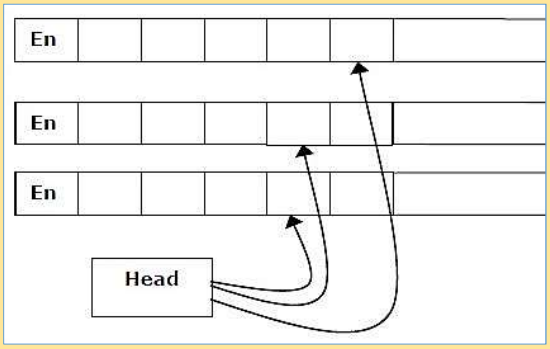
MODULE 6

MULTI – TAPE TURING MACHINE

- Multi-tape Turing Machines have multiple tapes where each tape is accessed with a separate head.
- Each head can move independently of the other heads. Initially the input is on tape 1 and others are blank.
- At first, the first tape is occupied by the input and the other tapes are kept blank.
- Next, the machine reads consecutive symbols under its heads and the TM prints a symbol on each tape and moves its heads.

A Multi-tape Turing machine can be formally described as a 6-tuple $(Q, T, B, \delta, q_o, F)$ where



Q is a finite set of states

T is the tape alphabet

B is the blank symbol

q_o is the initial state

F is the set of final states

δ is a relation on states and symbols where

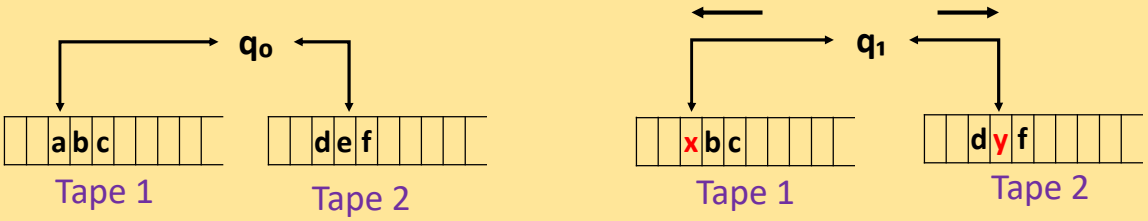
$\delta: Q \times T^n \rightarrow Q \times T^n \times \{L, R\}^n$ where n is the number of tapes

➢Every Multi-tape Turing machine has an equivalent single-tape Turing machine.

Example

n = 2

$\delta(q_o, a, e) = (q_1, x, y, L, R)$



NON-DETERMINISTIC TURING MACHINE

- Only the difference is in transition function

➢ **δ of Deterministic TM**

$Q \times \Sigma \rightarrow T \times (R/L) \times Q$

➢ **δ of Deterministic TM**

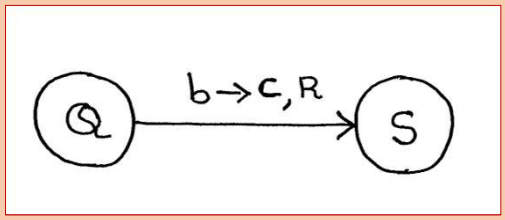
$Q \times \Sigma \rightarrow P \{ T \times (R/L) \times Q \}$ where P = power set

- In a particular state, up on reading a particular input symbol, the Non Deterministic TM will go to more than one state.

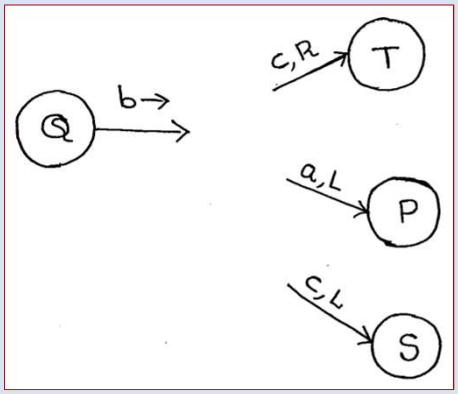
Prepared By Mr.EBIN PM, AP, IESCE EDULINE 5

Deterministic & Non – Deterministic TM

Deterministic TM



Non-Deterministic TM

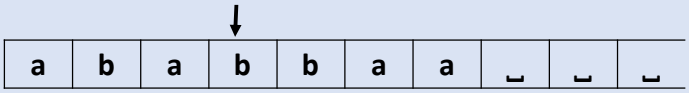


Prepared By Mr.EBIN PM, AP, IESCE 6

❖Configuration of Deterministic and No-Deterministic TM

Configuration :

- ➢It is a way to represent the entire state of a TM at a moment during computation
- ➢It is a string which shows
- The current state
- The current position of the head
- The entire tape contents

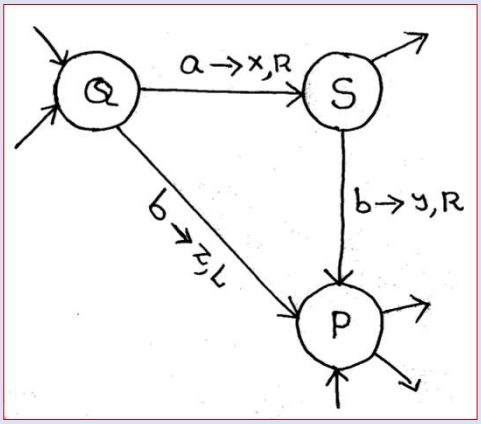


abaQbbaa

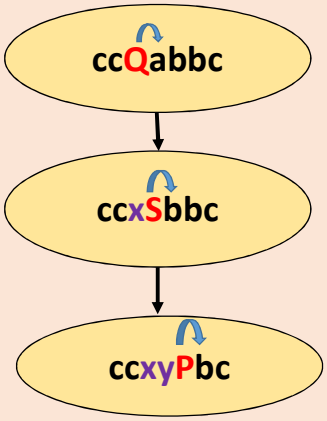
Prepared By Mr.EBIN PM, AP, IESCE EDULINE 7

Deterministic TM and its computation history using Configuration

Deterministic TM



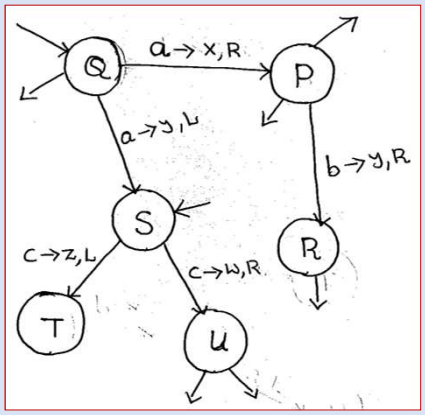
Configuration



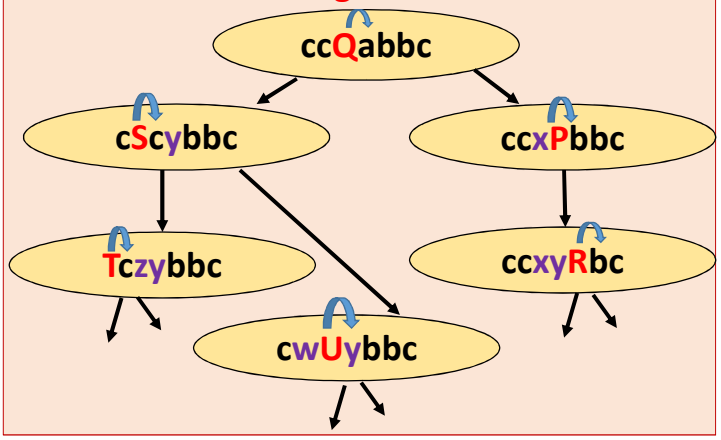
Prepared By Mr.EBIN PM, AP, IESCE 8

Non-Deterministic TM and its computation history using Configuration

Non- Deterministic TM



Configuration



- With Non determinism, at each moment in the computation, there can be more than one successor configuration.

❖Outcome of Non-Deterministic TM

- **Accept** – If any branch of the computation accepts, then the Non-Deterministic TM will accept
- **Reject** – If all branches of the computation halt and reject (ie., no branches accept, but all computations halt) then the Non-Deterministic TM rejects.
- **Loop** – Computation continues but accept is never encountered. Some branches in the computation history are infinite.

RECURSIVE & RECURSIVE ENUMERABLE LANGUAGES

The Turing Machine may

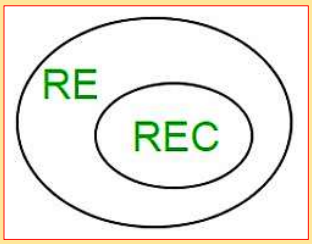
- Halt & Accept the input
- Halt and Reject the input
- Never Halt (Loop)

❖Recursive Language (REC)

- REC languages are also called as Turing decidable languages
- A language "L" is said to be recursive, if there exist a Turing Machine which will accept all the strings in "L" and reject all the strings not in "L".
- The Turing Machine will halt every time and give the answer(accepted or rejected) for each and every string input.

❖Recursive Enumerable Language (RE)

- RE languages are also called as Turing recognizable languages.
- A language "L" is said to be Recursively Enumerable Language, if there exist a Turing Machine which will accept(and therefore halt) for all the input strings which are in "L".
- But may or may not halt for all input strings which are not in "L".
- Recursive language is a subset of Recursive Enumerable Language.



❖Properties of Recursive Language (REC)

- **Union:** If L_1 and If L_2 are two recursive languages, their union $L_1 \cup L_2$ will also be recursive because if TM halts for L_1 and halts for L_2, it will also halt for $L_1 \cup L_2$.
- **Concatenation:** If L_1 and If L_2 are two recursive languages, their concatenation $L_1.L_2$ will also be recursive.
- **Kleene Closure:** If L_1 is recursive, its kleene closure L_1^* will also be recursive.
- **Intersection:** If L_1 and If L_2 are two recursive languages, their intersection $L_1 \cap L_2$ will also be recursive.
- **Complement:** complement of recursive language L_1 which is $\sum^* - L_1$, will also be recursive

❖Properties of Recursive Enumerable Language (RE)

- **Union:** If L_1 and If L_2 are two recursive enumerable languages, their union $L_1 \cup L_2$ will also be recursive enumerable
- **Concatenation:** If L_1 and If L_2 are two recursive enumerable languages, their concatenation $L_1.L_2$ will also be recursive enumerable
- **Kleene Closure:** If L_1 is recursive enumerable, its kleene closure L_1^* will also be recursive enumerable
- **Intersection:** If L_1 and If L_2 are two recursive enumerable languages, their intersection $L_1 \cap L_2$ will also be recursive enumerable.
- **Complement:** RE languages are not closed under complement, which means complement of RE language need not be RE.

DECIDABILITY & UNDECIDABILITY

❖Decidable Language

- A language L is decidable if it is a recursive language.
- All decidable languages are recursive languages and vice-versa

❖Partially Decidable Language

- A language L is partially decidable, if L is a recursively enumerable Language

❖Undecidable Language

- A language is undecidable if it is not decidable
- An undecidable language may sometimes be partially decidable but not decidable.
- If a language is not even partially decidable, then there exists no Turing Machine for that language

Recursive Language	Turing Machine will always Halt
Recursively Enumerable Language	Turing Machine will halt some times & may not halt some times
Decidable Language	Recursive Language
Partially Decidable Language	Recursively Enumerable Language
Undecidable	No Turing Machine for that language

UNIVERSAL TURING MACHINE

- A Turing machine is said to be universal Turing machine if it can accept:
  - The input data, and
  - An algorithm (description) for computing.
- This is precisely what a general purpose digital computer does. A digital computer accepts a program written in high level language.
- Thus, a general purpose Turing machine will be called a universal Turing machine if it is powerful enough to simulate the behavior of any digital computer, including any Turing machine itself.

- A universal Turing machine is just a Turing machine whose programming simulates other Turing machines.
- That is, the input to the UTM is a description of a Turing machine T and an input for T, and the UTM simulates T on that input.
- It's universal in the sense that, for any problem that can be solved by Turing machines, you could either use a Turing machine that directly solves that problem, or you could use a UTM and give it the description of a TM that directly solves the problem.
- We can say that UTM is an interpreter for (all) Turing machines

- More precisely, a universal Turing machine can simulate the behavior of an arbitrary Turing machine over any set of input symbols.
- Thus, it is possible to create a single machine that can be used to compute any computable sequence.
- If this machine is supposed to be supplied with the tape on the beginning of which is written the input string of quintuple separated with some special symbol of some computing machine M, then the universal Turing machine U will compute the same strings as those by M.

- The model of a Universal Turing machine is considered to be a theoretical breakthrough that led to the concept of stored programmer computing device.
- Designing a general purpose Turing machine is a more complex task.
- Once the transition of Turing machine is defined, the machine is restricted to carrying out one particular type of computation.
- Digital computers, on the other hands, are general purpose machines that cannot be considered equivalent to general purpose digital computers until they are designed to be reprogrammed.

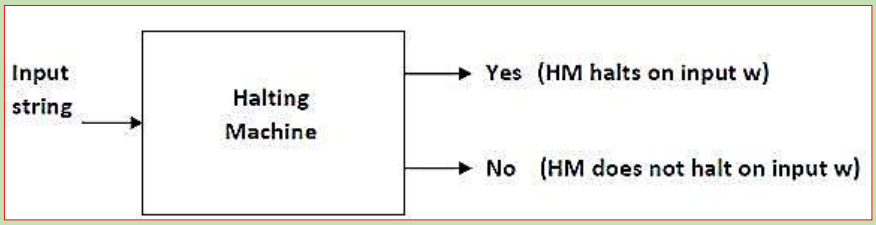
- By modifying our basic model of a Turing machine we can design a universal Turing machine.
- The modified Turing machine must have a large number of states for stimulating even a simple behavior. We modify our basic model by:
  - Increase the number of read/write heads
  - Increase the number of dimensions of input tape
  - Adding a special purpose memory
- All the above modification in the basic model of a Turing machine will almost speed up the operations of the machine can do.

- A number of ways can be used to explain to show that Turing machines are useful models of real computers.
- Anything that can be computed by a real computer can also be computed by a Turing machine.
- A Turing machine, for example can simulate any type of functions used in programming language.
- Recursion and parameter passing are some typical examples.
- A Turing machine can also be used to simplify the statements of an algorithm.

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 input".
- Input – A Turing machine and an input string w.
- Problem – Does the Turing machine finish computing of the string w in a finite number of steps? The answer must be either yes or no.
- Proof – At first, we will assume that such a Turing machine exists to solve this problem and then we will show it is contradicting itself. We will call this Turing machine as a Halting machine that produces a 'yes' or 'no' in a finite amount of time.

- If the halting machine finishes in a finite amount of time, the output comes as 'yes', otherwise as 'no'. The following is the block diagram of a Halting machine



- This is an undecidable problem because we cannot have an algorithm which will tell us whether a given program will halt or not in a generalized way i.e by having specific program/algorithm.

- In general we can't always know that's why we can't have a general algorithm.
- The best possible way is to run the program and see whether it halts or not.
- In this way for many programs we can see that it will sometimes loop and always halt.
- Thus one implication of the halting problem is that there can be no computer programs (Turing machines) that check whether or not any arbitrary computer program stops for a given input