumming lemma, check whether this lang. is regular or not.

Sol:- The pumming lemma says that there must be strings x, y, z such that all words of form xy^4z are in L.

Let us take a typical string -

$w = aabb$

Observation 1:- $w = aabb$

If y string contained ab, then

$x = a$

$z = b$

or, xy^nz

$\Rightarrow a(ab)^nb$ (put $n=2$)

$\Rightarrow aababb$.

In this case, the resultant string contains a after b. which is not possible according to given language.

Thus. $aababb \notin L$

~~Observation 2:-~~

Hence, this is not a regular language.

Q> Prove that ~~on~~ a^nb^{2n} $n \geq 0$ is not regular by using pumming lemma.

Sol:- $L = \{a^nb^{2n} \; n \geq 0\}$

$L = \{abb, aabbbb, aaabbbbbb, \ldots\}$

The pumming lemma says that there must be strings x, y, z such that all words of the form xy^nz are in L.

Let us take a string of L.

$w = aabbbb$.

Observation 1:- $w = aabbbb$

If string y contained bb which is underlined in above w, then $x = aa$ and $z = bb$.

Then, xy^nz $\quad n>0$

$\Rightarrow aa(bb)^n bb$

After putting $n = 2$, we get,

$\Rightarrow aabbbbbb$

Conclusion 1:- In this string (aabbbbbb), a occurs 2 times and b occurs 6 times. But b can occur only twice of a, i.e. if $a = 2$ then b must be 4. Thus, aabbbbbb does not belong to L.

Observation 2:- $w = aabbbb$

If string y contained abb, then $x = a$ and $z = bb$.

$\therefore xy^nz \quad (n>0)$

$\Rightarrow a(abb)^n bb$.

After putting $n=2$, we get,

$\Rightarrow aabbabbbb$.

Conclusion 2:- In this string (aabbabbbb), a does occurs after b too. which is not possible to be a word in L.

Thus, $aabbabbbb \notin L$.

Hence, ~~the~~ $a^n b^{2n}$ $(n \geq 0)$ is not a regular language.

Page No.
Date

(Context-Free Grammar) (Ch-12)

Context-Free Grammar (CFG) :-

CFG is a collection of three things:-

1. an alphabet Σ of letters called terminals from which we are going to make strings that will be the words of a language.
2. A set of symbols called non-terminals, one of which is the symbol S, standing for "start here".
3. A finite set of production of form
   one non-terminal → finite string of terminals and/or non-terminals.

CFL :-

The language generated by CFG is called CFL.

Q ⇒ Let terminals be a and non-terminals be S and production be

$$\left.\begin{array}{l} S \to aS \\ S \to \Lambda \end{array}\right] \text{- CFG for } a^*$$

Generate ~~CFG for aaaaaa~~ aaaaaa by given CFG.

Sol:- $S \to aS$

$S \to \Lambda$

we can write above CFG as

$S \to aS \mid \Lambda$.

Now generate aaaaaa.

$S \Rightarrow aS$

$\Rightarrow aaS \quad (S \to aS)$

$\Rightarrow aaaS \quad (S \to aS)$

$\Rightarrow aaaaS \quad (S \to aS)$

$\Rightarrow aaaaaS \quad (S \to aS)$

$\Rightarrow aaaaaaS \quad (S \to aS)$

$\Rightarrow aaaaaa\wedge \quad (S \to \wedge)$

$\Rightarrow aaaaaa.$

This is the derivation of a^6 in this CFG.

Q ⇒ Generate CFG for $(a+b)^*$

Sol ⇒ To generate CFG for $(a+b)^*$, let a and b are two terminals and S be the non-terminals and production be :-

$S \to aS$

$S \to bS$

$S \to \wedge$

ie;

$S \to aS \mid bS \mid \wedge$

Let us check this production by generating a string 'ababb'.

To generate ababb., the derivation should be

$S \Rightarrow aS$

$\Rightarrow abS \quad (S \to bS)$

$\Rightarrow abaS \quad (S \to aS)$

$\Rightarrow ababS \quad (S \to bS)$

$\Rightarrow ababbS \quad (S \to bS)$

$\Rightarrow ababb\wedge \quad (S \to \wedge)$

Thus, we can generate ababb or any string of $(a+b)^*$ from this CFG production.

Therefore, CFG should be

$S \to aS \mid bS \mid \wedge$ or $S \to X \mid Y$, $X \to \wedge$
$Y \to aY \mid bY \mid a \mid b$

Q⇒ Generate CFG for $(a+b)^* aa (a+b)^*$

Sol⇒ $S \to XY$
$X \to aX \mid aY \mid a$
$Y \to Ya \mid Yb \mid a$

or $S \to XaaX$
$X \to aX \mid bX \mid \wedge$

Q⇒ Generate CFG for $a^n b^n$

$S \to aSb \mid \wedge$

Q⇒ Generate CFG for Even-palindrome.

$S \to aSa \mid bSb \mid \wedge$

Q⇒ Generate CFG for odd-palindrome

$S \to aSa \mid bSb \mid a \mid b$

Q⇒ Generate CFG for palindrome:-

$S \to aSa \mid bSb \mid a \mid b \mid \wedge$

Q⇒ Generate CFG for $b^* a^*$

$S \to XY$, $X \to bX \mid \wedge$, $Y \to aY \mid \wedge$

Page No.
Date

Derivation Tree

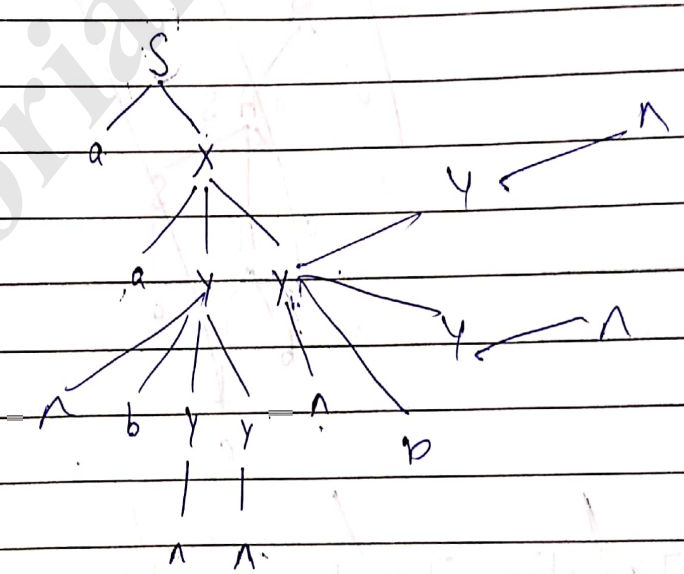
We can define by CFG by making a tree, this tree is derivation tree.

Root Vertex → Start Symbol (S)
Vertex :- Non-terminal Symbols
Leaves :- Terminal Symbols including (∧).

e.g. Let CFG :-

$S \rightarrow aX$
$X \rightarrow aYY$
$Y \rightarrow bYY \mid \wedge$

then tree must be



Page No.
Date

Ambiguous Grammar

A G is ambiguous if we can draw more than 1 distinct derivation tree for G.

Q. $S \rightarrow d \mid a \mid AS$
$A \rightarrow bS$
Prove that it is ambiguous.

Page No.
Date

Q:) Prove that the CFG

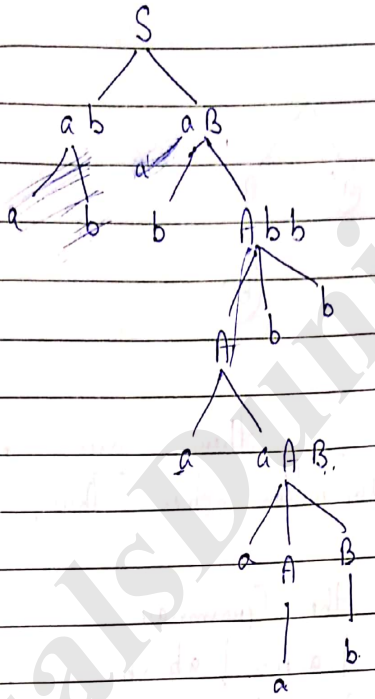
$S \rightarrow aB \mid ab$

$A \rightarrow aAB \mid a$

$B \rightarrow Abb \mid b.$

is ambiguous.

Sol:) Derivation Tree for given CFG is



~~Let us~~ To check whether it is ambiguous or not, let us take a string which can be generated by this CFG.

Let $w = ab$.

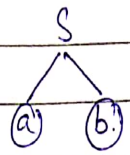
Now, draw all possible derivation trees for this particular string.

$S \Rightarrow ab$ or $S \Rightarrow aB$

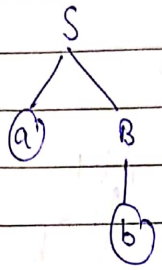
$\Rightarrow ab.$

Page No.
Date

Derivation Tree 1:-



Derivation Tree 2:-



For string 'ab', there are more than one distinct derivation tree. Thus, it is ambiguous.

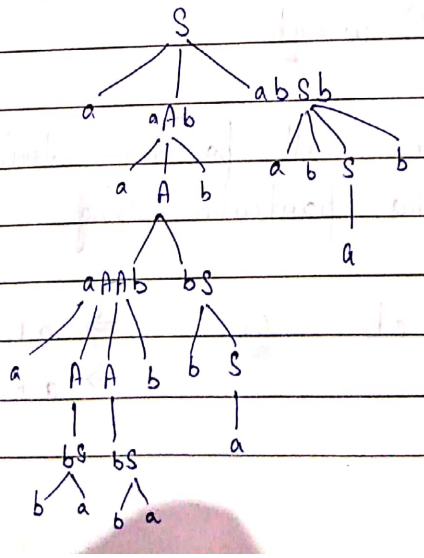
Q ⇒ Prove that the Grammar

S → a | aAb | abSb

A → aAAb | bS

is ambiguous.

Sol ⇒ Derivation Tree for this Grammar is



TutorialsDuniya.com

Download FREE Computer Science Notes, Programs, Projects, Books PDF for any university student of BCA, MCA, B.Sc, B.Tech CSE, M.Sc, M.Tech at https://www.tutorialsduniya.com

- Algorithms Notes
- Artificial Intelligence
- Android Programming
- C & C++ Programming
- Combinatorial Optimization
- Computer Graphics
- Computer Networks
- Computer System Architecture
- DBMS & SQL Notes
- Data Analysis & Visualization
- Data Mining
- Data Science
- Data Structures
- Deep Learning
- Digital Image Processing
- Discrete Mathematics
- Information Security
- Internet Technologies
- Java Programming
- JavaScript & jQuery
- Machine Learning
- Microprocessor
- Operating System
- Operational Research
- PHP Notes
- Python Programming
- R Programming
- Software Engineering
- System Programming
- Theory of Computation
- Unix Network Programming
- Web Design & Development

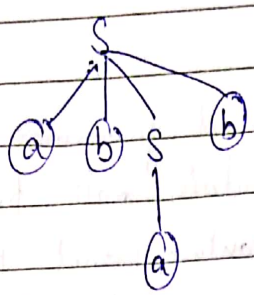
Please Share these Notes with your Friends as well

Page No.
Date

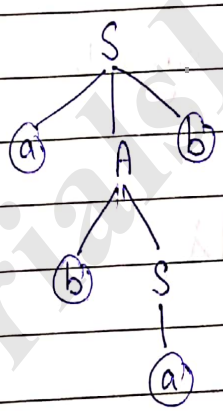
Let's take a string 'abab' to check it is ambiguous or not.

Draw all possible derivation tree for this string.

Derivation Tree 1:-



Derivation Tree 2:-



For string 'abab', there are more than one distinct derivation tree.

Therefore, this grammar is ambiguous.

Date

Normal forms (Ch-7 from 3rd ref.)

Specification of Grammar:-

Reduction of Grammar means simplification of a grammar by the elimination of unnecessary symbols.

Condition for reduced grammar:-

1. Does not contain null production. $(A \to \Lambda)$
2. Does not contain unit production. $(A \to B)$ (non-terminal → non-terminal)
3. Does not contain useless symbols.

Removable of useless symbol

eg

$S \to A$

$A \to aA \mid \lambda$

$B \to bA$

$S \to A$

$A \to aA$

$A \to \lambda$

'B' is useless symbol. — we can't reach to B.

Thus remove prod 3. , we get,

$S \to A$

$A \to aA \mid \lambda$.

This is the simplified grammar after removing useless symbols.

Page No.
Date

Removable of NULL production

Order of form ($A \to \epsilon$ or $A \to \wedge$) are called NULL prod.

Steps to remove nullable variable.

① find all nullable variable.

② Create two versions of all prod. containing nullable variables on their R.H.S, one without & one - with nullable variable

③ Remove ϵ production.

④ Production whose R.H.S does not have any nullable variable are included directly.

e.g. $S \to aS \mid bS \mid \epsilon$, eliminate ϵ.

$S \to aS$ — ①
$S \to bS$ — ②
$S \to \epsilon$ — ③

Step 1:- find all nullable variable.

S.

Step 2:- creating 2 versions.

(i) $S \to aS$ → $S \to aS$, $S \to a$

(ii) $S \to bS$ → $S \to bS$, $S \to b$

Step 3:-

Remove ϵ prod.

$S \rightarrow aS \mid a$

$S \rightarrow bS \mid b$

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

Removal of unit prod.

A prod. of form non-terminal → one non-terminal is called as unit prod.

e.g. $A \rightarrow B$

Algorithm to remove unit prod.

while (there exist a unit prod. $A \rightarrow B$)

{

select a unit production $A \rightarrow B$ ∃ a prod. $B \rightarrow X$, where X is terminal.

For (every non-unit prod, $B \rightarrow X$)

Add production, $A \rightarrow X$ to grammar

Eliminate $A \rightarrow B$ from grammar.

}

e.g. Remove unit prod.

$S \rightarrow AB$

$A \rightarrow a$

$B \rightarrow \epsilon \mid b$

$C \rightarrow D$

$D \rightarrow E$

$E \rightarrow a$

Page No.
Date

Sol. $D \to E$
$E \to a$.

We can write it,
$D \to a$.
and remove $D \to E$ from grammar
and add $D \to a$

Now, grammar is
$S \to AB$
$A \to a$
$B \to b$
$B \to C$
$C \to D$ ⎫ ⟹ $C \to a$.
$D \to a$ ⎭
$E \to a$

Now, Grammar is
$S \to AB$
$A \to a$
$B \to b$
$B \to C$ ⎫ → $B \to a$
$C \to a$ ⎭
$D \to a$
$E \to a$

Now, Grammar is
$S \to AB$
$A \to a$
$B \to b$
$B \to a$
$C \to a$
$D \to a$
$E \to a$
Useless

Remove Useless — C, D, E from G.

we get,

$S \rightarrow AB$

$A \rightarrow a$

$B \rightarrow b \mid a$.

Ques⇒ Begin with the Grammar:-

$S \rightarrow ABC \mid BaB$

$A \rightarrow aA \mid BaC \mid aaa$

$B \rightarrow bBb \mid a \mid D$

$C \rightarrow CA \mid AC$

$D \rightarrow \epsilon$

(i) Eliminate ϵ productions.

(ii) Eliminate any unit productions in the resulting grammar.

(iii) Eliminate any useless symbols in the resulting grammar.

Sol⇒ (i) Elimination of ϵ productions:-

Step 1 :- find all nullable variable.

i.e. D

Step 2 :- Creating two versions

$S \rightarrow ABC$ ~~$S \rightarrow ABC$~~

$S \rightarrow AC$

$S \rightarrow BaB$ $S \rightarrow a \mid aB \mid Ba$

$S \rightarrow BaB$

Page No.
Date

$A \to BaC$ ⟨ $A \to aC$, $A \to BaC$

$B \to bBb$ ⟨ $B \to bb$, $B \to bBb$

Remove G,

$S \to ABC \mid AC \mid a \mid aB \mid Ba \mid BaB$

$A \to aC \mid BaC \mid aA \mid aaa$

$B \to bb \mid bBb \mid a$

$C \to AC \mid CA$

(ii) There is not unit production

(iii) C & A are useless.

$S \to ABC \mid AC \mid a \mid aB \mid Ba \mid BaB$

$B \to bb \mid bBb \mid a$

Page No.
Date

Pumping Theorem (Ch-16)

* Pumping Theorem for CFL :-

Pumping lemma (for CFL) is used to prove that a language is not a context free.

Assume L is a context free lang., there is a pumping length n such that any string $w \in L$ of length $\geq n$ can be written as

$$|w| \geq n$$

we can break w into 5 strings,

$$w = uvxyz,$$

such that

Pumping condition —

- $\Rightarrow |vxy| \leq n$
- $\Rightarrow |vy| \geq 1 \neq \epsilon$
- $\Rightarrow$ for all $k \geq 0$, the string $uv^kxy^kz \in L$

select k such that the resulting string is not in L.

To prove that a language is not CF using pumping lemma (CFL) follow the steps:- (We prove using contradiction)

1. Assume that L is context-free.
2. The pumping length say 'n'.
3. All strings longer than n can be pumped $|w| \geq n$
4. Now find a string 'w' in L such that $|w| \geq n$
5. Divide w into uvxyz
6. show that $uv^kxy^kz \notin L$ for some k.
7. The consider the ways that w can be

divided into uvxyz

(8) Show that none of these can satisfy all the 3 pumping condition at same time.

(9) w cannot be pumped (contradiction).

Q:⇒ Find out whether $L = \{x^n y^n z^n \mid n \geq 1\}$ is context-free or not.

Sol⇒

Let L is context free.

⇒ now we can take a string 'w' such that

&

$$w = x^n y^n z^n$$

⇒ we divide w in 5 parts uvxyz.

Case (i) :- Let $n = 4$.

then, $w = x^4 y^4 z^4$

& v & y. Each contain only one type of symbol

$$w = xxxx\,yyyy\,zzzz$$

Divide it into 5 parts (uvxyz)

x xxx yyyy z z z z

u v x y z

Acc. to pumping lemma :-

⇒ $uv^k xy^k z$ (let take $k = 2$)

⇒ $uv^2 xy^2 z$

⇒ $x(xx)^2 xyyyyz(z)^2 zz$

⇒ xxxxx x yyyy z zzzz

$\Rightarrow x^6 y^4 z^5 \notin L$

Now, in this x, y & z are not equal that means $x^6 y^4 z^5 \notin L$.

The resultant string does not satisfy the condition, the no of x, the no of y and no of z should be equal. But here is not.

That's why our assumption is wrong. Therefore, given language L is not a context - free.

Case (ii):- Either V or Y has more than one kind of symbols.

$W = x^n y^n z^n$ (Divide it into 5 parts — uvxyz)

Let $n = 4$,

then $w = xxxx\ yyyy\ zzzz$.

Now, divide this string into uvxyz.

$\underbrace{xx}_{u}\ \underbrace{xxyy}_{v}\ \underbrace{y}_{x}\ \underbrace{y}_{y}\ \underbrace{zzzz}_{z}$

$u = xx$

$v = xxyy$

$x = y$

$y = y$

$z = zzzz$

let assume $K = 2$,

Page No.
Date

Now, acc. to pumping lemma condition,

$\Rightarrow uv^k xy^k z$

$\Rightarrow uv^2 xy^2 z$

$\Rightarrow (xx)^1 (xxyy)^2 (yy)^1 (yy)^2 (zzzz)$

$\Rightarrow xx\,xxyy\,xxyy\,yyyy\,zzzz$

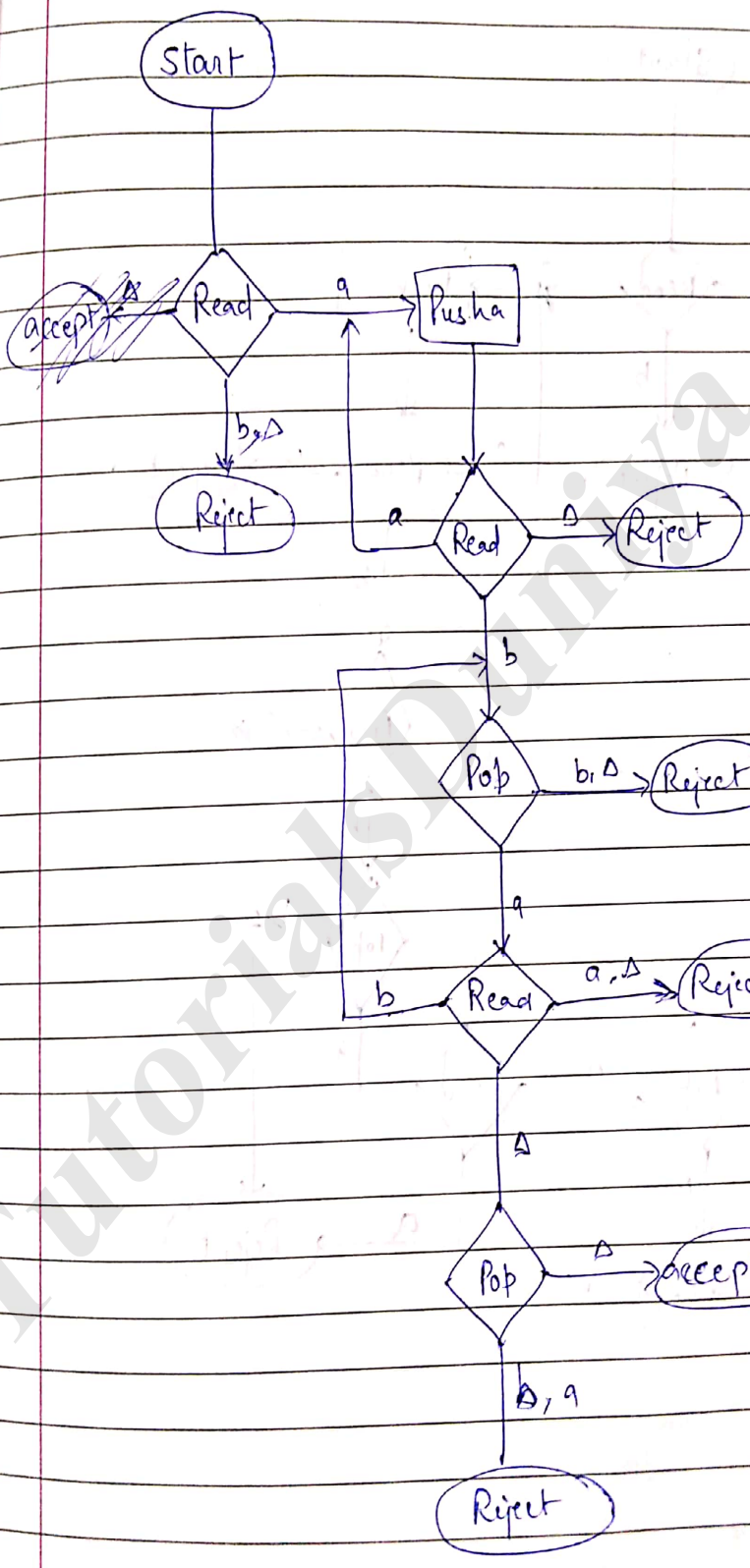
$\Rightarrow x^4 y^2 x^2 y^5 z^4$ ∉ L.

This string is not following the pattern of L.
x can't be come after y.
Therefore, this is not a context free.

PDA

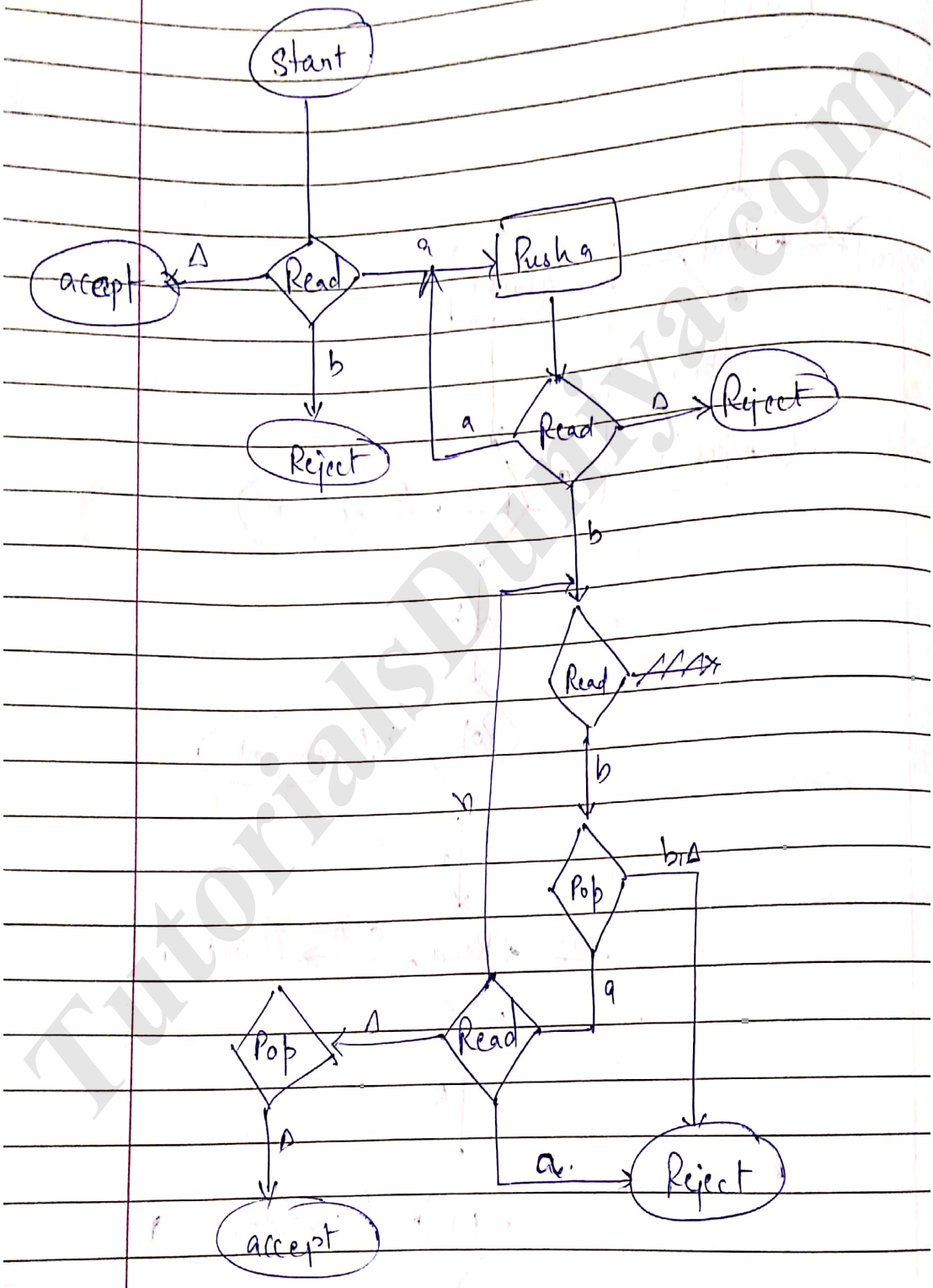
Page No.
Date

Q=> Design PDA for $a^n b^n$ $(n \geq 1)$



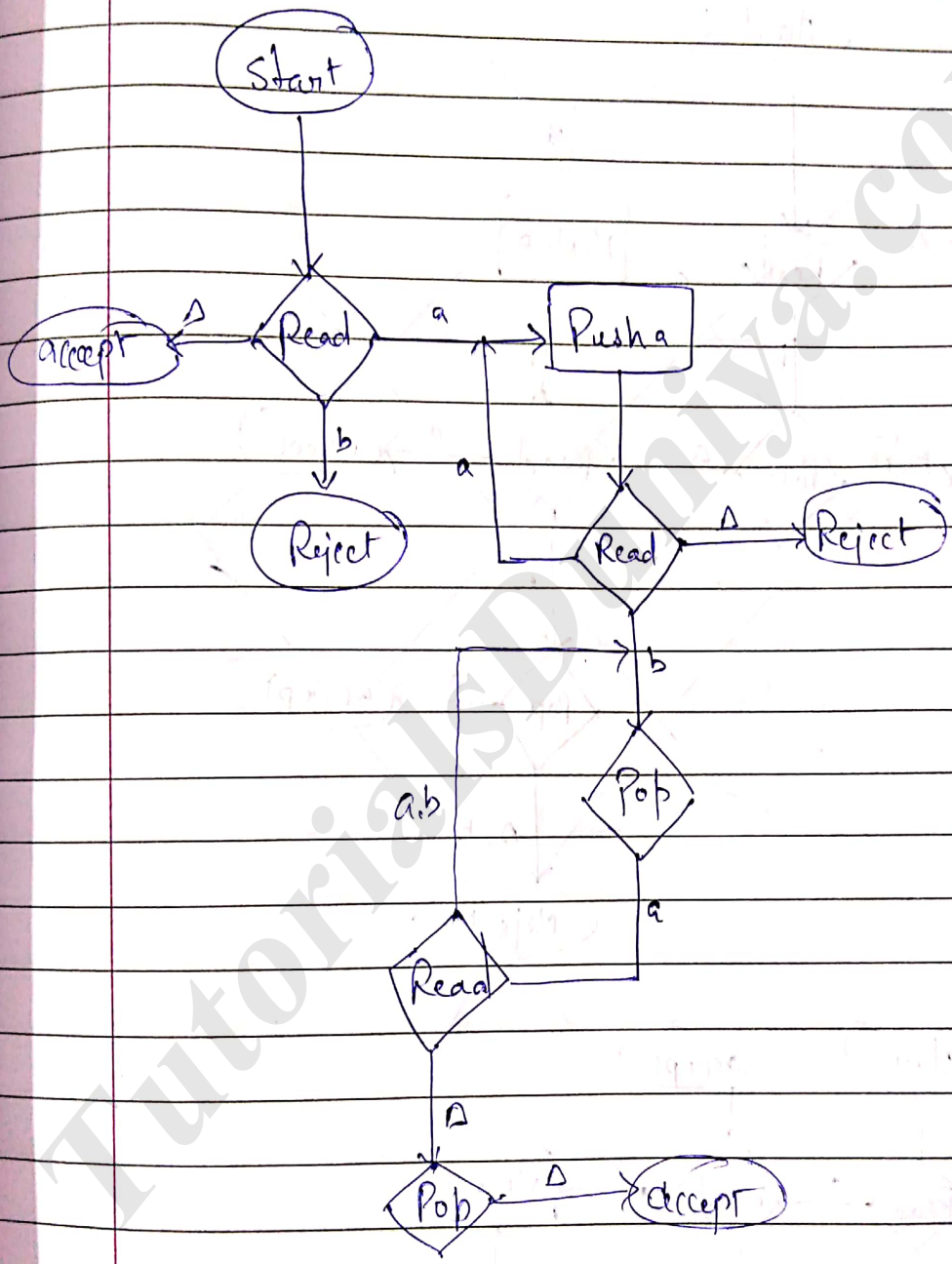
Page No.
Date

Q=> Design PDA for $a^n b^{2n}$



Page No.
Date

Q: Design PDA for $a^n S$ where S starts with length(S) = n.



TutorialsDuniya.com

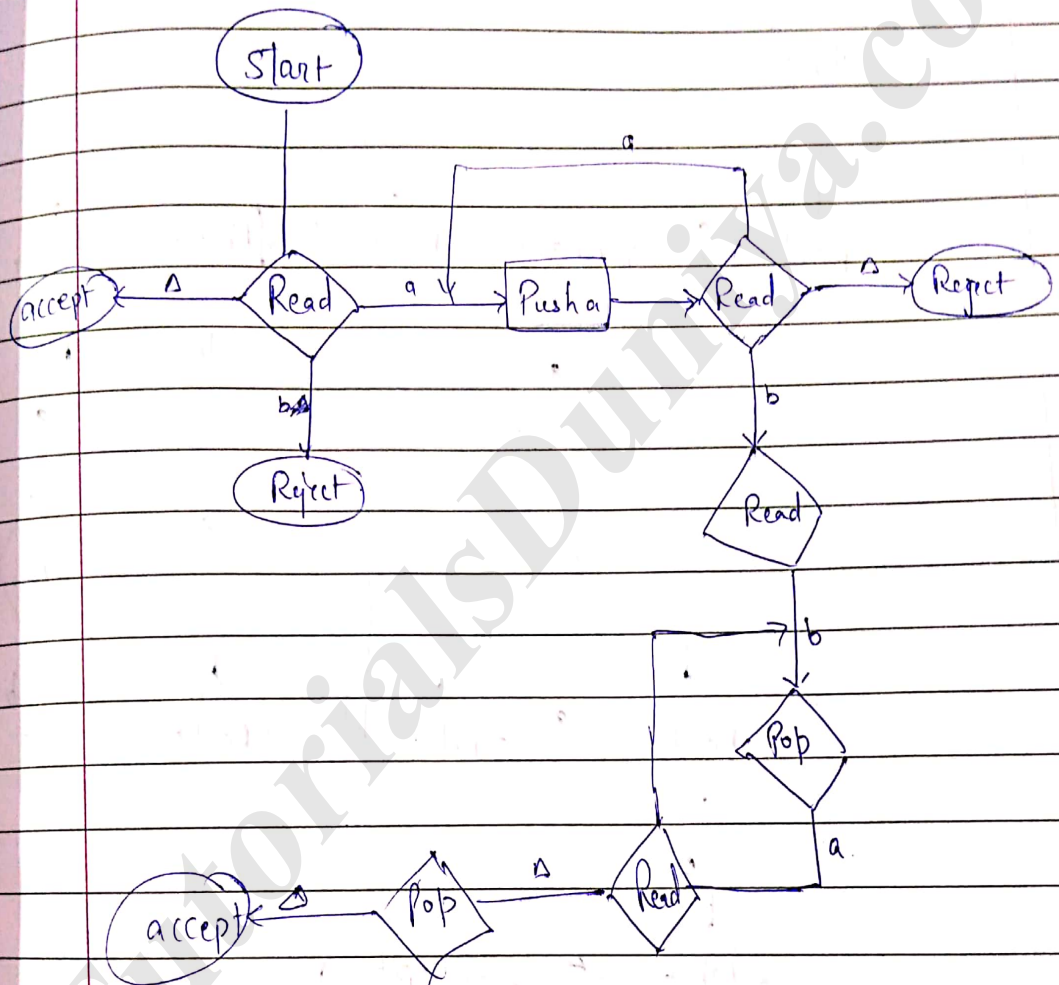
Download FREE Computer Science Notes, Programs, Projects, Books PDF for any university student of BCA, MCA, B.Sc, B.Tech CSE, M.Sc, M.Tech at https://www.tutorialsduniya.com

- Algorithms Notes
- Artificial Intelligence
- Android Programming
- C & C++ Programming
- Combinatorial Optimization
- Computer Graphics
- Computer Networks
- Computer System Architecture
- DBMS & SQL Notes
- Data Analysis & Visualization
- Data Mining
- Data Science
- Data Structures
- Deep Learning
- Digital Image Processing
- Discrete Mathematics
- Information Security
- Internet Technologies
- Java Programming
- JavaScript & jQuery
- Machine Learning
- Microprocessor
- Operating System
- Operational Research
- PHP Notes
- Python Programming
- R Programming
- Software Engineering
- System Programming
- Theory of Computation
- Unix Network Programming
- Web Design & Development

Please Share these Notes with your Friends as well

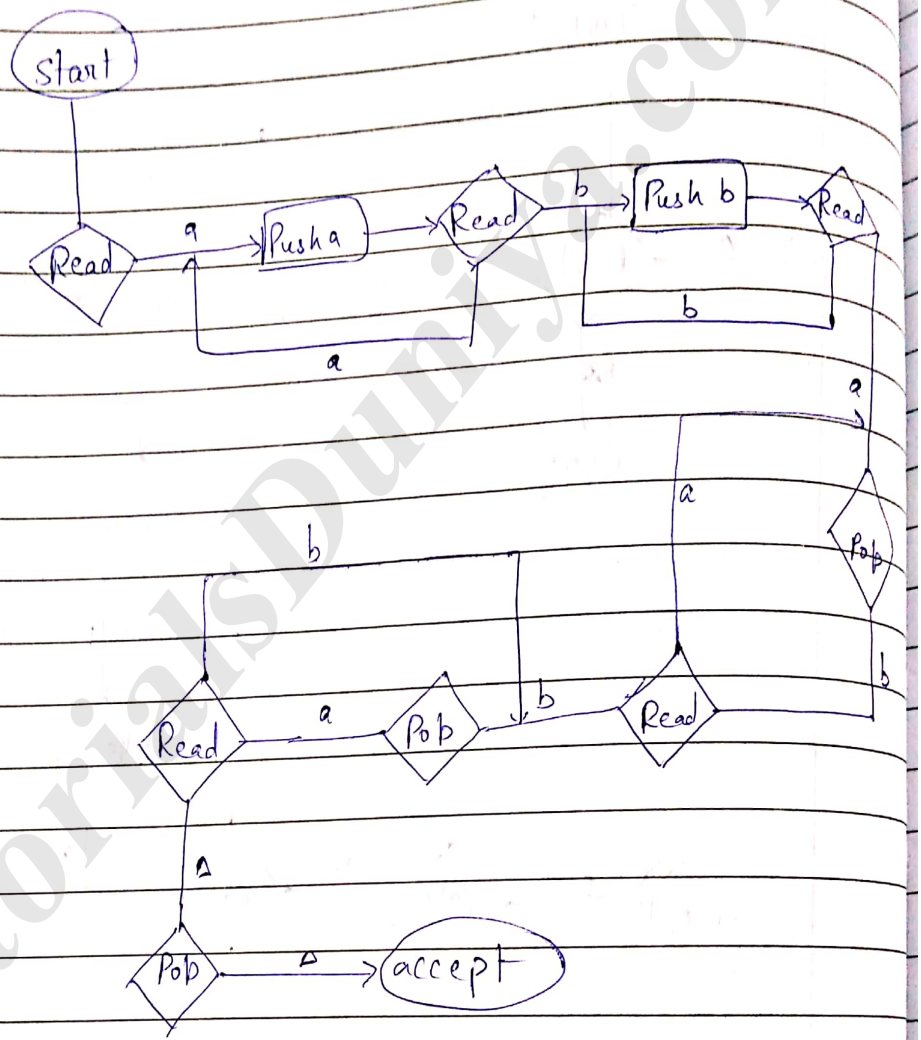
Page No.
Date

Q:- Design PDA for $a^n b^{n+1}$ $(n \geq 1)$



Date

Q=> Design PDA for $a^n b^m a^m b^n$ $(n, m \geq 1)$



Date

Turing Machine

Turing Machine

A turing machine is a quintuple $(K, \Sigma, \delta, s, H)$, where

K – finite set of states

Σ – alphabet containing blank symbol $\sqcup$ and left end symbol $\triangleright$, but not $\rightarrow, \leftarrow$

s – initial state

H – Halting state

δ – transition function.

$$\delta(q, \Sigma) \rightarrow (q, \Sigma, L/R)$$

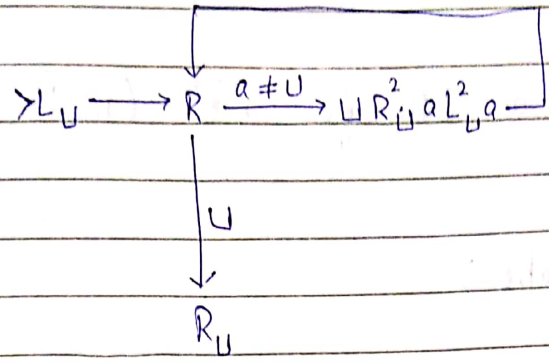
A turing machine consists of a finite control, a tape and a head that can be used for reading or writing on that tape.

Copying Machine

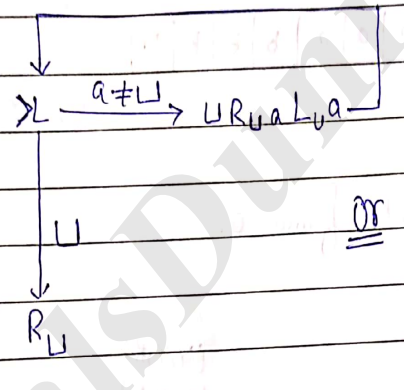
The copying machine C copies the data. If C starts with input w, that is, if string w, containing only nonblank symbols but possibly empty, is put on an otherwise blank tape with one blank square to its left, and the head is put on the blank square to the left of w, then the machine eventually stops with w⊔w on an otherwise blank tape. We say that C transforms

Page No.
Date

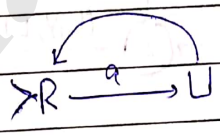
$\sqcup w \sqcup$ into $\sqcup w \sqcup w \sqcup$.



Right-shifting Machine $S_\rightarrow$ transforms $\sqcup w \sqcup$, where w contains no blank, into $\sqcup \sqcup w \sqcup$



Erase a.



Continue Right-Shifting Machine, the rightward analog of machine. This machine finds the right end of input and then goes to the left as long as it sees 1's, changing all of them to 0's. When it sees a 0, it changes it into 1 and halt. If it sees a $\sqcup$ while looking for 0, this means that the input no has a binary representation that is all 1's. & so machine again write 1 in place of $\sqcup$ and halts., after shifting the whole string one position to the right.

Page No.
Date

Designation of Turing Machine.

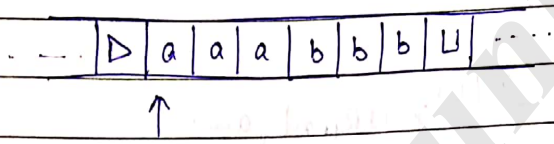
not necessary to complete T.M.

Q⇒ Design a T.M for $a^n b^n$ $(n \geq 1)$

Sol⇒

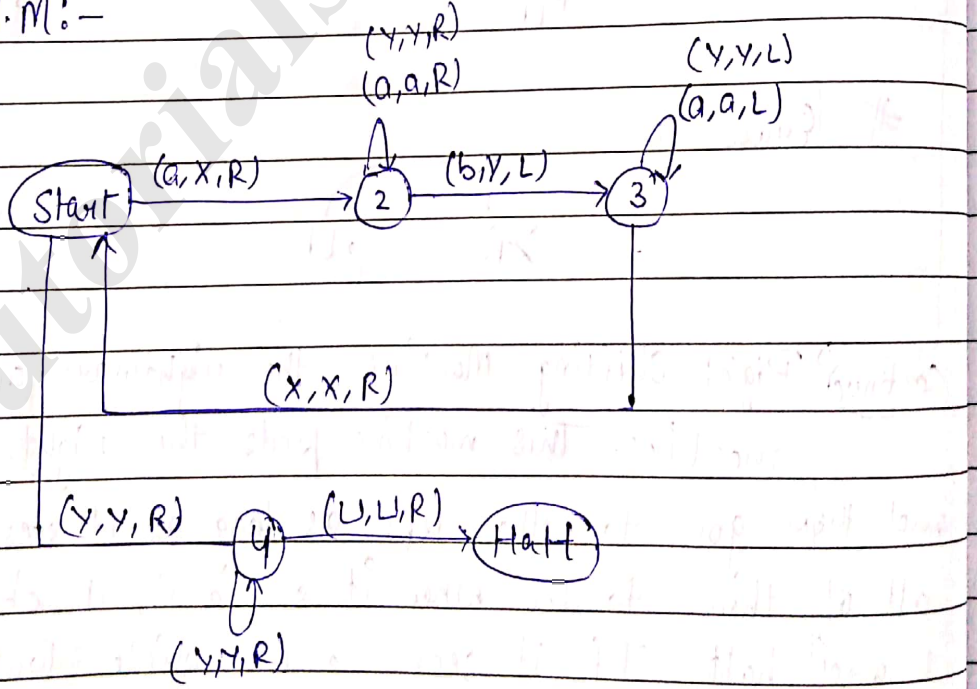
Input tape ⇒

Let's take an example of string which is accepted by given language. i.e., aaabbb.



Head starts from a.

T.M:-



Page No.
Date

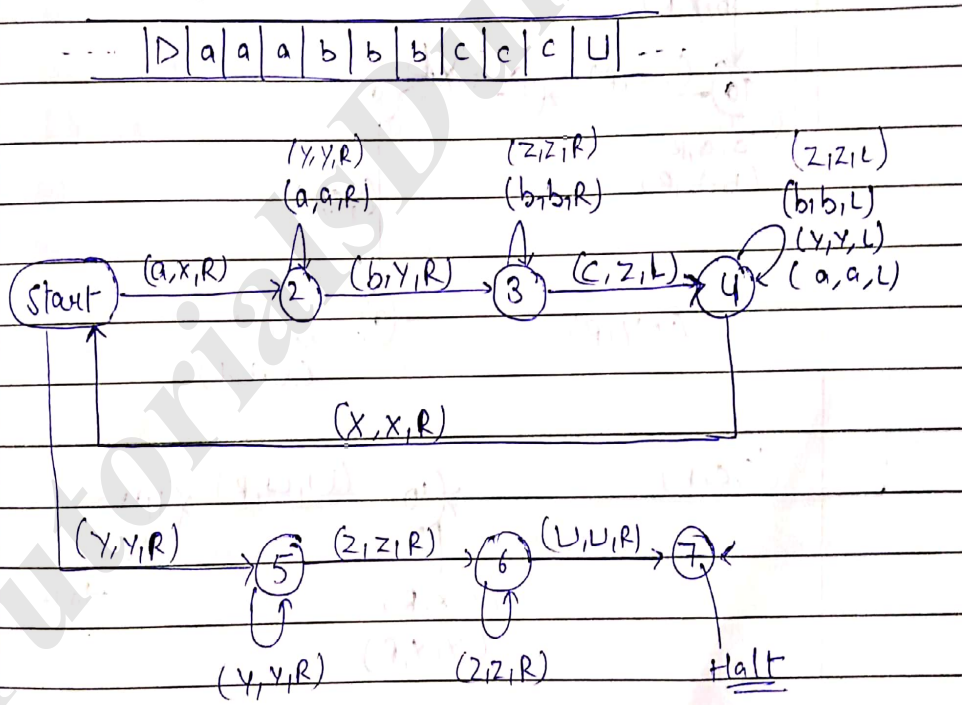
Q⇒ Draw Transition Table for $a^n b^n$ $(n \geq 1)$.

Sol⇒

	State	a	b	X	Y	⊔
(→)	1	(2, X, R)	H	H	(4, Y, R)	H
	2	(2, a, R)	(3, Y, L)	H	(2, Y, R)	H
	3	(3, a, L)	H	(1, X, R)	(3, Y, L)	H
	4	H	H	H	(4, Y, R)	(5, ⊔, R)
(Halt) (*)	5	H	H	H	H	H

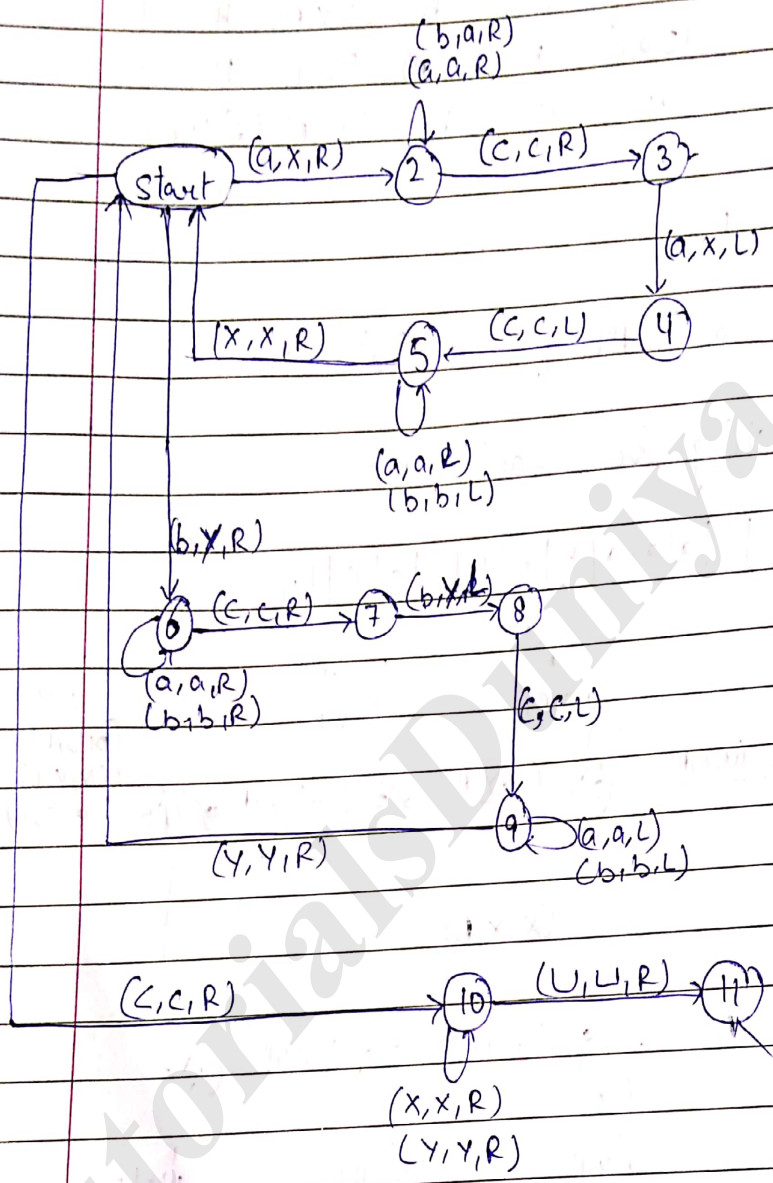
Q⇒ Design a T.M for $a^n b^n c^n$ $(n \geq 1)$

Sol⇒ Let string be aaabbbccc.



Q⇒ Design a T.M for WCW | W ∈ (a,b).

Sol ⇒ | ▷ | a | b | a | a | c | a | b | ␣ | a | ␣ | ...



Recursive Language:-

A language L is recursive if L = L(M) for some TM 'M' such that:

(1) if w is in L, then M accepts & then halts.

(2) if w is not in L, then M halts (without accepting it)

Date

Recursively Enumerable Languages

A language L is recursively enumerable iff there is a turing machine M that semidecides L.

Theorem:-

If language L is recursive, then L is recursively enumerable also.

Proof:- let $M = (K, \Sigma, \delta, s, H)$ be a T.M.
let Σ be an alphabet which also consists $\cup, \triangle$.
let $L \subset \Sigma^*$ be a language.
We say that M semidecides L if for every string $w \in \Sigma^*$, the following is true:
$w \in L$ iff M halts on input 'w'.
A language is recursively enumerable iff there is a T.M 'M' that semidecides L.

Theorem:-

The complement of a recursive language is also recursive.

Proof:- If L is decided by T.M. $M = \{K, \Sigma, \delta, s, \{y, n\}\}$ then L^* is decided by T.M $M' = \{K, \Sigma, \delta', s, \{y, n\}\}$ which is identical to M except that it reverses the roles of the two special halting states y and n. That is, δ' is defined as follows:

$$\delta'(q,a) = \begin{cases} n & \text{if } \delta(q,a) = y, \\ y & \text{if } \delta(q,a) = n, \\ \delta(q,a) & \text{otherwise.} \end{cases}$$

It is clear that $M'(w) = y$ iff $M(w) = n$
Therefore M' decides $\bar{L}$.

Random Access Turing Machine

A random access turing machine is a pair $M = (K, \Pi)$, where $K > 0$ is the no of registers, and $\Pi = (\pi_1, \pi_2, \ldots, \pi_p)$ the program, is a finite sequence of instructions, where each instruction π_i is of one of the type like read, write, etc. We assume that a ~~that~~ last instruction, π_p, is always a halt instruction.

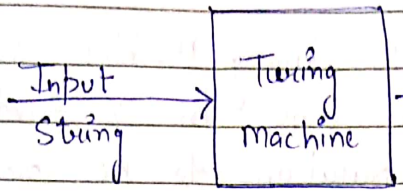
Diagonalization Language

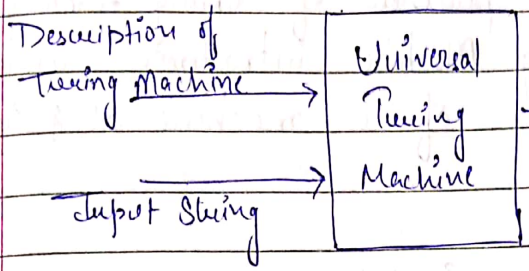
The language (L_d), ~~is~~ the diagonalization language, is the set of strings w_i such that $w_i \notin L(M_i)$.

L_d is not recursively enumerable language. That is, there is no TM that accepts L_d.

Page No.
Date

Universal Turing Machine





Turing Machine	Universal Turing Machine
1.) Hardwired Machine.	1.) Programmable Turing Machine.
2.) It does one task (execute one program).	2.) It can do different types of task.

- A Universal Turing Machine is a specified Turing machine that can simulate the behaviour of any Turing Machine.
- To design a Universal Turing Machine, we have to increase the number of read/write

heads, dimensions of input tape and adding a special purpose memory in basic model of Turing Machine.

- The problem with Turing Machine is that a different machine must be constructed for every new computation to be performed, that is for every input output relation.

- To solve this problem of Turing machine, Universal Turing Machine introduced, which takes description of a machine M & input on the tape.

- The UTM then simulate M on the contents of input tape.
Therefore, UTM can simulate any other turing machine.

- So for any problem that can be solved by TM, we can either use a TM that directly solves the problem or we could use a UTM & give it the description of a TM that directly solves it.

As a TM can solve any computing problem, a UTM can be programmed to do the same.

UTM is a general purpose machine that can be used to compute any computable problem.

Page No.
Date

Halting Problem

Halting Problem is undecidability. It is not a problem, it just asks a question :-

"Is it possible to tell whether a given machine will halt for some given input ?"

E.g. Input → A T.M & i/p string w.

Problem → Does the TM finish computing of the string w in a finite no' of steps ?

The answer must be Yes or No.

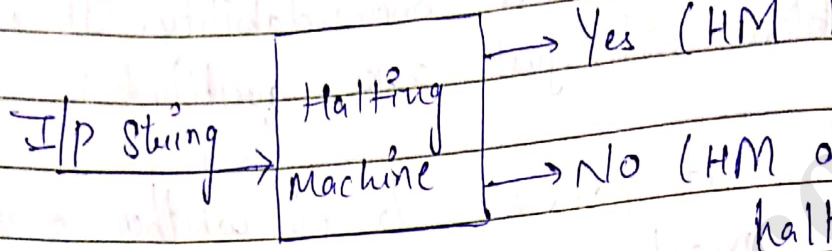
Proof → Assume a T.M exists to solve this problem and then we will show it is contradicting itself.

We will call this T.M as a halting machine that produce a 'Yes' or 'No' in a finite amount of time.

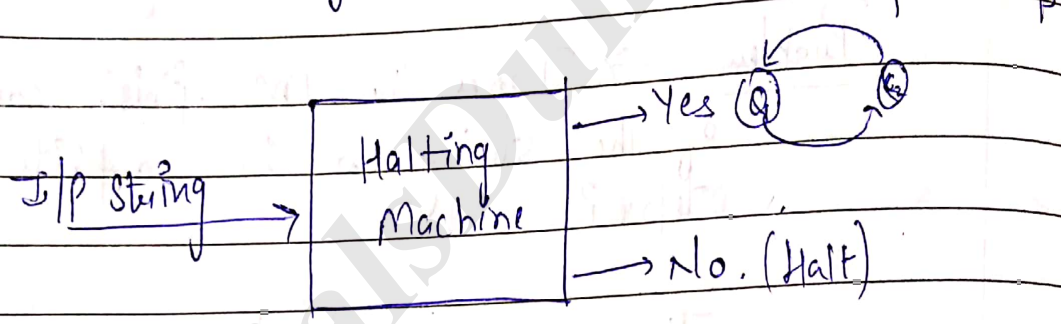
If halting machine finishes in a finite amount of time, output comes as 'Yes' otherwise as 'No'.

Date

The block diagram of Halting machine :-



Inverted halting Machine :-



~~If H.M~~

This is the design of inverted halting machine as :-

- if it returns Yes, then loop forever
- if it returns No, then Halt.

TutorialsDuniya.com

Download FREE Computer Science Notes, Programs, Projects, Books PDF for any university student of BCA, MCA, B.Sc, B.Tech CSE, M.Sc, M.Tech at https://www.tutorialsduniya.com

- Algorithms Notes
- Artificial Intelligence
- Android Programming
- C & C++ Programming
- Combinatorial Optimization
- Computer Graphics
- Computer Networks
- Computer System Architecture
- DBMS & SQL Notes
- Data Analysis & Visualization
- Data Mining
- Data Science
- Data Structures
- Deep Learning
- Digital Image Processing
- Discrete Mathematics
- Information Security
- Internet Technologies
- Java Programming
- JavaScript & jQuery
- Machine Learning
- Microprocessor
- Operating System
- Operational Research
- PHP Notes
- Python Programming
- R Programming
- Software Engineering
- System Programming
- Theory of Computation
- Unix Network Programming
- Web Design & Development

Please Share these Notes with your Friends as well