

UNIT-5

Types of Turing Machines

TM can also be deterministic or non deterministic but this doesn't make them any more or less powerful.

→ But if the tape is restricted so that you can only see use of the part of the tape with the input the TM becomes less powerful and can only recognise context sensitive language.

Other turing machines variations are equivalent to the original TM.

- Multi track
- Multi tape
- Multi head
- Multi dimensional tape.
- The off-line turing machine.

Multi tape Turing Machine

A turing machine with several tapes we call it a multi tape T.M

Every tapes have their own Read/Write head.

For N -tape Turing machine

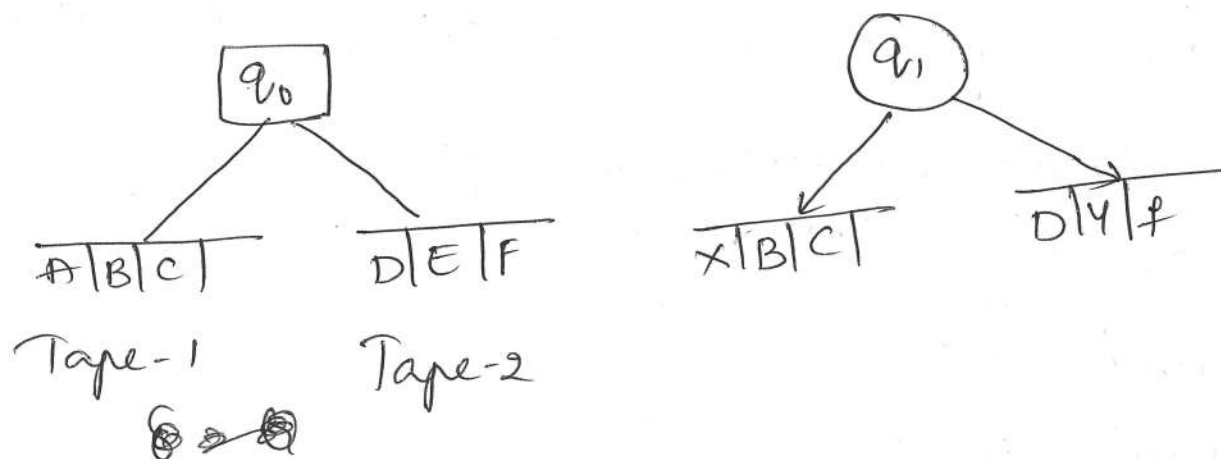
$$M = \{ (Q, \Sigma, \delta, q_0, B, F) \}.$$

We define $\delta = Q \times \Sigma^n \rightarrow Q \times \Sigma^n \times \{L, R\}^N$

Example

if $n=2$ with current configuration

$$\delta(q_0, a, e) = (q_1, x, y, L, R).$$



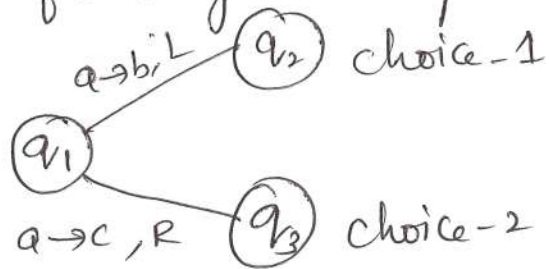
Non Deterministic TM

It is similar to DTM except that for any input and current state it has a no. of choices.

A string is accepted by NDTM if there is a sequence of moves that leads to a final state.

The transition function = $Q \times X \rightarrow 2^{Q \times X \times \{L, R\}}$

A NDTM is allowed to have more than one transition for a given tape symbol



Multi-head Turing machine

- It has a no. of heads instead of one
 Each head independently reads/writes symbol
 and moves left/right or keeps stationary

Off-line Turing Machine:

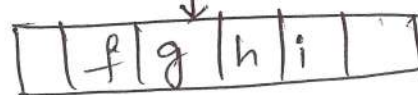
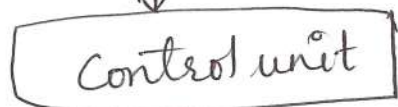
An offline Turing machine has two tapes which are as follows

- One tape is read only and contains the input.

- The other is read write and is initially blank.



Read only input
file's tape.



write/read tape

Turing Machine & Halting.

Halting problem is undecidability. It is not a problem. It just asks question.

"Is it possible to tell whether a given m/c will halt for some given i/p."

Eg:- Input: A T.M & i/p string w .
problem: Does the T.M finish computing of the string w in a finite no. of steps.

proof:- Assume T.M exists to solve this problem & then we will show it is contradicting itself.

Let us call TM as a halting m/c that produces

Yes or no in a finite amount of time.

- If the halting m/c finishes in a finite amount of time the o/p comes as 'yes' otherwise as 'no'.

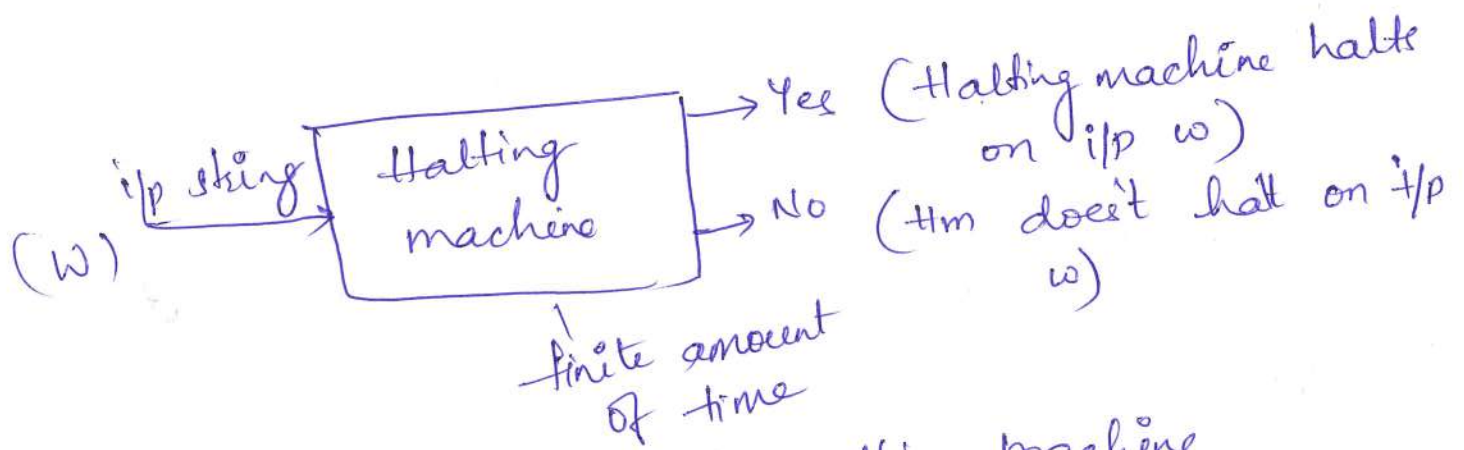


Fig:- Block diagram for Halting machine

Now we will design an inverted halting machine as:-

- If H/m returns Yes, then loop forever
- If " " " No, " Halt.

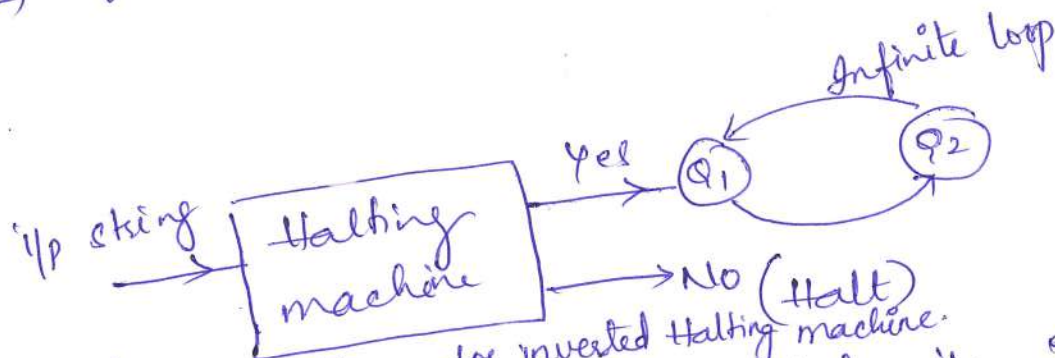


Fig: Block diagram for inverted Halting machine.

- After that a m/c $(H_M)_2$ which i/p itself constructed as

- If $(H_M)_2$ halts on i/p, loop forever.
- else halt

after this we got a contradiction then the halting problem is undecidable.

Undecidability

The class of problems which can be answered as 'yes' are called solvable or decidable. otherwise the class of problems is said to be unsolvable or undecidability.

already done { Undecidable problem about Turing machine }
The reduction is used to prove whether given language is decidable or not.

Decidable language :-

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

Partially Decidable languages:

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:-

A language 'L' is said to be recursive if there exists a T.M 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 an answer (accepted or rejected) for each and every string input.

Recursive Enumerable languages:-

A language 'L' is said to be a recursively enumerable language if there exists a T.M which will accept (and therefore halt) for all the input strings which are in 'L'.

But may or may not halt for all the strings which are not in 'L'.

Recursive language	TM will always Halt
Recursive Enumerable lang	TM will halt sometimes & may not halt sometimes
Decidable languages	Recursive language
partially Decidable lang	Recursively Enumerable language
Undecidable	No TM for that language.

Undecidability of the Halting problem.

Can we design a machine which if given a program can find out or decide if that program will always halt or not on a particular input?

$H(P, I)$

$\begin{cases} \text{Halt} \\ \text{not Halt} \end{cases}$

This allows us to write another program

$C(x)$

if $(H(x, x) == \text{Halt})$

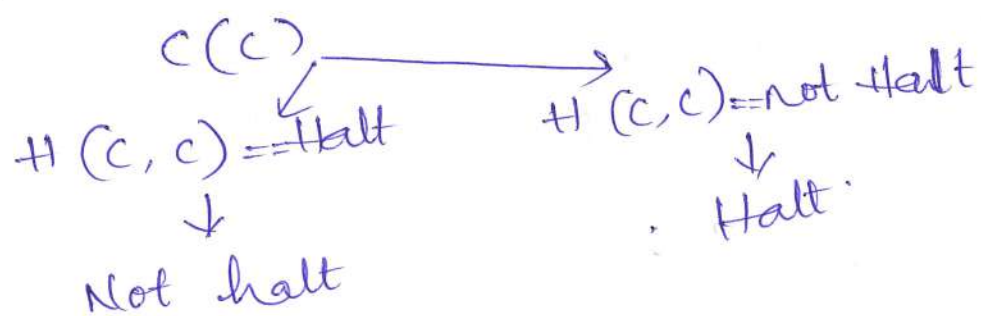
loop forever;

else

return;

5-5

if we run 'c' on itself:



Contradiction

i.e we can't have a m/c which will tell that program will halt or not. This proves halting problem is undecidable

The Post Correspondence Problem

	Reg	DCFL	CFL	CSC	Recursive	Recursive Enumerable
Membership problem $w \in \Sigma^*$	✓ (FA)	✓	✓ (CYK)	✓	✓	X
Infiniteness problem $L = \text{finite or infinite}$	✓	✓	✓	X	X	X
Emptiness problem $L = \emptyset$	✓	✓	✓	X	X	X
Equality problem $L_1 = L_2$	✓	✓	X	X	X	X
L is ambiguous?	✓	✓	X	X	X	X
Completeness $L = \Sigma^*$	✓	✓	X	X	X	X
$L_1 \cap L_2 = \emptyset$	✓	X	X	X	X	X
if $L_1 \leq L_2$ subset problem	✓	X	X	X	X	X

✓ $\rightarrow$ Decidable

X $\rightarrow$ undecidable.

A language that is not recursively Enumerable.

"Recursive Enumerable language"

→ L is Recursive Enumerable if there is TM.

(01)

Language accepted by a Turing machine is recursively Enumerable.

Three states : Halt and Accept
Halt and Reject
Never halt / loop



- closed under all except → Complement.

"Recursive language"

- L is recursive if there is halting / Total Turing machine

→ Two state: Halt and Accept
Halt and Reject

→ closed under all except Homomorphism and substitution

Undecidable problem about Turing machine

The reduction is used to prove whether given language is decidable or not.

Reduction: Is a technique in which if a problem P_1 is reduced to a problem P_2 then any solution of P_2 solves P_1 .

Example: If we have an algorithm to convert an instance of a problem P_1 to an instance of a problem P_2 that have the same answer then it is called as P_1 reduced P_2 . Hence if P_1 is not recursive then P_2 is also not recursive. Similarly if P_1 is not recursively enumerable then P_2 is also not recursively enumerable.

Properties of Recursive languages.

1. Union - union of 2 recursive language is recursive.

If L_1 and L_2 are two recursive languages their union $L_1 \cup L_2$ will also be recursive language 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 L_2 are two recursive languages, their concatenation $L_1 \cdot L_2$ will also be recursive.

ex: $L_1 = \{a^n b^n c^n \mid n \geq 0\}$

$$L_2 = \{d^m e^m f^m \mid m \geq 0\}$$

$$L_3 = L_1 \cdot L_2$$

$$= \{a^n b^n c^n d^m e^m f^m \mid m \geq 0 \text{ and } n \geq 0\} \text{ is also recursive.}$$

Kleen closure :- If L_1 is recursive, its kleen closure L_1^* will also be recursive

$$L_1 = \{a^n b^n c^n \mid n \geq 0\}$$

$$L_1^* = \{a^n b^n c^n \mid n \geq 0\}^* \text{ is also recursive.}$$

Intersection : If L_1 and L_2 are two recursive languages, their intersection $L_1 \cap L_2$ will also be recursive.

$$L_1 = \{a^n b^n c^n \mid n \geq 0\}$$

$$L_2 = \{a^m b^m c^m \mid m \geq 0\}$$

$$L_3 = L_1 \cap L_2$$

$$= \{a^n b^n c^n \mid n \geq 0\} \text{ will also be recursive}$$

Complement : If L_1 is recursive then the complement of $\Sigma^* - L_1$ will also be recursive.

Post Correspondence problem (PCP) - Introduced by Emil post in 1946, is an undecidable decision problem. The PCP problem over an alphabet Σ is stated as follows -

Given the following two lists M and N of non empty strings over Σ -

$$M = (x_1, x_2, x_3, \dots, x_n)$$

$$N = (y_1, y_2, y_3, \dots, y_n)$$

we can say that there is a post correspondence solution. If for some $i_1, i_2, \dots, i_k$ where $1 \leq i_j \leq n$ the condition $x_{i_1} \dots x_{i_k} = y_{i_1} \dots y_{i_k}$ satisfies.

Example's Find whether the lists:

$M = (abb, aa, aaa)$ and $N = (bba, aaa, aa)$ have a post correspondence solution?

	x_1	x_2	x_3
M	abb	aa	aaa
N	bba	aaa	aa

Here $x_2 x_1 x_3 = 'aaabbaaa'$

and $y_2 y_1 y_3 = 'aaabbaaa'$

we can see that

$$x_2 x_1 x_3 = y_2 y_1 y_3$$

Hence the solution is $i=2, j=1$ and $k=3$
 Find whether the set $M = (ab, bab, bbaaa)$
 and $N = (a, ba, bab)$ have a post correspondence solution?

	x_1	x_2	x_3
M	ab	bab	bbaaa
N	a	ba	bab

In this case there is no solution because

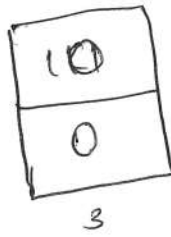
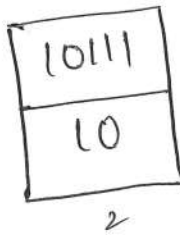
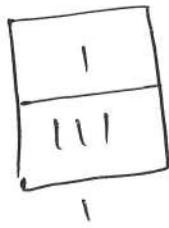
$$|x_2 x_1 x_3| \neq |y_2 y_1 y_3| \quad (\text{lengths are not same})$$

Hence it can be said that this post correspondence problem is undecidable.

Domino ^{files}

Representation

: Domino can be used n no. of times.



- arrange tiles in order that string made by numerators is same as string made by Denominator

Step: use a domino whose first symbol is same.

A : ^② 10111 ^① 1 ^① 1 ^③ 10
 B : 10 111 111 0

A = B

Modified Post Correspondence Problem

(ATM)

We know that acceptance problem ^{is} undecidable. and so PCP problem is also undecidable

Modified PCP : The modified PCP has an additional requirement than PCP, first pair in the solution is the first pair in the list. Remaining sequence in this solution can be in any order.

The MPCP problem is to determine whether there exists a sequence of one or more integers $i_1, i_2, \dots, i_m$ such that

$$w_{i_1} w_{i_2} \dots w_{i_m} = x_1 x_2 \dots x_m.$$

↓

first element
of list A

w_1
 w_2
 $\vdots$
 w_m

↓

first element
of list B

x_1
 x_2
 $\vdots$
 x_m

Example :-

list A

1	10
2	11
3	110

list B

10
011
11

modified PCP has solution 1, 3, 2

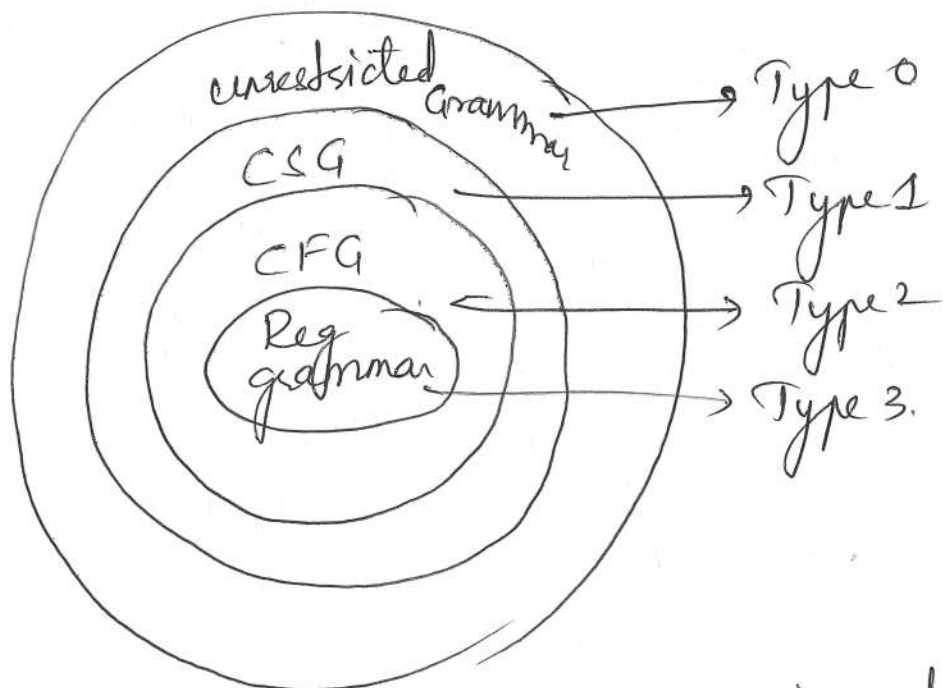
$w_1 w_3 w_2$
10 110 11

$x_1 x_3 x_2$
10 110 11

Chomsky Hierarchy.

→ Represents the class of languages that are accepted by the different machines. The category of language in Chomsky's Hierarchy is as follows.

1. Type 0 → known as unrestricted Grammar
2. Type 1 → " " Context Sensitive "
3. Type 2 → " " Context free Grammar
4. Type 3 → Regular Grammar



Here every language of Type 3 is also Type 2, 1 and 0, similarly Type 2 is also of Type 1 and 0, ...

Type 0 Grammar — Unrestricted Grammar

There is no restriction on the grammar rules of these types of languages. These languages can be efficiently modeled by Turing machines.

$$\text{Eg: } bAa \rightarrow aa$$

$$S \rightarrow S$$

Type 1 Grammar — Context Sensitive Grammar

It is used to represent CSL and follows rules such as:

- The CSG may have more than one symbol on the L.H.S of their production rule.
- The no. of symbols on the L.H.S must not exceed the no. of symbols on R.H.S
- The rule of the form $A \rightarrow \epsilon$ is not allowed unless A is a start symbol. It doesn't occur on the R.H.S of any rule.
- The type 1 grammar should be Type 0. In type 1, production is in the form of $V \rightarrow T$ where the count of symbol $V \leq T$

Ex: - $S \rightarrow AT$
 $T \rightarrow XY$
 $A \rightarrow a$

Type 2 Grammar - The CFG

CFL are the languages which can be represented by CFG. Type 2 should be type 4. The production rules is of the form

$$A \rightarrow \alpha$$

where A is any single non-terminal and α is any combination of terminals & nonterminals.

Example $A \rightarrow aBb$
 $A \rightarrow b$
 $B \rightarrow a$

Type 3 Grammar :- Regular Grammar

Regular languages are those languages which can be described using regular expressions. These languages can be modelled by NFA or DFA.

Type 3 is most restricted form of grammar

The type 3 grammar should be type 2 & type 1.

It should be in the form of

$$V \rightarrow T^*V / T^*$$

$$A \rightarrow XY$$

other

Few Important Examples of undecidable problems

1, whether CFG generates all the strings or not

- As a CFG generates infinite strings, we can't ever reach up to the last string and hence it is undecidable.

2, Whether two CFGs are equal?

Since we cannot determine all the strings of any CFG, we can't predict that two CFG are equal or not

3, Is a language learning which is a CFL regular?

This is an undecidable problem as we can not find when the production rules of the CFL whether it is regular or not

Other undecidable problems related to Turing machine are:

- Membership problem of Turing machine
- Finiteness of a Turing machine
- Emptiness " " " "

Counter Machines

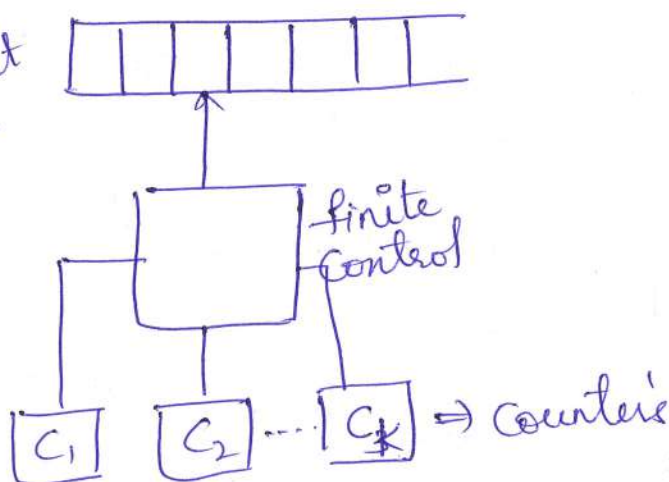
A counter machine can store any non negative integer

Operations of CM

- check if a counter contains zero
- Increment counter value by 1
- Decrement " value by 1

Counter Machines

Read
only input
tape



" tape can only move
towards right.

A counter can store any non negative integer

Operations allowed on counter are

- check if a counter contains zero value.

- Increment counter value by one

- Decrement " " " one provided
counter value before decrement is positive.

Counter machine depends on present state,
input symbol being read

Example.

$c_1 \quad c_2 \quad c_3$

$(q, a, \text{positive}, \text{zero}, \text{positive}) \Rightarrow (\overset{\text{next state}}{p}, \underset{c_1}{\text{inc}}, \underset{c_2}{\text{inc}}, \underset{c_3}{\text{dec}})$

$\Rightarrow$ counter machine is more restricted i.e

A counter can be seen as a restricted version of a stack

→ if top element is $\$$ then counter is zero.

→ push symbol into stack if we increment counter

→ If we see top of element is x then we can pop.

→ A one counter is restricted pushdown automata i.e. Deterministic PDA.

Consider if we have three counters.

3 counter machines can simulate a T.M

A counter machine with 3 counters can simulate 2 stacks

We represent each of the two stacks by a counter

C_1 represents Stack 1

C_2 " Stack 2.

