The Halting problem

Turing machine –

A Turing machine is a mathematical model of computation.

A Turing machine is a general example of a CPU that controls all data manipulation done by a computer.

Turing machine can be halting as well as non halting and it depends on algorithm and input associated with the algorithm.

The Halting problem – Given a program/algorithm will ever halt or not?

Halting means that the program on certain input will accept it and halt or reject it and halt and it would never go into an infinite loop.

Basically halting means terminating. So can we have an algorithm that will tell that the given program will halt or not.

In terms of Turing machine, will it terminate when run on some machine with some particular given input string.

The answer is no

we cannot design a generalized algorithm which can appropriately say that given a program will ever halt or not?

The only way is to run the program and check whether it halts or not.

We can refrain the halting problem question in such a way also: Given a program written in some programming language(c/c++/java) will it ever get into an infinite loop(loop never stops) or will it always terminate(halt)?

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.

The Halting Problem tells that it is not easy to write a computer program that executes in the limited time that is capable of deciding whether a program halts for an input.

An example of writing the Halting Problem is as follows −

INPUT − Program P and a string S.

OUTPUT − if P stops on S, it returns 1.

Otherwise, if P enters into an endless loop on S, it returns 0.

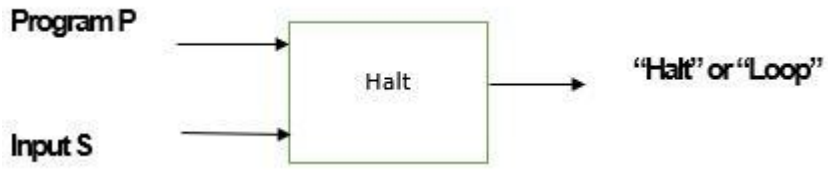
Let us consider the Halting Problem called H having the solution.

Now H takes the following two inputs −

- Program P
- Input S.

If P stops on S, then H results in “halt”, otherwise H gives the result “loop”.

The **diagrammatic representation of H** is as follows −



Example

ATM = {(M,w) | M is a TM and M halts at input w }.

We can build a universal Turing machine which can simulate any Turing machine on any input.

Let's consider TM which recognizing the Altering Turing Machine (ATM) –

```
Recognize-ATM (<M,w>)
  Simulate M using UTM till it halts
  If M halts and accept then
    Accept
  Else
    Reject
```

Suppose, if M goes into an infinite loop on input w, then the TM Recognize-ATM is going to run forever which means TM is only a recognizer, not a decider.

A decider for this problem would call a halt to simulations that loop forever.

Now the question is whether an ATM is TM decidable is equivalent to asking the question whether we can tell if a TM M will halt on input w.

Because of this, both versions of this question are generally called the halting problem.

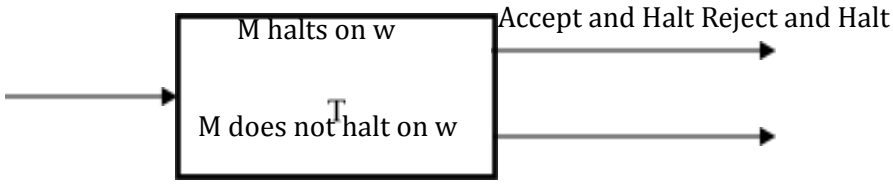
Halting problem for Turing machine

Halting problem of Turing machines over ∑ = {a, b} is i.e. the problem of determining whether or not an arbitrary Turing machine M over alphabet ∑ = {a, b} Halts for any arbitrary string *w* over ∑ is unsolvable.

Theorem: The halting problem is undecidable.

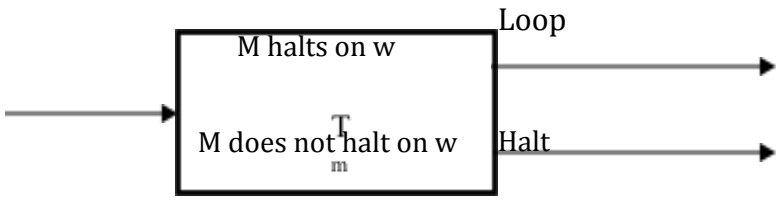
Proof: This is going to be proven by "proof by contradiction".

Suppose that the halting problem is decidable. Then there is a Turing machine T solves the halting problem. That is, given a description of a Turing machine M (over the alphabet) and a string w, T writes "yes" if M halts on w and "no" if M does not halt on w, and then T halts.



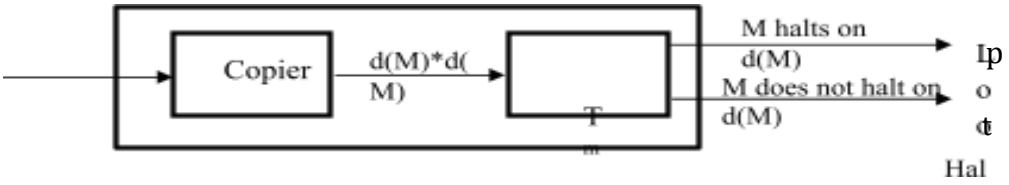
Turing Machine T

We are now going to construct the following new Turing machine T_c. First we construct a Turing machine T_m by modifying T so that if T accepts a string and halts, then T_m goes into an infinite loop (T_m halts if the original T rejects a string and halts).



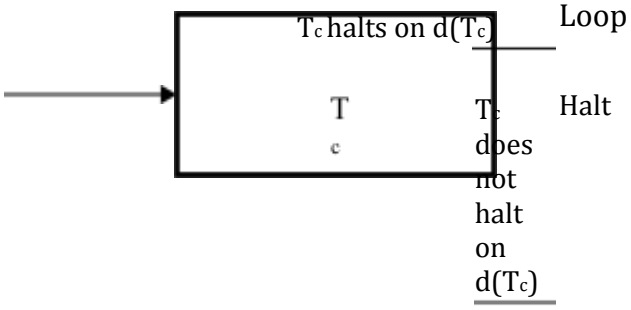
Turing Machine T_m

Next using T_m we are going to construct another Turing machine T_c as follows: T_c takes as input a description of a Turing machine M, denoted by d(M), copies it to obtain the string d(M)*d(M), where * is a symbol that separates the two copies of d(M) and then supplies d(M)*d(M) to the Turing machine T_m.



Turing Machine T_c

Let us now see what T_c does when a string describing T_c itself is given to it. When T_c gets the input $d(T_c)$, it makes a copy, constructs the string $d(T_c)*d(T_c)$ and gives it to the modified T. Thus the modified T is given a description of Turing machine T_c and the string $d(T_c)$.



Turing Machine T_c on input $d(T_c)$

The way T was modified the modified T is going to go into an infinite loop if T_c halts on $d(T_c)$ and halts if T_c does not halt on $d(T_c)$.

Thus T_c goes into an infinite loop if T_c halts on $d(T_c)$ and it halts if T_c does not halt on $d(T_c)$. This is a contradiction. This contradiction has been deduced from our assumption that

there is a Turing machine that solves the halting problem. Hence that assumption must be wrong. Hence there is no Turing machine that solves the halting problem.