APJ ABDUL KALAM TECHNOLOGICAL UNIVERSITY

FIFH SEMESTER B.TECH DEGREE MODEL EXAMINATION, NOVEMBER 2017

Department**: Computer Science and Engineering**

Subject: - CS301: Theory of Computation

Time: 3 hours Max. Marks: 100

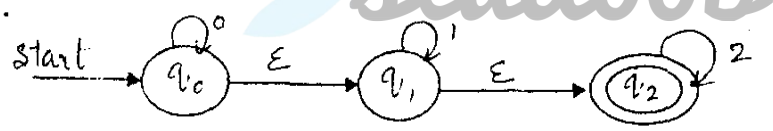
PART A Answer *all* questions

1. Explain a) Language of DFA b) Extended transition function. (3)
2. Design a DFA, which accepts the strings with even number of 0's and even number of 1's over {0,1} (3)
3. Construct the finite automaton equivalent to the regular expression i) R.S ii) R* (3)
4. Design a Moore machine that takes set of all strings over {a,b} as input and prints 1 as output for every occurrence of **baa** as a substring. (3)

Total: (12)

PART B
Answer any *two* full questions

5. Convert the given NFA with ε-moves into NFA without ε-moves (9)



6. State and prove the equivalence of DFA and NFA. (9)
7. Reduce the following DFA where q_0 is the initial state and q_2 is the final state. (9)

State	0	1
q_0	q_1	q_5
q_1	q_6	q_2
q_2	q_0	q_2
q_3	q_2	q_6
q_4	q_7	q_5
q_5	q_2	q_6
q_6	q_6	q_4
q_7	q_6	q_2

Total: (18)

PART C
Answer *all* questions

8. Show that the language L={$0^i 1^i$ | i is an integer and i>=1} is not regular using Pumping Lemma. (3)
9. Define CFG. Give CFG generating the set of palindromes over alphabet {a,b}. (3)
10. Explain Closure properties of CFL. (3)
11. Define Push Down Automata. (3)

Total: (12)

PART D
Answer any *two* full questions

12. a. Show that the grammar is ambiguous S -> aS | aSbS | ε (3)
    b. Explain i) Useless symbols ii)Unit Production iii)Null Production (6)
13. Find a Griebach Normal form grammar equivalent to the following CFG (9)
    S -> AA | 0
    A -> SS | 1
14. Design a Push Down Automata that accepts the language L={wcw^r | w in $(a,b)^+$}: (9)

Total: (18)

PART E
Answer any four full questions

15. a. Explain the basic Turing machine model with a neat diagram. (5)
    b. Define Linear Bounded Automata (5)
16. Design a Turing machine that accepts the language over {a,b} L={ $0^n 1^n$ | n >=1} (10)
17. a. List the basic closure properties of recursive and recursively enumerable sets. (5)
    b. Explain Chomsky hierarchy. (5)
18. Explain the halting problem of Turing machine. (10)

Total: (40)

APJ ABDUL KALAM TECHNOLOGICAL UNIVERSITY

FIFH SEMESTER B.TECH DEGREE EXAMINATION, DECEMBER 2017

Department**: Computer Science and Engineering**

Subject: - CS301: Theory of Computation

Time: 3 hours Max. Marks: 100

PART A

Answer *all* questions

1. a. **Language of DFA**:The language of a DFA A = (Q, Σ, δ, q 0, F), denoted L(A) is defined by L(A) = { w | δˆ (q 0, w) is in F } .The language of A is the set of strings w that take the start state q 0 to one of the accepting states. If L is a L(A) from some DFA, then L is a regular language

(1.5)

b. **Extended transition function**:The transition function can be extended to operate on words

$\delta\hat{} \in (Q \times \Sigma^* \rightarrow Q)$

$\delta\hat{}\ (q, \varepsilon) = q$ (1)

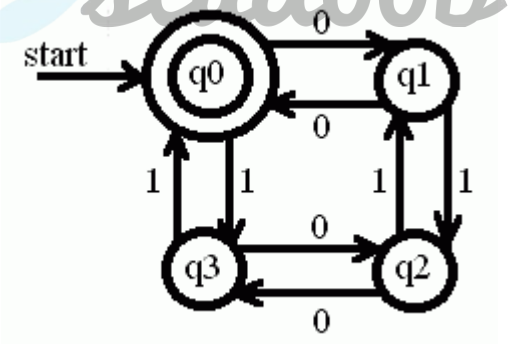
$\delta\hat{}\ (q, xw) = \delta\hat{}\ (\delta\ (q, x), w)$ (2)

where $q \in Q\ x \in \Sigma\ w \in \Sigma^*$

if we start in state q, read an input word w, and end up in state q ' , then $\delta\hat{}\ (q,w) = q$

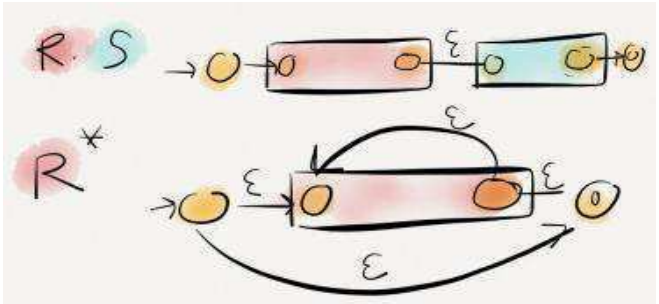
(1.5)

2.



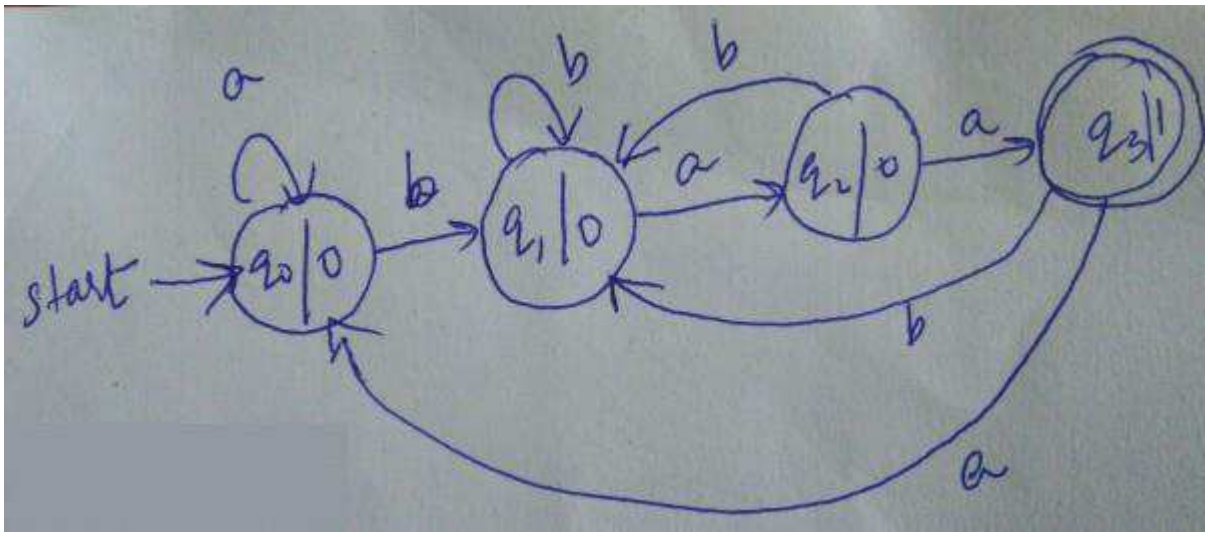
(3)

3.



(3)

4.



(3)

PART B

5.

	State	0	1	2
→	*{ q_0, q_1, q_2 }	{ q_0, q_1, q_2 }	{ q_1, q_2 }	{ q_2 }
	*{ q_1, q_2 }	Φ	{ q_1, q_2 }	{ q_2 }
	*{ q_2 }	Φ	Φ	{ q_2 }

→ **Initial State**

* **Final state**

(9)

6. For every NDFA, there exists a DFA which simulates the behaviour of NDFA.
Alternatively, if *L* is the set accepted by NDFA, then thereexists a DFA which also accepts *L*.

Proof Let M = (Q, L, 8, qQ, F) be an NDFA accepting L. We construct a DFA M' as:
M' = (Q', L, 8, (/o, F')
where
(i) Q' =2^Q
(ii) (ii) q'0 = [qo]; and
(iii) r is the set of all subsets of Q containing an element of F.

Now we can define δ':

(iv) $\delta'([q_1, q_2, \ldots, q_i], a) = \delta(q_1, a) \cup \delta(q_2, a) \cup \ldots \cup \delta(q_i, a)$.

Equivalently,

$$\delta'([q_1, q_2, \ldots, q_i], a) = [p_1, \ldots, p_j]$$

if and only if

$$\delta(\{q_1, \ldots, q_i\}, a) = \{p_1, p_2, \ldots, p_j\}.$$

Before proving $L = T(M')$, we prove an auxiliary result

$$\delta'(q_0', x) = [q_1, \ldots, q_i], \tag{3.4}$$

if and only if $\delta(q_0, x) = \{q_1, \ldots, q_i\}$ for all x in Σ^*.

We prove by induction on $|x|$, the 'if' part, i.e.

$$\delta'(q_0', x) = [q_1, q_2, \ldots, q_i] \tag{3.5}$$

if $\delta(q_0, x) = \{q_1, \ldots, q_i\}$.

When $|x| = 0$, $\delta(q_0, \Lambda) = \{q_0\}$, and by definition of δ', $\delta'(q_0', \Lambda) = q_0' = [q_0]$. So. (3.5) is true for x with $|x| = 0$. Thus there is basis for induction.

Assume that (3.5) is true for all strings y with $|y| \leq m$. Let x be a string of length $m + 1$. We can write x as ya, where $|y| = m$ and $a \in \Sigma$. Let $\delta(q_0, y) = \{p_1, \ldots, p_j\}$ and $\delta(q_0, ya) = \{r_1, r_2, \ldots, r_k\}$. As $|y| \leq m$, by induction hypothesis we have

$$\delta'(q_0', y) = [p_1, \ldots, p_j] \tag{3.6}$$

Also,

$$\{r_1, r_2, \ldots, r_k\} = \delta(q_0, ya) = \delta(\delta(q_0, y), a) = \delta(\{p_1, \ldots, p_j\}, a)$$

By definition of δ',

$$\delta'([p_1, \ldots, p_j], a) = [r_1, \ldots, r_k] \tag{3.7}$$

Hence,

$$\delta'(q'_0, ya) = \delta'(\delta'(q'_0, y), a) = \delta'([p_1, \ldots, p_j], a) \quad \text{by (3.6)}$$
$$= [r_1, \ldots, r_k] \quad \text{by (3.7)}$$

Thus we have proved (3.5) for $x = ya$.

By induction, (3.5) is true for all strings x. The other part (i.e. the 'only if' part), can be proved similarly, and so (3.4) is established.

Now, $x \in T(M)$ if and only if $\delta(q, x)$ contains a state of F. By (3.4), $\delta(q_0, x)$ contains a state of F if and only if $\delta'(q'_0, x)$ is in F'. Hence, $x \in T(M)$ if and only if $x \in T(M')$. This proves that DFA M' accepts L. ∎

(9)

7.

$$Q_1^0 = F = \{q_2\}, \qquad Q_2^0 = Q - Q_1^0$$

$$\pi_0 = \{\{q_2\}, \{q_0, q_1, q_3, q_4, q_5, q_6, q_7\}\}$$

$$\pi_1 = \{\{q_2\}, \{q_0, q_4, q_6\}, \{q_1, q_7\}, \{q_3, q_5\}\}$$

$$\pi_2 := \{\{q_2\}, \{q_0, q_4\}, \{q_6\}, \{q_1, q_7\}, \{q_3, q_5\}\}$$

$$\pi_2 = \pi_3.$$

$$M' = (Q', \{0, 1\}, \delta', q'_0, F')$$

where

$$Q' = \{[q_2], [q_0, q_4], [q_6], [q_1, q_7], [q_3, q_5]\}$$

$$q'_0 = [q_0, q_4], \qquad F' = [q_2]$$

and δ' is defined by

Transition Table of Minimum State Automaton

State/Σ	0	1
$[q_0, q_4]$	$[q_1, q_7]$	$[q_3, q_5]$
$[q_1, q_7]$	$[q_6]$	$[q_2]$
$[q_2]$	$[q_0, q_4]$	$[q_2]$
$[q_3, q_5]$	$[q_2]$	$[q_6]$
$[q_6]$	$[q_6]$	$[q_0, q_4]$

(9)

8.

Solution

Step 1 Suppose L is regular. Let n be the number of states in the finite automaton accepting L.

Step 2 Let $w = 0^n1^n$. Then $|w| = 2n > n$. By pumping lemma, we write $w = xyz$ with $|xy| \leq n$ and $|y| \neq 0$.

Step 3 We want to find i so that $xy^iz \notin L$ for getting a contradiction. The string y can be in any of the following forms:

Case 1 y has 0's, i.e. $y = 0^k$ for some $k \geq 1$.

Case 2 y has only 1's, i.e. $y = 1^l$ for some $l \geq 1$.

Case 3 y has both 0's and 1's, i.e. $y = 0^k1^j$ for some $k, j \geq 1$.

In Case 1, we can take $i = 0$. As $xyz = 0^n1^n$, $xz = 0^{n-k}1^n$. As $k \geq 1$, $n - k \neq n$. So, $xz \notin L$.

In Case 2, take $i = 0$. As before, xz is 0^n1^{n-l} and $n \neq n - l$. So, $xz \notin L$.

In Case 3, take $i = 2$. As $xyz = 0^{n-k}0^k1^j1^{n-j}$, $xy^2z = 0^{n-k}\,0^k1^j0^k1^j1^{n-j}$. As xy^2z is not of the form 0^i1^i, $xy^2z \notin L$.

Thus in all the cases we get a contradiction. Therefore, L is not regular.

(3)

9. A CFG consists of the following components:

- a set of *terminal symbols*, which are the characters of the alphabet
- a set of *nonterminal symbols*, which are placeholders for patterns of terminal symbols that can be generated by the nonterminal symbols.
- a set of *productions*, which are rules for replacing (or rewriting) nonterminal symbols (on the left side of the production) in a string with other nonterminal or terminal symbols (on the right side of the production).
- a *start symbol*, which is a special nonterminal symbol that appears in the initial string generated by the grammar.

CFG generating the set of palindromes over alphabet {a,b}

S –> aSa | bSb | a | b| ε

(3)

10. Context-free languages are **closed** under –

- Union
- Concatenation

- Kleene Star operation

Let L_1 and L_2 be two context free languages. Then $L_1 \cup L_2$ is also context free.
Let L_1 and L_2 be two context free languages. Then $L_1 . L_2$ is also context free.
If L is a context free language, then L* is also context free.

(3)

11. Pushdown Automata is a finite automata with extra memory called stack which helps to recognize Context Free Languages.

A Pushdown Automata (PDA) can be defined as :

- Q is the set of states
- ∑ is the set of input symbols
- Γ is the set of pushdown symbols (which can be pushed and popped from stack)
- q0 is the initial state
- Z is the initial pushdown symbol (which is initially present in stack)
- F is the set of final states
- δ is a transition function which maps Q x { ∑ ∪ ε } x Γ into Q x Γ *. In a given state, PDA will read input symbol and stack symbol (top of the stack) and move to a new state and change the symbol of stack.

(3)

PART D

Answer any *two* full questions

12. a

Wecan derive the string aab as follows

```
S  -> aSbS -> aaSbS -> aabS -> aab
S  -> aS   -> aaSbS -> aabS -> aab
```

Two parse trees exist
So grammar is ambiguous

(3)

b. A symbol which is non generating or not reachable from start symbol is called **Useless Symbol**
S -> AB
A -> a HERE B is **Useless Symbol**
Any production rule in the form A → B where A, B ∈ Non-terminal is called **Unit production**
Any production rule in the form **A → epsilon** where A ∈ Non-terminal is called **Null production**

(3*2=6)

13. S -> AA | 0
A -> SS | 1

first rename the variables:
A1 -> A2 A2 | 0
A2 -> A1 A1 | 1

three of the four productions are just fine,
only A2 -> A1 A1 is a problem.

Apply algorithm 1:
A2 -> A2 A2 A1 | 0 A1
So, now we have still have one problem production:
A2 -> A2 A2 A1

Apply algorithm 2:
A2 -> 1 | 0 A1 | 1 B | 0 A1 B
B -> A2 A1 | A2 A1 B
So now our grammar looks like:

A1 -> A2 A2 | 0
A2 -> 1 | 0 A1 | 1 B | 0 A1 B
B -> A2 A1 | A2 A1 B

Now we must fix A1, so that is only starts with terminals:
A1 -> 1 A2 | 0 A1 A2 | 1 B A2 | 0 A1 B A2 | 0

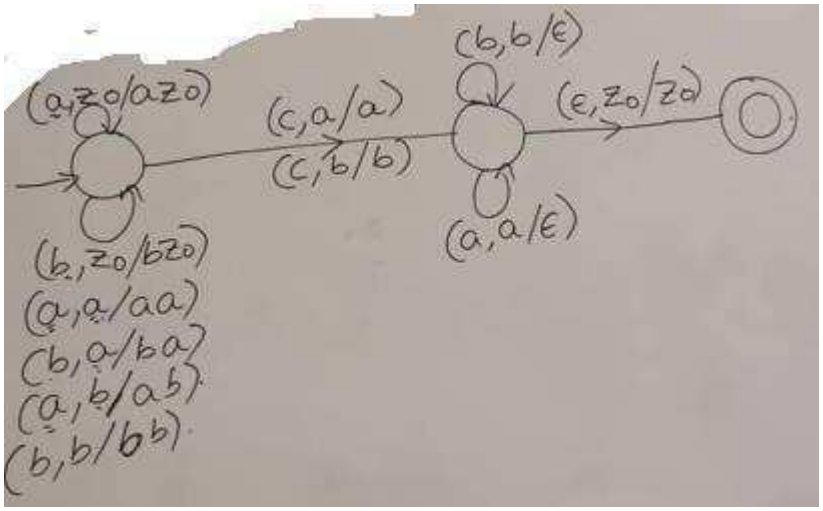
then we must B in a similar fashion (replacing initial occurences of A2)
B -> 1 A1 | 0 A1 A1 | 1 B A1 | 0 A1 B A1 |
1 A1 B | 0 A1 A1 B | 1 B A1 B | 0 A1 B A1 B

And now we have the following grammar:

A1 -> 1 A2 | 0 A1 A2 | 1 B A2 | 0 A1 B A2 | 0
A2 -> 1 | 0 A1 | 1 B | 0 A1 B
B -> 1 A1 | 0 A1 A1 | 1 B A1 | 0 A1 B A1 |
1 A1 B | 0 A1 A1 B | 1 B A1 B | 0 A1 B A1 B is in GNF.

(9)

14.



(9)

PART E
Answer any four full questions

15.

a.

The Turing machine can be thought of as finite control connected to a R/W (read/write) head. It has one tape which is divided into a number of cells. The block diagram of the basic model for the Turing machine is given in Fig. 9.1.

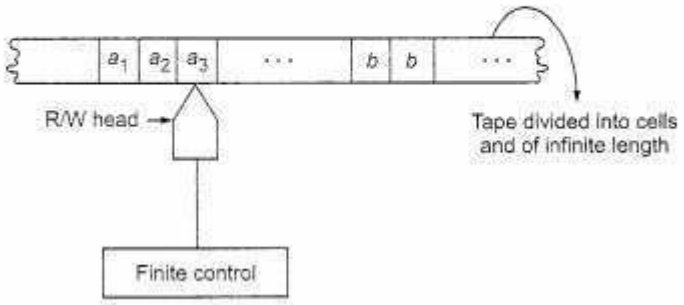


Fig. 9.1 Turing machine model.

Each cell can store only one symbol. The input to and the output from the finite state automaton are effected by the R/W head which can examine one cell at a time. In one move, the machine examines the present symbol under the R/W head on the tape and the present state of an automaton to determine

(i) a new symbol to be written on the tape in the cell under the R/W head,
(ii) a motion of the R/W head along the tape: either the head moves one cell left (L), or one cell right (R),
(iii) the next state of the automaton, and
(iv) whether to halt or not.

The above model can be rigorously defined as follows:

Definition A Turing machine M is a 7-tuple, namely $(Q, \Sigma, \Gamma, \delta, q_0, b, F)$, where

1. Q is a finite nonempty set of states.
2. Γ is a finite nonempty set of tape symbols.
3. $b \in \Gamma$ is the blank.
4. Σ is a nonempty set of input symbols and is a subset of Γ and $b \notin \Sigma$.
5. δ is the transition function mapping (q, x) onto (q', y, D) where D denotes the direction of movement of R/W head: $D = L$ or R according as the movement is to the left or right.
6. $q_0 \in Q$ is the initial state, and
7. $F \subseteq Q$ is the set of final states.

(5)

b. A linear bounded automaton can be defined as an 8-tuple (Q, X, ∑, q_0, ML, MR, δ, F) where −

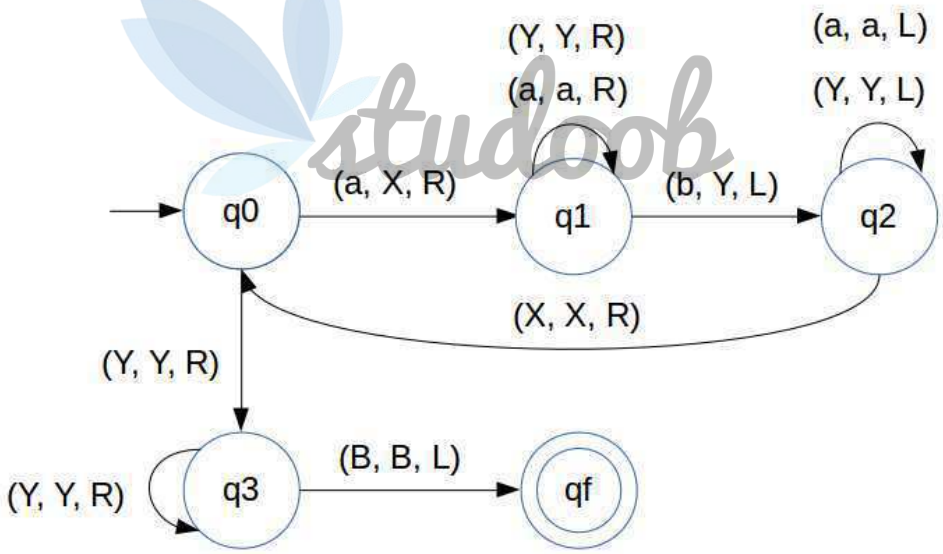
- **Q** is a finite set of states
- **X** is the tape alphabet

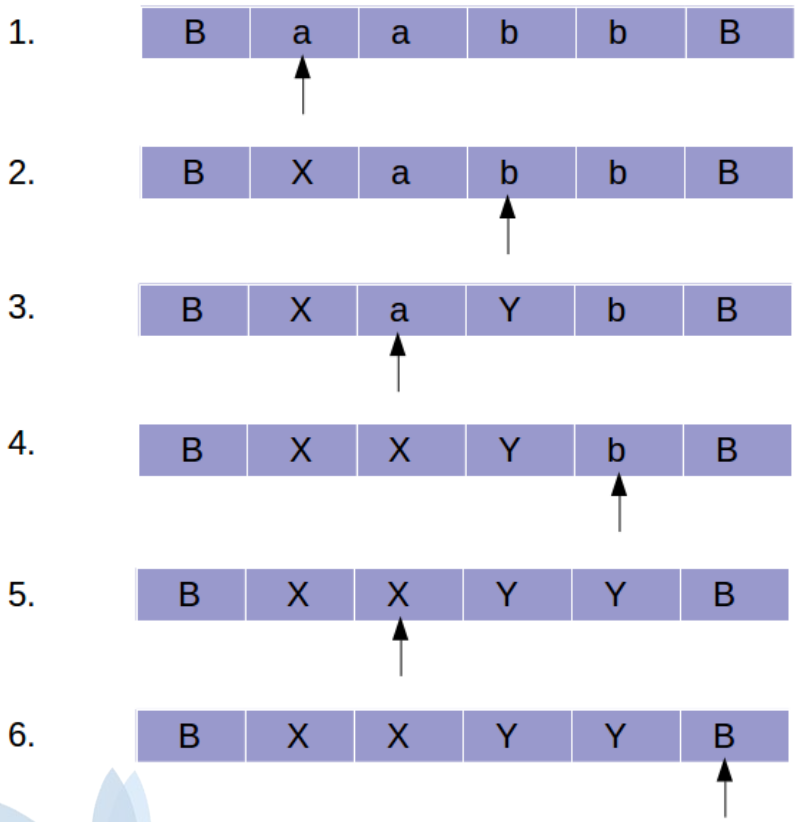
- $\sum$ is the input alphabet
- $\mathbf{q_0}$ is the initial state
- $\mathbf{M_L}$ is the left end marker
- $\mathbf{M_R}$ is the right end marker where $M_R \neq M_L$
- **δ** is a transition function which maps each pair (state, tape symbol) to (state, tape symbol, Constant 'c') where c can be 0 or +1 or -1
- **F** is the set of final states

A deterministic linear bounded automaton is always **context-sensitive** and the linear bounded automaton with empty language is **undecidable.**

(5)

16.





TAPE movement is shown below:
Now we will explain the logic step by step:

1. Input is given in the TAPE as "aabb", "B" is BLANK symbol. R/W head will point to first "a" (from left)
2. TM will mark first "a" with "X" (custom), and move to right, skipping all "a"s and will stop at first "b"
3. TM will mark "b" with "Y" (custom) and move to left till it finds "X" and then stop at one step ahead at "a"
4. TM will mark first "a" with "X", and move to right, skipping all "a"s and "Y"s and will stop at "b"
5. TM will mark "b" with "Y" and move to left till it finds "X" and then after it there is "Y" that means all "a"s are finished
6. TM will move to right to check whether all "b"s are finished. If R/W reaches "B"(BLANK) that means all "b"s are also finished.***That means TM has acceted the language***

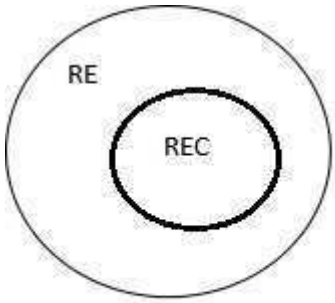
(10)

17. a. **Recursive Enumerable (RE) or Type -0 Language**

RE languages or type-0 languages are generated by type-0 grammars. An RE language can be accepted or recognized by Turing machine which means it will enter into final state for the strings of language and may or may not enter into rejecting state for the strings which are not part of the language. It means TM can loop forever for the strings which are not a part of the language. RE languages are also called as Turing recognizable languages.

Recursive Language (REC)

A recursive language (subset of RE) can be decided by Turing machine which means it will enter into final state for the strings of language and rejecting state for the strings which are not part of the language. e.g.; L= $\{a^n b^n c^n | n>=1\}$ is recursive because we can construct a turing machine which will move to final state if the string is of the form anbncn else move to non-final state. So the TM will always halt in this case. REC languages are also called as Turing decidable languages. The relationship between RE and REC languages can be shown in Figure



Closure Properties of Recursive Languages

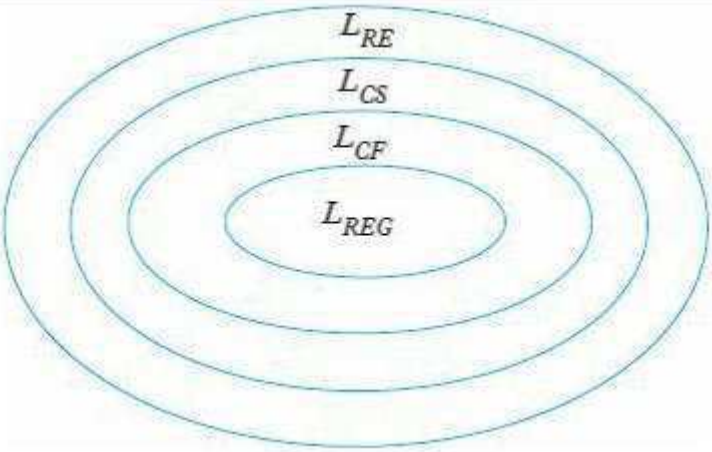
- **Union**: If L1 and If L2 are two recursive languages, their union L1∪L2 will also be recursive because if TM halts for L1 and halts for L2, it will also halt for L1∪L2.
- **Concatenation:** If L1 and If L2 are two recursive languages, their concatenation L1.L2 will also be recursive.
- **Kleene Closure:** If L1is recursive, its kleene closure L1* will also be recursive.
- **Intersection and complement**: If L1 and If L2 are two recursive languages, their intersection L1 ∩ L2 will also be recursive

As opposed to REC languages, RE languages are not closed under complementon which means complement of RE language need not be RE.

(5)

b. Chomsky hierarchy. Noam Chomsky, a founder of formal language theory, provided an initial classification into four language types, type 0 to type 3. Type 0 languages are those generated by unrestricted grammars, that is, the recursively enumerable languages. Type 1 consists of the context-sensitive languages, type 2 consists of the context-free languages, and type 3 consists of the regular languages. As we have seen, each language family of type i is a proper subset of the family of type

i − 1. A diagram exhibits the relationship clearly

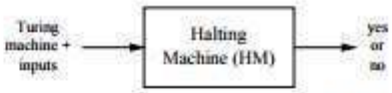


(5)

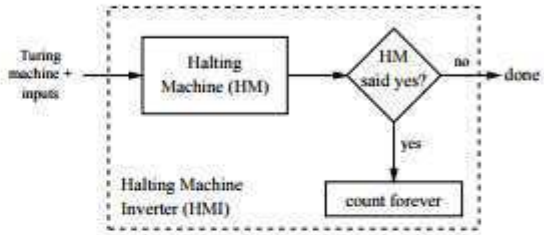
18. The halting problem is easy to state and easy to prove undecidable.
The problem is this:

given a Turing machine and an input to the Turing machine, does the Turing machine finish computing inafinite number of steps (a finite amount of time)? In order to solve the problem, an answer, either yes or no, must be given in a finite amount of time regardless of the machine or input in question. Clearly some machines never finish. For example, we can write a Turing machine that counts upwards starting from one.
You may find the proof structure for undecidability of the halting problem easier to understand if you first think about a related problem with which you may already be familiar, the Liar's paradox (which is at least 2,300 years old). In its stengthened form, it is the following sentence: "This sentence is not true."

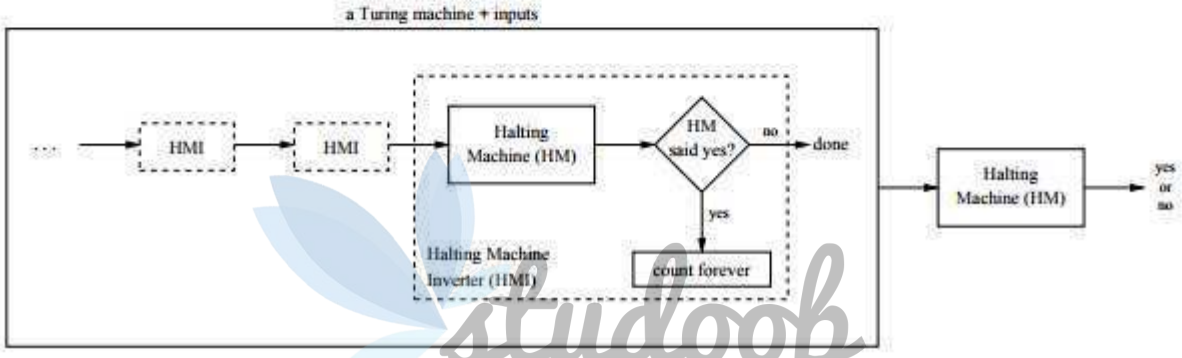
To see that no Turing machine can solve the halting problem, we begin by assuming that such a machine exists, and then show that its existence is self-contradictory. We call the machine the "Halting Machine," or HM for short. HM is a machine that operates on another machine and its inputs to produce a yes or no answer in finite time: either the machine in question finishes in finite time (HM returns "yes"), or it does not (HM returns "no"). The figure illustrates HM's operation.



From HM, we construct a second machine that we call the HM Inverter, or HMI. This machine inverts the sense of the answer given by HM. In particular, the inputs are fed directly into a copy of HM, and if HM answers "yes," HMI enters an infinite loop. If HM answers "no," HMI halts. A diagram appears to the right.



The inconsistency can now be seen by asking HM whether HMI halts when given itself as an input (repeatedly), as shown below. Two copies of HM are thus being asked the same question. One copy is the rightmost in the figure below and the second is embedded in the HMI machine that we are using as the input to the rightmost HM. As the two copies of HM operate on the same input (HMI operating on HMI), they should return the same answer: a Turing machine either halts on an input, or it does not; they are deterministic.



Let's assume that the rightmost HM tells us that HMI operating on itself halts. Then the copy of HM in HMI (when HMI executes on itself, with itself as an input) must also say "yes." But this answer implies that HMI doesn't halt (see the figure above), so the answer should have been no!

Alternatively, we can assume that the rightmost HM says that HMI operating on itself does not halt. Again, the copy of HM in HMI must give the same answer. But in this case HMI halts, again contradicting our assumption.

Since neither answer is consistent, no consistent answer can be given, and the original assumption that HM exists is incorrect. Thus, no Turing machine can solve the halting problem.

(10)