20/10/2017

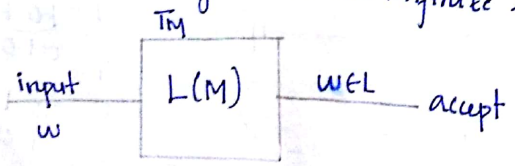
6. Recursive & Recursively Enumerable Language

~~Construct~~

~~Recursively enum ge and Recursive A language L is~~

A language L is said to be recursively EL if there exist a TM that accept it, ie, there exist a turing m/c M such that for every $w \in L$

$q_0 w \vdash_M^* x_1 q_f x_2$ with q_f is a final state and x_1 and x_2 are strings after processing w. ~~It~~ This definition says nothing about what happens for w not an element of L. It may be the m/c halts in a non-final state/it never halts and goes into an infinite loop

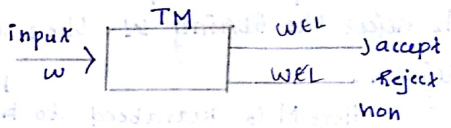


Rec E

Recursive Language:

A language L is called recursive if for some turing machine 'm' such that,

- If $w \in L$ then m accept w and halt
- If $w \notin L$ then m reject w and halt on non-final state



31/10/2017

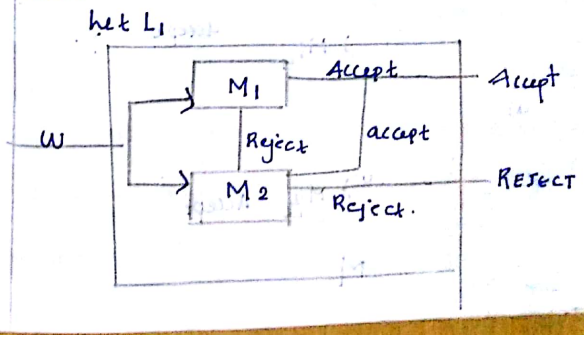
closure properties of Recursive and Recursively Enumerable language

Property 1

The union of two recursive languages are recursive

[recursive language is closed under union]

PROOF

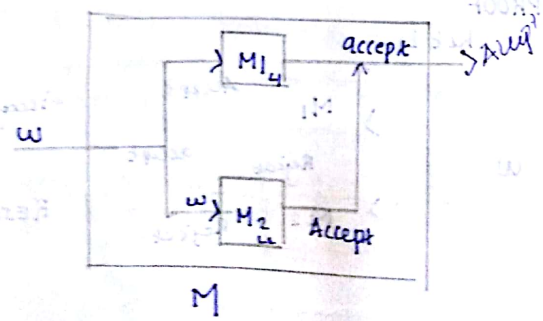


Let L_1 be a recursive language accepted by M_1. L_2 be a recursive language accepted by M_2. We construct a new TM that accepts $L_1 \cup L_2$. If w is accepted by M_1 then M accept. If M_1 reject w then the m/c M stimulates M_2 and accepts if M_2 accept. Since both M_1 and M_2 reject the string w then the m/c M reject w

Here M is guaranteed to ~~halt~~ halt on all i/p's

Property 2

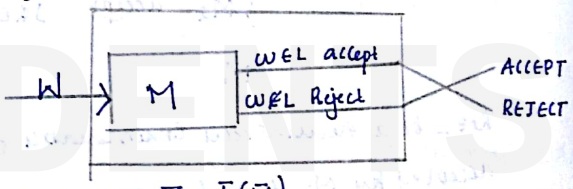
Union of two recursively enumerable language (recursively enumerable language are closed under union)



Let L_1 be a recursively enumerable language accepted by M_1 and L_2 be a recursively enumerable language accepted by M_2. We construct a new TM M such that it accepts the language $L_1 \cup L_2$ is shown in above figure

property - 3

If L is recursive language then complement of L ie L' is also recursive [Recursive languages are closed under complementation.]



$\overline{M} = L(\overline{M})$

1/11/2017 Let L be a recursive language accepted by TM M and $\overline{L}$ is complement of L

" " $\overline{M}$

TM M modifies as follows to create $\overline{M}$

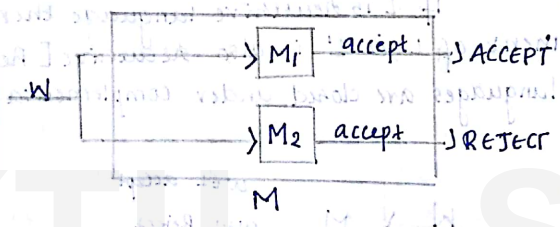
* Accepting states of M are made non accepting states of $\overline{M}$
* All non accepting states of M are made

accepting state of M

* Since M guaranteed to halt on all i/p's w so, $\overline{M}$. ∴ $\overline{M}$ is also a recursive language.

Property-4

If both language L and it's complement are recursively enumerable then L is recursive

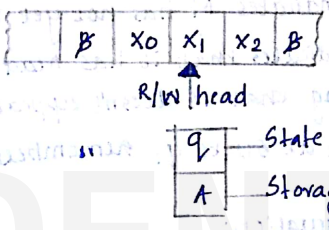


Let L be a recursively enumerable language accepted by M. Then L complement is the complement of L accepted by TM M_2 we create a new TM M using M_1 and M_2 shown in above figure. From above figure it is guaranteed that M halt on all inputs of w. ∴ ~~M~~ the language accepted by M ie L is recursive.

Programming Techniques In TM

1) Storage at State / finite Control

In this type of TM states in the control unit not only to represent current position of the prgm but also to hold a finite amount of data Below figure suggest this technique.



A state in TM is a tuple [q, A] where q is a state and A is the stored value at state q. The transition funct^n (δ) can be viewed as follows

$$\delta([q, A], X) = ([P, B], Y, D)$$

- [q, A]: Current State + Stored Value
- X: Input
- P: next state + Stored value
- Y: replacing symbol
- D: direction

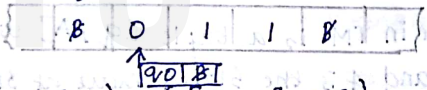
eg: Design a TM that accept the string generated by the language $10^* + 01^*$ using storage at the state concept

Solution: Let M be a TM corresponds to the language $10^*, 01^*$

The state in the Q remembers what the TM is doing.

* State q_0 indicates M has not yet read its first symbol
* State q_1 indicates that it has read the first symbol and checking that it doesn't appear else where.
* The storage at state q_1 remembers the first symbol read.

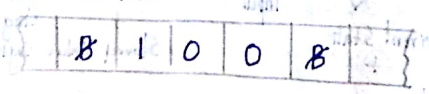
Initial configuration:



$\delta([q_0, B], 0) = ([q_1, 0], 0, R)$

$\delta([q_1, 0], 1) = ([q_1, 0], 1, R)$

$\delta([q_1, 0], B) = ([q_2, B], B, H)$

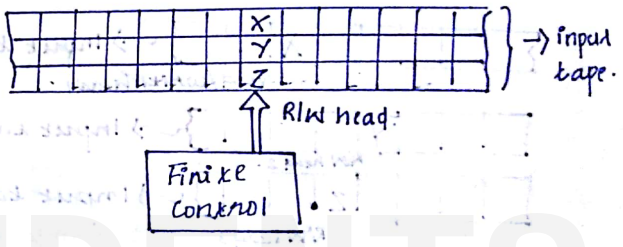


$\delta([q_0, B], 1) = ([q_1, 1], 1, R)$

$\delta([q_1, 1], 0) = ([q_1, 1], 0, R)$

$\delta([q_1, 1], B) = ([q_2, B], B, H)$

2) TM with multiple tracks on single Tape.

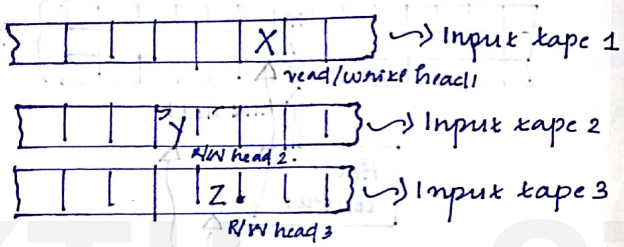


In this technique the i/p tape of the TM is divided into several tracks each track can hold (alt) . Each track are divided into several cells. Each cell can hold one symbol The Tape alphabet of TM consist of tuples with one component for each track below fig illustrates . TM with ~~machine~~ multiple tracks on a single input tape, A delta function (δ) on a multitrack TM can be represented as

$\delta(q, [X, Y, Z]) = (P, [A, B, C], D)$

- current state
- input symbol
- next state
- replacing symbols
- direction of head movmt (R/L)

3) Multitape TM



The device has finite control unit and multiple no: of input tapes. Each tape is divided into cells and each cell hold one symbol of the Finite tape ~~State~~ alphabet (Γ). The above fig illustrate multitape TM with three i/p tape. The transition function of multiple tape TM depends on current state and the symbol scanned by ~~Tap~~ Each of the tape head.

9/11/17

In 3-Tape TM δ can be represented as,

$\delta(q, [X, Y, Z] = (P, [A, D_1], [B, D_2], [C, D_3])$

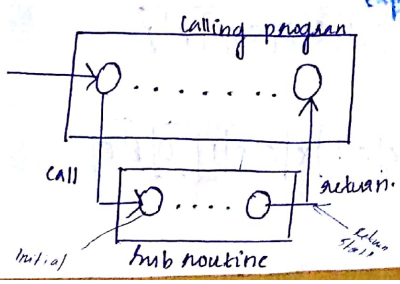
- current State
- i/p symbol scanned by R/W head 1/2/3 respectively
- next State
- Direction of head movement (Right/left)

A, B, C – replacing symbol

- In one move, a multi-tape TM does the following

1. The control enters into a new state (P)
2. On each tape, a new symbol is written on the cell being scanned
3. Each head make a move which can be ~~written~~ right or left

Sub routine.



A subroutine is a set of states that performs some useful function.

- This set of state includes a start state and another state that serves as the return state.
- The call of the subroutine occurs whenever there is a transition to the initial state of the subroutine.

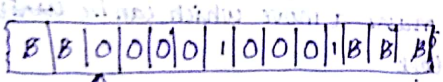
Eg: Design a TM that performs multiplication

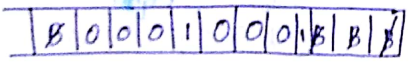
Input: Strings with $0^m 1 0^n$ on input tape.

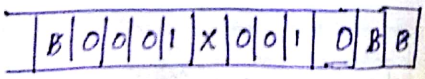
Output: computation ends with 0^{mn} on the tape

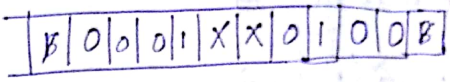
Initial configuration.

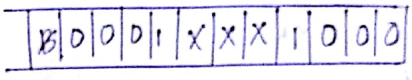
Assume input = $0^4 1 0^3$

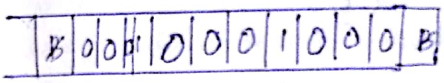


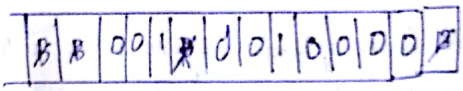




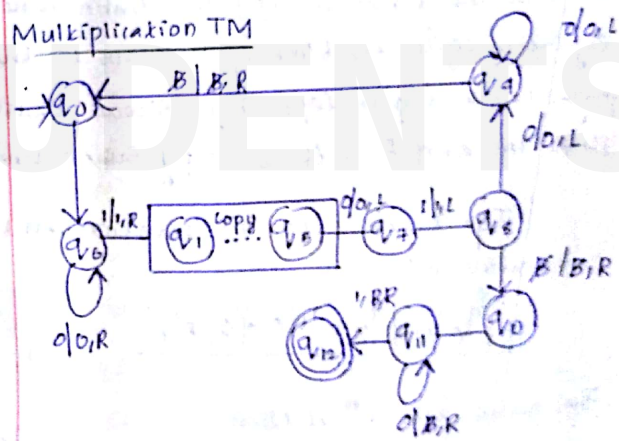








Multiplication TM



Copy Subroutine TM

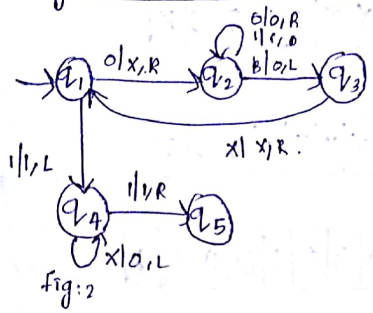


fig:2

The ~~Hault~~ heart of this algorithm is a subroutine we call it has a copy. This subroutine algorithm copies the block of n zeros to the right end of input tape. Fig:1 shows multiplicatn TM and fig:2 shows copy subroutine TM.

Tape status after 1st iteration (first subroutine call.

B	0	0	0	1	0	0	0	1	0	0	0	B	B

Final output (after n^{th} iteration)

0	0	0	0	0	0	0	0	0	0	0	0

Enumarating TM.

$M \in N$ A TM can be identified with integers

A TM $M = (Q, \Sigma, \Gamma, \delta, B, q_0, F)$ can be represented as a binary strings. For that we must assign different integers to the state, tape symbols & direction, Each transition function in enumarated TM can be represented by $\delta(q_i, X_j) = (q_K, X_l, D_m)$ then code for each function is

$$0^i 1 0^j 1 0^k 1 0^l 1 0^m$$

The compte code for TM M consist of set of transmitions code seperated by pair of ones.

$$C_1 11 C_2 11 C_3 11 \cdots 11 C_n$$

eg: Suppose the TM M ~~equal to~~ $= (Q, \Sigma, \Gamma, \delta, B, q_0, F)$

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

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

$\Gamma = \{0, 1, B\}$

$\delta(q_1, 1) = (q_3, 0, R)$

$\delta(q_3, 0) = (q_1, 1, R)$

$\delta(q_3, 1) = (q_2, 0, R)$

To enumarate above TM first we must assign different integers (or binary string) to the states, tape symbol & direction

Enumarating symbols	Tape symbols	Direction
$q_1 = 0$	0 = 0	L = 0
$q_2 = 00$	1 = 00	R = 00
$q_3 = 000$	B = 000	

Code for first transition funct^n

$\delta(q_1, 1) = (q_3, 0, R)$

$C_1 = 0100100010100$

Code for 2nd transition funct^n

$\delta(q_3, 0) = (q_1, 1, R)$

$C_2 = 000101010010 0$

Code for 3rd transition funct^n.

$\delta(q_3, 1) = (q_2, 0, R)$

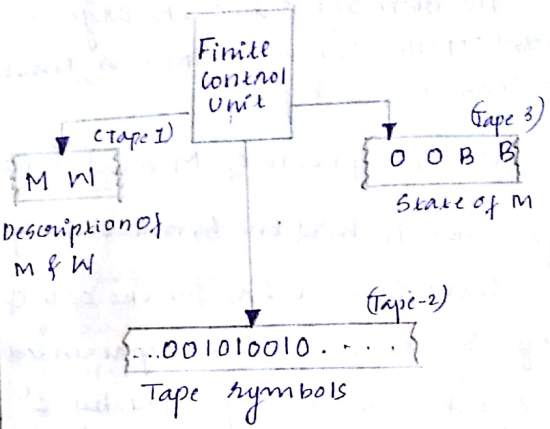
$C_3 = 0001001 0010100$

TM = $C_1 11 C_2 11 C_3 11$

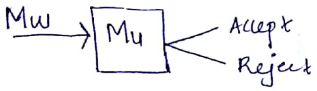
$= 0100100010\ 100 11\ 000101010010011$

$000100...$

UNIVERSAL TM



A TM is a special purpose computer once δ is defined the m/c is restricted to carry out one particular type of computation. A reprogrammable TM called universal TM (M_U) is an automaton that given as input the description of any TM M and a string w. M_U can simulate the computat^n of M on w



Organisation of a Universal TM

- A universal TM is a multi tape TM

The above fig shows the organisation of universal TM. Here universal TM is a multi tape TM consists of 3 tapes

* First tape holds description of M and input w.
* Second tape is used to hold the simulated tape of M using the same format as for the code of M. ie, tape symbol x_i of M will be represented as The tape symbols will be seperated by '1'
* Third tape of Mu hold the current state of M. In which the state q_i represented by i no: of zeros.

The operation of Mu can be summarised as follows

Step 1: Check code for M is valid for some TM 'M'. If not, halt without accepting

Step 2: Initialise second tape to contain the input w in its coded form

eg:-

i/p	0	1	B
coded format	0	00	000

Step 3: place start state of M to the third tape. Then move Mu's instead of second tape to the 1st simulated cell

Step 4: To simulate a move of M, Mu searches on its 1st tape for a transition $0^i 1 0^j 1 0^k 1 0^l 1 0^m$ such that 0^i in the state on tape 3. 0^j is the tape symbol of M that begins at position head

This transition is:

- changes the content of tape 3 to 0^k
- Replace 0^j with 0^l on tape 2
- Move head on tape 2 to the left/right depending on the value of 0^m

Step 5: If M has no transition that matches simulated symbol and tape in step 4. M halt in the simulated configuration M

6: If M enter its accepting state the Mu accept

→ Non-Deterministic TM (NTM)

A non-deterministic TM (NTM) ~~dif~~ differs from the deterministic TM (DTM) by having a transition function δ such that for each state q and tape symbol x, $\delta(q, x)$ is a set of ~~tuples~~ triples.

$$\delta(q,x) = \{(q_1, Y_1, D_1), (q_2, Y_2, D_2), \ldots (q_k, Y_k, D_k)\}$$

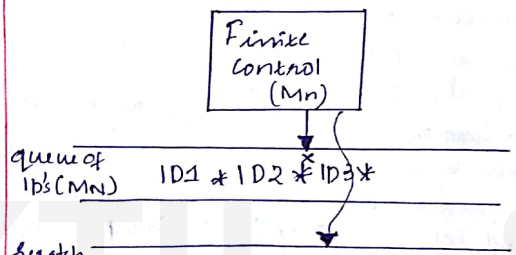
where K is any +ve integer

Theorem

Equivalance of DTM & NTM

If M_N is a NTM, then there is a DTM M_D such that $L(M_N) = L(M_D)$

M_D is designed multitape TM shown in below figure.



The first tape of M_D holds a sequence of ID's of M_N, including the state of Mn 1 ID of M_N is marked as current ID, whose successor ID's are in the processor of being discovered All ID's to the left of the current one have been explored and can be ignored subsequently.

The process the current ID M_D does the following

1. M_D examines/check the state and scan symbol of the current ID. If the state in the current id is accepting Then M_D accepts and simulates M_N
2. However if the state is not accepting then state-symbol combination has k moves Then M_D uses its second tape to copy the ID.
3. M_D modifies each of k ID according to a different one of the k-choices of moves that M_N has from its current ID
4. M_D returns to the marked current ID, erases the mark and moves the mark to the next ID to the right. The cycle then repeat with step 1.

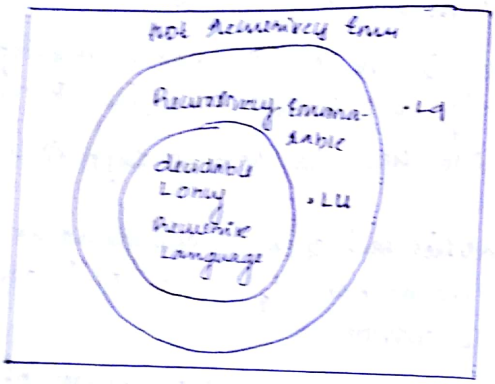
3/10/17

Decidability

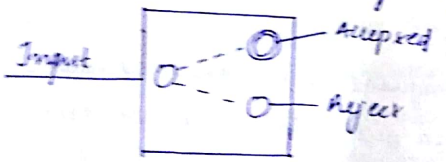
A language L is called decidable if there is a TM which accepts and hault on every i/p string.

- Every decidable languages are recursive language
- Every " " " Turing acceptable
- Recursive languages are corresponds to algorithm(a well defined sequence of step to solve a problem.

Below figure shows the relationship b/w Recursively and non recursively ~~Enumerable~~ Languages.



For a decidable language, for each i/p string the TM halts either at accept/at the reject state as depicted in the following diagram.



→ Eg:s of decidable Language

Is a number 'P' ~~is~~ prime

Given a regular language L and string w

If $w \in L$.

- Does a given DFA accept the empty Langu

Undecidable language

A language L is called Undecidable if it is not a Recursive language.

- A decision problem 'P' is called undecidable if the language L of all ~~'S'~~ 'Yes' instance to P is not decidable
- Undecidable language are not recursively language but sometimes there may be recursively Enumerable.
- un Recursively Enumerable Lang corresponds to Procedure.

Example

- The halting problem of TM
- Post Correspondance Problem (PCP)
- ~~Clique~~ clique problem
- Vertex Cover problem.
- 3 CNF problem.

⇒ TM halting Problem:-

Input to halting Problem: $\langle M, w \rangle$, a TM M and i/p string w

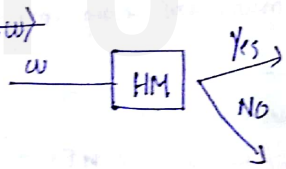
problem definition: does a TM (M) finish computing of the string w in a finite no: of ~~states~~ steps?

The answer must be either Yes or No

PROOF:

At first, we will assume that such ~~proble~~ TM exist to solve this problem we call it as Halting Machine (HM) That produces 'Yes' or 'No' in a finite amount of time. If the halting mlc in a finite " " " , the o/p comes Yes otherwise No

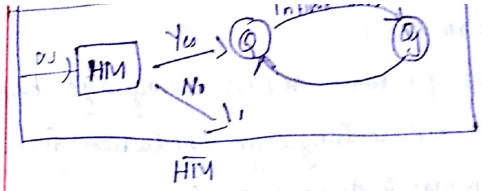
The following is the block diagram of HM



Now we will define an inverted ~~halting~~ ~~HM~~ n/s $\overline{HM}$ as follows,

* If HM returns Yes then $\overline{HM}$ loops forever
* If HM returns No then $\overline{HM}$ halts.

The following fig is block diagram of $\overline{HM}$.



$\overline{HM}$

further a machine HM_2 which i/p itself is constructed as follows. If HM_2 halts on i/p loop forever else halt.

Here we got a contradiction Hence the halting problem is undecidable.

Chomsky Classification of Grammars.

According to Noam Chomsky there are four types of grammars they are type 0, type-1, type-2 and type-3. The following table shows how they are diffe from each other.

Grammar type	Grammar Accepted	Language Accepted	Automaton
TYPE 0	~~Gente~~ Unrestricted Grammar	Recursively Enumerated language	Turing Machine (TM)
TYPE 1	Context Sensitive Grammar	Context Sensitive language	Linear bounded Automata (LBA)
TYPE 2	Context free Grammar	Context free language.	Push down Automata (PDA)
TYPE 3	Regular Grammar	Regular language	Finite state Automaton. FSA

TYPE-3 GRAMMAR:

It generates regular language. Regular grammars are linear languages. Production format of linear grammar is defined as follows:

$X \to a$

$X \to aX$ where $X \in V$ and $a \in T$

eg: Let $G = (V, T, S, P)$ be a regular grammar then $P = \begin{cases} S \to a \\ S \to aS \end{cases}$

Type-3 language are accepted by finite state automaton.

TYPE-2 Grammars

It generate context free language. It is accepted by pushdown automaton. TYPE-2 grammar or context free grammar is defined as, $G = \{V, T, S, P\}$ where all productions are of the form $V \to (V \cup T)^*$ where $V \in V$ is a variable and T is a terminal.

eg: $L = a^n b^n$ where $n \geq$

Let CFG, $G = \{V, T, S, P\}$

$$P = \begin{cases} S \to aSb \\ S \to ab \end{cases}$$

Type-1 Grammar

Type-1 grammar generate context sensitive Grammar. It is accepted by Linear bounded automata. Type-1 grammar is defined as $G = \{V, T, S, P\}$ where all productions are of the form $\alpha A \beta \to \alpha \gamma \beta$ where $A \in V$ and $\alpha, \beta, \gamma \in (V \cup T)^*$

NOTE:

A string α, γ, β ~~be~~ may be empty but A must be non-empty

* The rule $n \to \epsilon$ is allowed if ~~S~~ doesn't appear on the RHS of any rule.

eg: $G = \{V, T, S, P\}$ $P = \begin{cases} S \to aBac \\ Ba \to cDcD \end{cases}$

TYPE-0 Grammar:

It generates recursively enumerable language. It is accepted by Turing Machine. $G = \{V, T, S, P\}$ be a type-0 grammar. Then all productions gives $\alpha \to \beta$ where α contains strings of terminals and Variable with atleast one Variable on α side.

Relationship blw Type-0, 1, 2, 3 grammar