For more notes visit https://collegenote.pythonanywhere.com

Introduction to Turing Machine

A Turing machine is an abstract machine, introduced by the English mathematician Alan Turing – 1936.This is a model of computation which provides much of the theoretical foundation for the modern computer.

Turing began by considering a human computer i.e. a human solving a problem with pencil and paper. The method of solving could be assumed to operate under these three rules

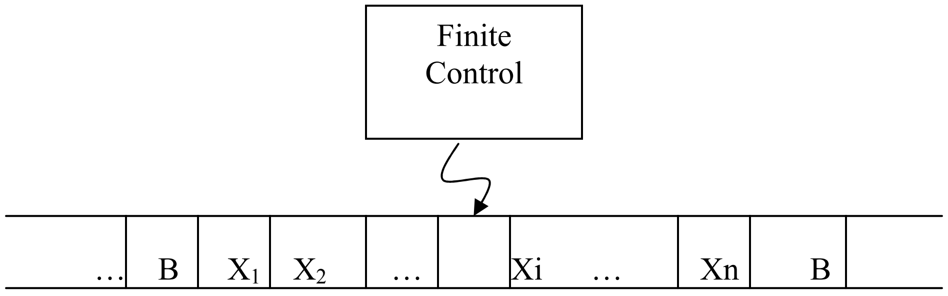
1. The only things written in paper are symbols from some fixed finite set – Alphabet
2. Each step taken by the computer depends only on the symbol he is currently examined – state of mind
3. His state of mind might change as a result of symbols he has seen or computation he has made,

Turing then set out to build an abstract machine that obeys these rules and can duplicate what he took to be the primitive steps carried out by a human computer during computation;

1. Examining individual symbol on the paper
2. Erasing a symbol or replacing it by another.
3. Transferring attention from one part of the paper to another

A Turing Machine will have

- finite alphabet of symbols (two alphabets – an input alphabet and a possibly larger alphabet for use during computation),
- A finite set of states, corresponding to the possible "state of mind" of human computer.
- A linear "tape" which is potentially infinite to both end.



The tape is marked off into squares, each of which can hold one symbol from the alphabet. If square has no symbol in it, then it contains blank. We think, reading and writing as being done by a tape head. A single move of TM is determined by the current state and current tape symbol and consist of three things.

1. Replacing the symbol in the current square by another, possibly different symbol.
2. Moving the tape head one square right or left or leaving it where it is.
3. Moving from the current state to another, possibly different state.

For more notes visit https://collegenote.pythonanywhere.com

The tape serves.

- The input device (Input is simply the string assumed to finite)
- The memory available for use during computation
- The output device (output is the string of symbols left on the tape at the end of computation)

The most significant difference between the Turing machine and the simpler machine (FA or PDA) is that- In a Turing machine processing a string is no longer restricted to a single left- to- right pass through the input. The tape head can move in both directions and erase or modify any symbol it encounters. The machine can examine part of the input , modify it , take time to execute some computation in a different area of the tape, return to re- examine the input , repeat any of these actions and perhaps stop the processing before it has looked at all the input.

Formal Definition of Turing Machine

A Turing Machine (TM) is a 7- tuple

$M=(Q,\sum,\Gamma,\delta,q_0,B,F)$ where.

Q : The finite set of states of the finite control / states

$\sum$: The finite set of input symbols

Γ : The complete set of tape symbols; $\sum$ is always subset of Γ

δ : The transition function . defined by

$Q\times\Gamma\to Q\times\Gamma\times(R,L,S)$ where R, L, S is the direction of movement of head – left or right or stationary

i.e. $\delta(q,x)=\delta(P,Y,D)$ means TM in state q and current tape symbol X, moves to next state p replacing tape symbol X with Y and move the head either direction or remains same cell of input tape.

q_0 :The start state , a member of Q

B : The blank symbol. This symbol B $\in\Gamma, B\notin\sum$

F : The set of final or accepting states , F$\subseteq$ Q.

Instantaneous Descriptions for TM:

The configuration of a TM is described by Instantaneous Description (ID) of TM like PDA. A string $X_1 X_2 ... X_{i-1}\ q\ X_i X_{i+1} ..... X_n$ represents the ID of TM in which

1. q is the state of TM
2. The tape head scanning the ith symbol from the left
3. $X_1 X_2 ..... X_n$ is the portion of tape between the leftmost and rightmost non-blank. If the head is to the left of leftmost nonblank or to the right of right most non-blank then some prefix or suffix of $X_1 X_2 .... X_n$ will be blank and i will be 1 or n respectively.

Moves of TM : The moves of TM, $M=(Q,\sum,\Gamma,\delta,q_0,B,F)$ is described by notation $\vdash$ for single move and $\vdash^*$ for Zero or more move as in PDA.

(a) For $\delta(q,X_i)=(P,Y,L)$ i.e next move is leftward then

$$X_1X_2....X_{i-1}qX_iX_{i+1_N}.......X_n \quad\vdash\quad X_1X_2....X_{i-2}pX_{i-1}YX_{i+1}.......X_n$$

For more notes visit https://collegenote.pythonanywhere.com

reflects the change of state from q to p and the symbol Xi is replaces with Y then tape head is positional at i-1(next scan is X_{i-1})

1. if i =1 , M moves to the blank to the left of X_1 i.e.
$X_1X_2....X_{i-1}qX_iX_{i+1_n}...X_n \vdash pBYX_2.....X_n$
or $qX_1X_2....X_{i-1}X_iX_{i+1_n}...X_n \vdash pBYX_2.....X_n$

2. If i =n, Y=B, then M moves to the state p and the symbol B written over X_n joins the infinite sequence of trailing blanks which does not appear in next ID as $X_1X_2...X_{n-1}qX_n \vdash X_1X_2....X_{n-2}pX_{n-1}$

(b) if $\delta(q, X_i) = (p, Y, R)$ i.e next move is rightward . then
$X_1X_2....X_{i-1}qX_iX_{i+1_n}...X_n \vdash X_1X_2....X_{i-1}YpX_{i+1}....X_n$ which reflects that symbol X_i is replaced with Y and the head has moved to cell i+1 .

1. If i = n, then i+1 cell holds blank which is not part of previous ID
$X_1X_2....X_{n-1}pX_n \vdash X_1X_2....X_{n-1}YpB$
2. If i =1, Y=B, then the symbol B written over X_i joins the infinite sequence of leading blanks and does not appear in next ID $qX_1X_2....X_n \vdash pX_2X_3....X_n$

Equivalently ID can be written as : (q, x$\underline{a}$y) where q is a state. x and y are strings on $\Gamma, a \in \Gamma$ and underlined symbol represents the tape head position

An Example : A TM accepting language $\{0^n1^n \mid n \geq 1\}$

- Given finite sequence of 0's and 1's on its tape preceded and followed by blanks.
- Alternatively, TM will change 0 to an X and then a 1 to a Y, until all 0's & 1's are matched .
- Starting at left end of the input, it repeatedly changes a 0 to an X and moves to the right over whatever 0's and Y's it sees until comes to a 1.
- It changes 1 to a Y , and moves left, over Y's and 0's until it finds X. At that point it looks for a 0 immediately to the right . If finds one 0 then changes it to X and repeats the process changing a matching 1 to Y.

Now TM will have
M= $(\{q_0, q_1, q_2, q_3, q_4\}, \{0,1\}, \{0,1, X, Y, B\}, \delta, q_0, B, \{q_4\})$
The transition rule for the move of M is described by following transition table .

State	0	1	X	Y	B
q_0	(q_1, X,R)	–	–	(q_3, Y, R)	-
q_1	(q_1, 0,R)	(q_2, Y,L)	–	(q_1, Y, R)	–
q_2	(q_2, 0, L)	–	(q_0, X, R)	(q_2, Y, L)	–
q_3	–	–	–	(q_3, Y, R)	(q_4, B, R)
q_4	–	–	–	–	

Acceptance of 0011 by M is described by following sequence of moves

For more notes visit https://collegenote.pythonanywhere.com

$q_00011 \vdash Xq_1 011 \vdash X0q_111 \vdash Xq_20Y1 \vdash q_2X0Y1$

$\vdash Xq_0 0Y1 \vdash XXq_1Y1 \vdash XXYq_11 \vdash XXq_2YY$

$\vdash Xq_2XYY \vdash XXq_0YY \vdash XXYq_3Y$

$\vdash XXYYq_3B \vdash XXYYBq_4B$ (Halt and Accept)

For string 0110

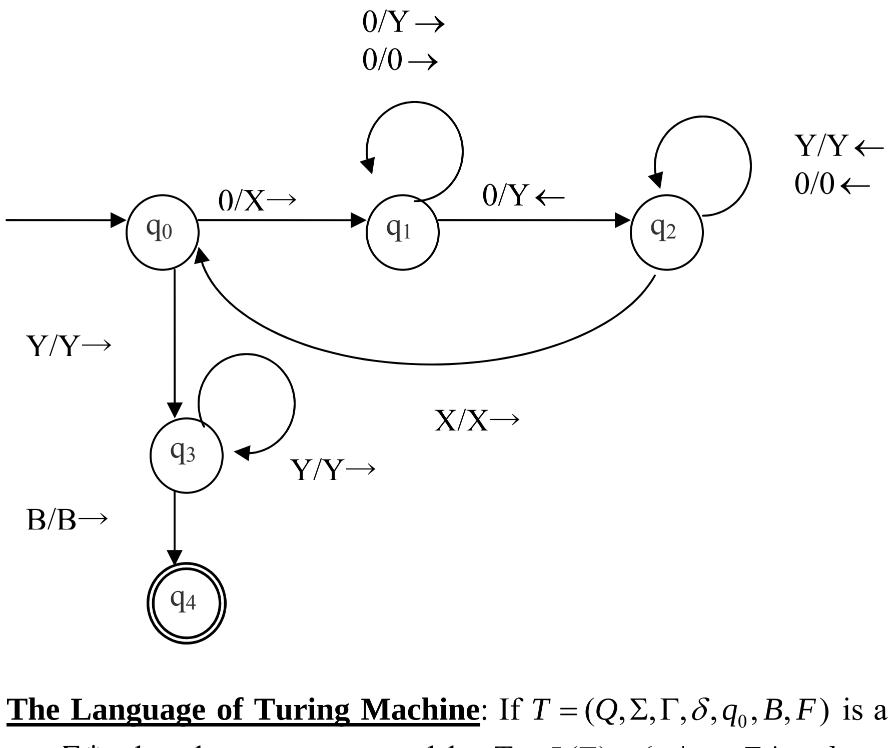
$q_00110 \vdash Xq_2110 \vdash q_2XY10 \vdash Xq_0Y10 \vdash XYq_310$ (Halt and reject)

Since state q_3 has no move on symbol 1 .

Transition diagram : A transition diagram of TM consists of

- Set of nodes representing states of TM.
- An arc from state q to p is labeled by the item(s) of the form X/ YD where X and Y are tape symbol and D is direction. L or R. In diagram direction is represented by L or R(or ←(left arrow) or → right arrow.)

The transition diagram for the above TM is



The Language of Turing Machine: If $T = (Q, \Sigma, \Gamma, \delta, q_0, B, F)$ is a Turing machine and $w \in \Sigma^*$ then language accepted by T , $L(T) = \{w \mid w \in \Sigma^* \textit{ and } q_0w \vdash^* \alpha p \beta\ \}$ for some $p \in F$ and any tape strings α and β.

The set of languages that can be accepted using TM are called recursively enumerable Languages or RE languages.

For more notes visit https://collegenote.pythonanywhere.com

The Turing Machines are designed to perform at least the following three roles.

1. ***As a language recognizer***: TM can be used for accepting a language like Finite Automata and pushdown automata(PDA).

2. ***As a computer of functions***: A Turing machine represents a particular function. Initial input is treated as representing an argument of the function. And the final string on the tape when the TM enters the Halt state treated as representative of the value obtained by an application of the function to the argument represented by the initial string.

3. ***As an enumerator of strings of a Language***, that outputs the strings of a language, one at a time in some systematic order that is as a list.

Turing Machine for computing a Function

A Turing Machine can be used to compute functions. For such TM we adopt the following policy to input any string to a Turing Machine which is a input of the computable function.

The string w is presented in to the form BwB, where B is a blank symbol, and placed on to the tape, the head of Turing Machine is positioned at a blank symbol which immediately follows by the string w.

We can show by underlining that symbol to show current position of machine head in the tape as (q,Bw$\underline{B}$) . Or we can represent it by ID of TM in another format as BwqB. Turing machine is said to halt on input w if we can reach to halting state after performing some operations , that is if

TM = $(Q, \Sigma, \Gamma, \delta, q_0, B, \{q_a\})$ is a Turing Machine them TM is said to halt to on input w iff BwqB Yields to Bα q_{aB} for some string $\alpha \in \Gamma^*$ or equivalently we say

(q,Bw$\underline{B}$) $\vdash$ (q_a, Bα $\underline{B}$).

Definition:

A function $f(x) = y$ is said to be computable by a TM $(Q, \Sigma, \Gamma, \delta, q_0, B, \{q_a\})$ if (q_0 ,Bx$\underline{B}$) $\vdash^*$ (q_a, By$\underline{B}$) where x may be in some alphabet $\Sigma_1{}^*$ and y may be in some alphabet $\Sigma_2{}^*$ and $\Sigma_1, \Sigma_2 \subseteq \Sigma$. It means that if we give input x to the Turing Machine TM , it gives output as a string if it computes the function $f(x) = y$.

For more notes visit https://collegenote.pythonanywhere.com

Example: Design a TM which compute the function f(x) = x +1 for each x that belongs to the set of natural numbers.

Solution: Given function f(x) = x + 1. Here we represent input x on the tape by a number of 1's on the tape.

For example x = 1, input will be B1B,
for x = 2 , input will be B11B,
for x = 3, input will be B111B and so on.

Similarly output can be seen by the number of 1s on the tape when machine halts.

Let TM = $(Q, \sum, \Gamma, \delta, q_0, B, \{q_a\})$ where Q = $\{q_0, q_a\}$ Γ =\{1,B\} and halt state q_a . The transition function is defined as

Q	B	1
q_0	$(q_{0,1,N})$	$(q_{a,1,R})$
q_a	-	-

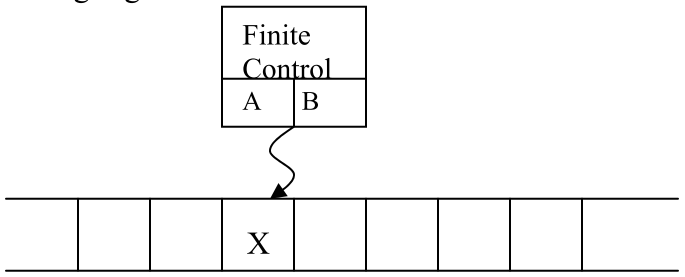
Let input x = 4, that is input tape contains input as B1111B
So $(q_0, B1111\underline{B}) \vdash (q_0, B1111\underline{1}) \vdash (q_a, B11111\underline{B})$ Which means output is 5.

For more notes visit https://collegenote.pythonanywhere.com

Introduction of Turing Machine -2

Storage in the state: In Turing machine, any state represents the position in the computation . But a state can also be used to hold a finite amount of data. The finite control of machine consists of a state q and some data portion. In this case a state is considered as a tuple – (state,data).

Following Figure illustrates the model.

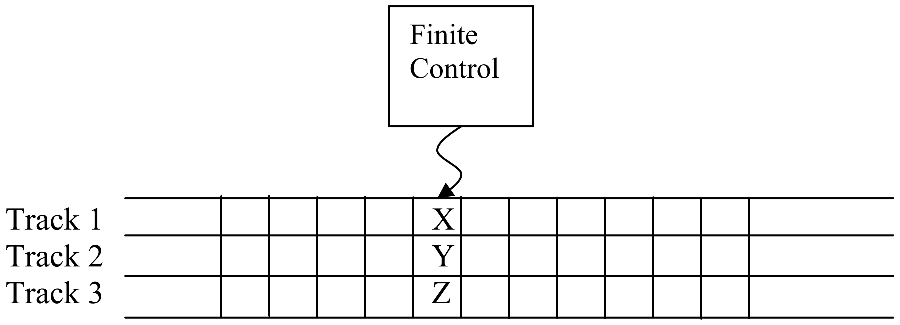


δ is defined by:

$\delta([q, A], X) = ([q_1, X], Y, R)$ means q is the state and data portion of q is A. The symbol scanned on the tape is copied into the second component of the state and moves right entering state q_1 and replacing tape symbol by Y.

Turing Machine with Multiple Tracks:

The tape of Turing machine can be considered as having multiple tracks. Each track can hold one symbol and the tape alphabet of TM consists of tuples with one component for each track. Following figure illustrates multiple track TM.



A TM with a tape with multiple tracks.

Tape alphabet Γ is a set consisting of tuples like

Γ ={(X,Y,Z),……………. }

The tape head moves up and down scanning symbols in the tape at one position.

Subroutine: A Complex Turing machine can be thought as built from a collection of interacting components like a general program. Such components of Turing machine are called subroutines. A Turing Machine subroutine is a set of states that performs some useful processes and can be called in to another machine for the part of that computation.

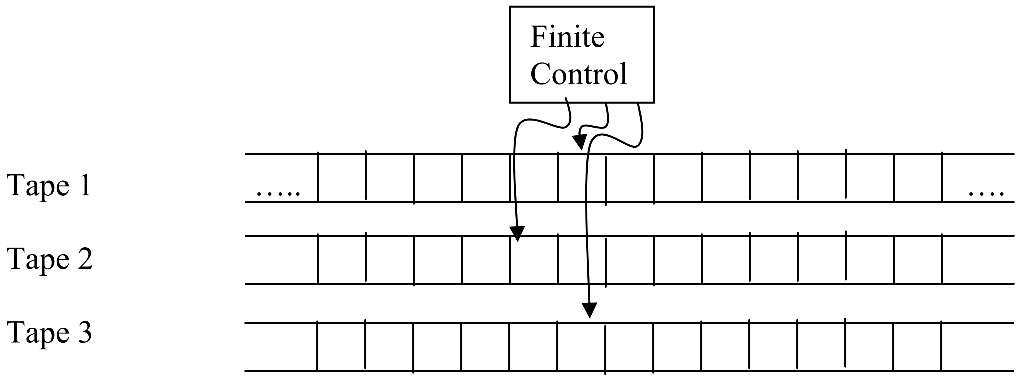
For more notes visit https://collegenote.pythonanywhere.com

Extension of TM:

Modern computing devices are based on the foundation of TM computation models. To simulate the real computers a TM can be viewed as multi-tape machine in which there are more than one tape. Adding extra tape adds no power to the computational model only the ability to accept the language is concerned.

1. Multi-tape Turing Machine:

- A multi-tape Turing machine consists of finite control and finite number of tapes. Each tape is divided into cells and each cell can hold any symbol of finite tape alphabets.
- TM has separate R/W head for each tape.
- The set of tape symbols includes a blank symbol(B) as in single tape machine.
- The tape symbol has a subset called input symbols i.e. we can say $\Sigma \subseteq \Gamma$.



A multi-tape TM

In Multi-tape TM, Initially,

1. Input (Finite sequence of input symbols) *w* is placed on the first tape.
2. All other cell of the tapes hold blanks
3. TM is in the initial state – q_0
4. The head of the first tape is at the left end of the input.
5. All other tape heads are at some arbitrary cell since all other tape except first tape consists completely blank.

A move of multi-tape TM depends on the state and the symbol scanned by each of the tape head. In one move the multi-tape TM does the following.

1. The control enters in a new state, which may be same previous state.
2. On each tape, a new symbol is written on the cell scanned, these symbol may be same as the symbol previously there.
3. Each of the tape head make a move either left or right or remains stationary. Different head may move different direction independently i.e. if head of first tape moves leftward ,at the same time other head can move another directions or remains stationary.

The formal notation of multi-tape TM is the generalization of one-tape TM. The initial configuration(initial ID) of multi-tape TM with n-tapes is represented as:

$(q_0, \underline{a}x, B, B, \ldots\ldots, B)$ – n+1 tuple. Where w = ax is an input string and head of first tape is scanning first symbol of w.

For more notes visit https://collegenote.pythonanywhere.com

The notation of configuration at any instant of TM can be generalized as: (n+1) tuple for n-tape TM.

(q, $x_1\underline{a_1}y_1$, $x_2\underline{a_2}y_2$,……………………, $x_n\underline{a_n}y_n$) Where each x_is are the portion of string on tapes before current head position, each a_is are the symbol currently scanning in each tapes and each y_is are the portion of string in tapes just rightward the current head position. q is the control state

Equivalence of One-tape and Multi-tape Turing Machine

Any recursively enumerable languages that are accepted by one-tape TMs are also accepted by multi-tape TM. Any n-tape TM for $n \geq 2$, are at least as powerful as 1-tape TMs.

Let $n \geq 2$ and $T_1 = (Q_1, \Sigma, \Gamma_1, \delta_1, q_1, B, F_1)$ be an n-tape TM, then there is a 1-tape TM $T_2 = (Q_2, \Sigma, \Gamma_2, \delta_2, q_2, B, F_2)$ with $\Gamma_1 \subseteq \Gamma_2$ which satisfies following conditions.

1. $L(T_1) = L(T_2)$ i.e. for any input $w \in \Sigma^*$, T_2 accepts iff T_1 accepts.
2. For any $w \in \Sigma^*$, if $(q_1, \underline{B}w, B, B, \ldots\ldots, B) \vdash^* (q_a, x_1\underline{a_1}y_1 , x_2\underline{a_2}y_2 , \ldots\ldots, x_n\underline{a_n}y_n)$ for some $a_i \in \Gamma_1$ and $x_i, y_i \in \Gamma_1^*$ then, $(q_2, \underline{B}w) \vdash^* (q_a, y\underline{a}z)$ for some $a \in \Gamma_2$ and $y, z \in \Gamma_2^*$

Simulating Multi-tape Turing Machine with 1-tape Turing Machine

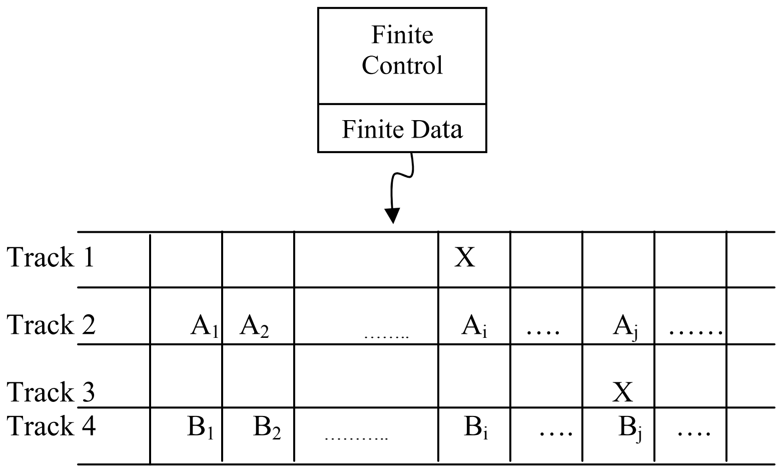
Theorem: Every Language accepted by a Multi-tape TM is recursively enumerable.

Or

Any languages that accepted by a Multi-tape Turing Machine are accepted by one-tape TM.

Proof:

- Let L is a language accepted by a n-tape TM $T_1 = (Q_1, \Sigma, \Gamma_1, \delta_1, q_1, B, F_1)$. We simulate T_1 with one-tape Turing Machine $T_2 = (Q_2, \Sigma, \Gamma_2, \delta_2, q_2, B, F_2)$ considering there are 2n tracks in the tape of T_2.
- Let us assume that n = 2 then for n>2 is generalization of this case.
- Now total no of tracks in T_2 will be 2*2 = 4.The second and fourth tracks of T_2 hold the contents of first and second tapes of T_1 and track 1 holds the head position of tape 1 and track 3 holds the head position of tape 2 of T_1 as in figure below.


Finite Control
Finite Data
Track 1
Track 2
Track 3
Track 4
X
A_1 A_2 …….. A_i …. A_j ……
X
B_1 B_2 ……….. B_i …. B_j ….

For more notes visit https://collegenote.pythonanywhere.com

- To simulate a move of T_1,
  - T_2's head must visit the n-head markers so that it must remember how many head markers are to its left at all times, that count is stored as a component of T_2's finite control.
  - After visiting each head marker and storing the scanned symbol in a component of its finite control, T_2 knows what tape symbols are being scanned by each of T_1's heads.
  - T_2 also knows the state of T_1 and stores its own finite control. i.e. T_2 knows what move T_1 will make.
- T_2 now revisits each of head markers on its tape, changes the symbol in the track representing corresponding tapes of T_1 and moves the head marker left or right.
- Finally , T_2 changes the state of T_1 as recorded in its own finite control. Hence T_2 has simulated one move of T_1.
- We select T_2's accepting states all those states that record T_1's state as one of the accepting state of T_1. Hence whatever T_1 accepts T_2 also accepts.

□

2. **Non-deterministic Turing Machine**:

A non- deterministic TM(NTM) $T = (Q, \Sigma, \Gamma, \delta, q_0, B, F)$ is defined exactly the same as an ordinary TM, except the value of transition function δ . In NTM , the values of the transition function δ are subsets, rather than a single element of the set $Q \times \Gamma \times \{R, L, S\}$

The notation for a TM configuration is unchanged.

- *xqay* $\vdash$ *wpbz* , means , beginning in configuration *xqay* , there is at least one move that will take TM in configuration *wpbz.*
- *xqay* $\vdash^*$ *wpbz* means there is at least one sequence of zero or more moves that take NTM from first configuration to second.
- The addition of non-determinism to Turing machines does not alter the definition of Turing-computable.

Turing Enumerable languages:

A language is enumerable if there is an algorithm for enumerating it. To enumerate a set means to list the elements one at a time. Any languages that can be enumerated by a TM are Turing Enumerable Languages.

Definition: Let T be a k-tape Turing machine($k \geq 1$) and $L \subseteq \Sigma^*$. We say T enumerates L if it operates so that the following conditions are satisfied.

1. The tape head on the first tape never moves to the left and no non-blank symbol printed on tape 1 is subsequently modified or erased.
2. For every $x \in L$, there is some point in the operation of T when Tape 1 has contents
   $x_1Bx_2Bx_3B\ldots\ldots\ldots Bx_nB$
   for some $n \geq 0$, where the strings $x_1, x_2, \ldots\ldots\ldots X_n$ are distinct elements of L. If L is finite, then nothing is printed after the B following the last element of L.

For more notes visit https://collegenote.pythonanywhere.com

Unrestricted Grammars: An unrestricted Grammar (also called phrase-structure Grammar) is a 4-tuple G = (V,T,P,S) where V and T are disjoint sets of variables and terminals respectively, $S \in V$ is start symbol and P is a set of productions of the form $\alpha \rightarrow \beta$

Where $\alpha, \beta \in (V \cup T)^*$ and α contatains at least one variable.

The productions are of the form.

$\alpha A \beta \rightarrow \alpha \gamma \beta$ or $\alpha A \beta \rightarrow \gamma$

An unrestricted Grammar are used to describe the languages that are not context-free. In fact the languages described by unrestricted grammar can be accepted by a TM. Hence they are recursively enumerable.

An unrestricted grammar generating $\{a^n b^n c^n \mid n \geq 1\}$

$S \rightarrow FS_1$
$S_1 \rightarrow ABCS_1$
$S \rightarrow ABC$
$BA \rightarrow AB$
$CA \rightarrow AC$
$CB \rightarrow BC$
$FA \rightarrow a$
$aA \rightarrow aa$
$aB \rightarrow ab$
$bB \rightarrow bb$
$bC \rightarrow bc$
$cC \rightarrow cc$

The string **aabbcc** can be derived as

$S \Rightarrow F\underline{S_1} \Rightarrow FABC\underline{S_1} \Rightarrow FAB\underline{CA}BC \Rightarrow FA\underline{BA}CBC$
$\Rightarrow FAAB\underline{CB}C \Rightarrow \underline{FA}ABBCC \Rightarrow \underline{aA}BBCC$
$\Rightarrow a\underline{aB}BCC \Rightarrow aa\underline{bB}CC \Rightarrow aab\underline{bC}C$
$\Rightarrow aabb\underline{cC} \Rightarrow aabbcc$

TM comuting a Function:

Let T= (Q,∑,$T = (Q, \Sigma, \Gamma, \delta, q_0, B, F)$ be a Turing Machine , let f be a partial function on ∑* with values in Γ^* We say that T computes f if for every $x \in \sum^*$ at which f is defined

$(q_0, \underline{B}x) \vdash^* (q_a, \underline{B}f(x))$

and no other $x \in \sum^*$ is accepted by T.

A partial function f is Turing computable or simply computable if there is a Turing machine computing f.

Partial Recursive Function:

A Partial Recursive Function is one which is allowed to have an infinite loop for some input values.

A Recursive Function is called a Total Recursive Function that always returns a value for all possible input values.

For more notes visit https://collegenote.pythonanywhere.com

Church- Turing Thesis

It is a mathematically un-provable hypothesis abut the computability. This hypothesis simplify states that – "Any algorithmic procedure that can be carried out at all (by human, a team of human or a computer) can be carried out by a TM."

This statement was first formulated by Alonzo Church a logician, in 1930s and it is referred to as Church Thesis or Church-Turing thesis. It is not a mathematically precise statement so un-provable . Later the invention of Turing Machine has accumulated enough evidence to accept this hypothesis. Following are the some of evidences.

1. In nature, a human normally works with 2D paper sheet, not 1D tape. The transfer of attention is not adjacent block like TM. However transferring attention from one place to another during computation can be simulated by Turing Machine as one human step by multiple stapes.
2. Various extensions of Turing Machine Model has been suggested to make computation efficient like doubly infinite tape, a tape with multiple tracks, multi-tape , non-determinism etc. In each case the computational power reserved.
3. Other theoretical model have been suggested that are closer to modern computers in their operation.(e.g. simple programming type language, grammars and others)
4. Since the introduction of the TM , no one has suggested any type of computation that ought to be included in the category of "Algorithmic Procedure".

After adopting Church-Turing Thesis, we are giving a precise meaning of the term:
An algorithm is a procedure that can be executed on a Turing Machine.

Another use of Church-Thesis is that- When we want to describe a solution to a problem, we will often satisfied with a verbal description of the algorithm, translating it into detailed Turing Machine implementations.

For more notes visit https://collegenote.pythonanywhere.com

Universal Turing Machine:

A Turing machine is created to execute a specific algorithm. If we have a Turing machine for computing one function , then for computing different function or doing some other calculation, a different TM will be required. Originally electronic computers were limited in a similar way , and changing the computation to be performed , requires rewriting the machine.

A Turing machine can, simulate a computer that had been loaded with an arbitrary program . I.e. A – TM can be used as a "stored program computer " taking its program as well as data from one or more tapes on which it is placed .

Modern computer are completely flexible in which the task it performs is to execute the instructions stored in its memory , and can represent an algorithm for computation . Turing describes a "Universal Computing Machine " that can be defined as:

Definition : A Universal Turing machine T_u is a TM whose input consist of program and a data set for the program to process. The program takes the from of a string specifying some other (special – purpose) TM T_1 and the data set is second string w interpreted as input to T_1. T_u then simulates the processing of w by T_1.

Encoding of TM:

For the representation of any arbitrary TM T_1, and an input string w over an arbitrary alphabet as strings $e(T_1)$, $e(w)$ over some fixed alphabet , a notational system should be formulated . Encoding the TM T_1 and input string w in to $e(T_1)$ and $e(w)$, it must not destroy any information , for the encoding of TM , we use the alphabet $\{0,1\}$ although the TM may have a much larger alphabet .

For encoding , we start by assigning positive integers to each state , each tape symbol and each of three directions *S,L and R* in the TM we want to encode. We assume two fixed infinite sets Q = { $q_1, q_2,$…………} and S = { $a_1, a_2,$ ………} so that for any TM $T_1 = (Q_1, \Sigma, \Gamma, \delta, q_0, B, F)$, we have $Q_1 \subseteq Q$ and $\Gamma \subseteq S$.

Hence once we have a subscript attached to every possible state and tape symbols, we can represent a state or a symbol by a string of 0's of the appropriate length. 1's are used as separators.

The Encoding Function e

First, associate to each tape symbol, to each state and to each of the three directions , a string of 0's. Let

$s(B) = 0$
$s(a_i) = 0^{i+1}$ for each $a_i \in S$
$s(q_i) = 0^{i+2}$ for each $q_i \in S$
$s(S) = 0$
$s(L) = 00$
$s(R) = 000$

Then each move of TM , described by formula
$\delta(q,a) = (p,b,D)$ is encoded as

$$e(m) = s(q)1s(a)1s(p)1s(b)1s(D)1$$

and for any TM T, with initial state q, T is encoded as:

$$e(T) = s(q)1e(m_1)1e(m_2)1..................1e(m_k)1$$

For more notes visit https://collegenote.pythonanywhere.com

Where $m_1, m_2, ...m_k$ are the distinct moves of T arranged in some arbitrary order.

Finally, any string $w = w_1 w_2 w_3 ....w_k$, for each $w_k \in$ S is encoded as:

$e(w) = 1s(w_1)1s(w_2)1..................1s(w_k)1$

- -The 1 at beginning of *e(w)* included so that a composite string of the form *e(T)e(w)*, there is no doubt for the stoppage of *e(T)* since separated by three 1's. Since no valid code for TM contains three 1's in a row, we can be sure that the first occurrence of 111 separates the code for TM $T_1$and input *w*.
- Also encoding of *s(a)* of a symbol $a \in S$ *is* different from the encoding *e(a)* of the one character string *a*.
- There may be any possible codes for a TM T_1. The codes for the TM for n transitions may be listed in any of n! orders.

<u>An Example: Let TM T is given as:</u>

$T = (\{q_1, q_2, q_3\}, \{a, b\}, \{a, b, B\}, \delta, q_1, B, F)$

where δ is defined by the following moves.

$m_1 = \delta(q_1, b) = (q_3, a, R)$

$m_2 = \delta(q_3, a) = (q_1, b, R)$

$m_3 = \delta(q_3, b) = (q_2, a, R)$

$m_4 = \delta(q_3, B) = (q_3, b, L)$

Encoding :

$s(q_1) = 000$

$s(q_2) = 0000$

$s(q_3) = 00000$

$s(a_1) = 00$ considering $a_1 = a$ *and* $a_2 = b$

$s(a_2) = 000$

$s(B) = 0$

$s(S) = 0$

$s(L) = 00$

$s(R) = 000$

Now $e(m_1) = s(q_1)1s(b)1s(q_3)1s(a)1s(R)1 = 0001000100000100100001$

$e(m_2) = s(q_3)1s(a)1s(q_1)1s(b)1s(R)1 = 00000100100010001000 1$

$e(m_3) = s(q_3)1s(b)1s(q_2)1s(a)1s(R)1 = 000001000100001001000 1$

For more notes visit https://collegenote.pythonanywhere.com

$e(m_4) = s(q_3)1s(B)1s(q_3)1s(b)1s(L)1 = 0000010100000010001001$

Now The code for T will be:

$e(T) = s(q_1)1e(m_1)1e(m_2)1e(m_3)1e(m_4)1$

=00010001000100000100100011000001001000100010001100000100010000010010001100
00010100000100010011
for this machine the input (T,w) where w= ab , the code will be,

e(w) = 1s(a)1s(b)1 = 10010001

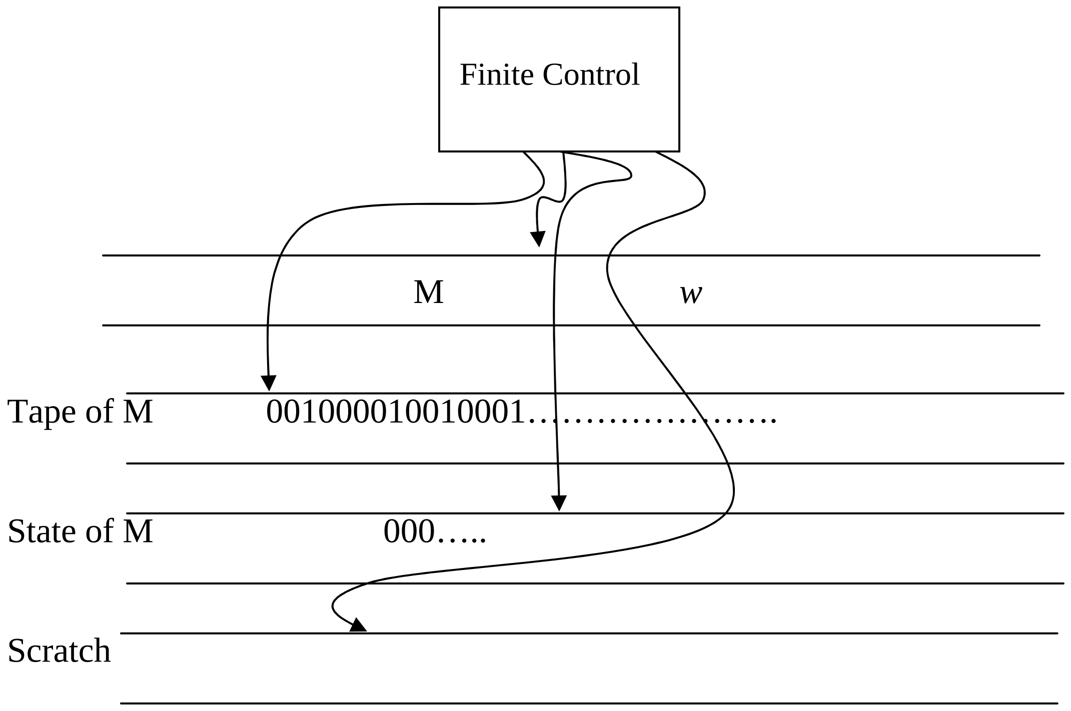
Code for (T,W) is:
000100010001000001001000**11**000001001000100010001**11**00000100010000010010001**1**10000
01010000010001001**11**10010001

Universal Languages:

The universal language L_u is the set of binary strings that encode a pair (*M*,*w*) where M is a TM with binary input alphabet, and w is a string in (0+1)*, such that w is in *L(M)*.

In other words, the languages that is accepted by universal TM is called universal Languages.

A universal TM U can be described as a multi-tape Turing machine in which the transitions of M are stored initially on the first tape along with string w. The second tape holds the simulated tape of M in the format same as the code of M. Third tape of U holds the state of M, with suitable encoding as in figure below.



A Sketch of Universal TM

For more notes visit https://collegenote.pythonanywhere.com

The Operation of Universal Turing Machine U can be described as:

1. Examine the input to make sure that the code for M is valid code for some TM. If not, U halts without accepting. Any invalid codes represents TM with no moves.
2. Initialize the second tape to contain the input *w* in encoded form (simulated tape of M)
3. Place code of q_1 (the start state of M) on the third tape and move the head of U's second tape to the first simulated cell.
4. To simulate a move of M , U searches on its first tape for a transition $0^i10^j10^k10^l10^m1$ such that 0^i is the state on tape 3, 0^j is the tape symbol of M that begins at the position on tape 2 scanned by U. This transition is the one move of M.

   U should:

   a. Change the contents of tape 3 to 0^k, i.e. simulate the state change of M
   b. Replace 0^j on tape 2 by 0^l i.e. change the tape symbol of M. If more or less space is needed($j \neq l$) use scratch tape and shifting over technique as:
      i) Copy onto a scratch tape the entire nonblank tape to the right of where new value goes
      ii) Write the new value using the correct amount of space for that value.
      iii) Recopy the scratch tape onto tape 2 , immediately to the right of new value.
   c. Move head on tape 2 to the position of the next 1 to the left or right or stationary. If m=1 (stationary), if m=2 (move left) and if m=3(move right) .

   Thus U simulates the one move of M.
5. If M has no transition that matches the simulated state and tape symbol, no transition found . So M halts hence U halts likewise.
6. If M enters its accepting state, then U accepts.

<u>The Halting Problem:</u>

The halting problem for a TM M , *H(M*) is defined as the set of input *w* such that M halts given input *w*, regardless of whether or not M accepts *w* . so, the halting problem is the set of pairs (*M, w)* such that w is in *H(M).*

The halting problem is related to the membership problem of RE languages . For a given TM M and given string *w*, instead of asking M accepts *w*, it asks whether M halts on input *w*. We abbreviate the problem Halts as :

Halts : Given a TM M and a string *w*, does M halt on input *w* ?

The domain of this problem is to be taken as the set of all Turing machines and all w; that is, we are looking for a single Turing machine that , given the description of and arbitrary TM and w, will predict whether or not the computation of TM applied to w halt. We can not find the answer by simulating the action of TM on w, say by performing it on universal Turing machine, because there is no limit on the length of computation. If TM enters an infinite loop, then no matter how long we wait, we can never be sure that TM is in fact in a loop. It may be simple case of very long computation .