UNIT-5

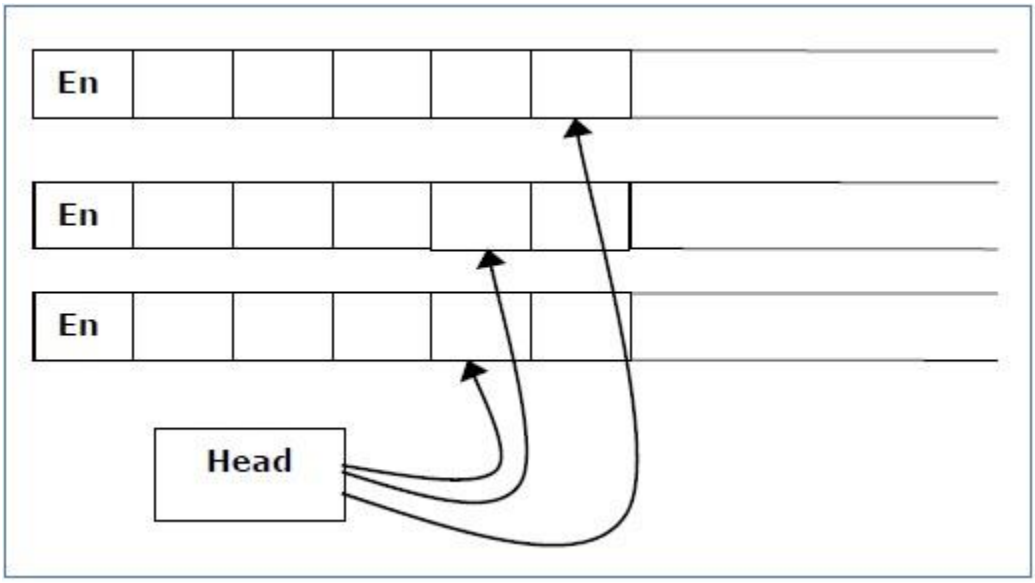
Types of Turing machine: Turing machines and halting

Undecidability: Undecidability, A Language that is Not Recursively Enumerable, An Undecidable Problem That is RE, Undecidable Problems about Turing Machines, Recursive languages, Properties of recursive languages, Post's Correspondence Problem, Modified Post Correspondence problem, Other Undecidable Problems, Counter machines.

Types of Turing Machines :

- Multi-Tape/Head Turing Machine
- Multi-track Turing Machine
- Nondeterministic Turing Machine

Multi-Tape/Head TM: 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, X, B, δ, q0, F) where −

- Q is a finite set of states
- X is the tape alphabet
- B is the blank symbol
- δ is a relation on states and symbols where δ: Q × Xk → Q × (X × {Left_shift, Right_shift, No_shift })k where there is k number of tapes

- q0 is the initial state
- F is the set of final states

Note − Every Multi-tape Turing machine has an equivalent single-tape Turing machine.

Multi-track Turing machine:

Multi-track Turing machines, a specific type of Multi-tape Turing machine, contain multiple tracks but just one tape head reads and writes on all tracks. Here, a single tape head reads n symbols from n tracks at one step. It accepts recursively enumerable languages like a normal single-track single-tape Turing Machine accepts.

A Multi-track Turing machine can be formally described as a 6-tuple (Q, X, ∑, δ, q0, F) where –

- Q is a finite set of states
- X is the tape alphabet
- ∑ is the input alphabet
- δ is a relation on states and symbols where δ(Qi, [a1, a2, a3,....]) = (Qj, [b1, b2, b3,....], Left_shift or Right_shift)
- q0 is the initial state
- F is the set of final states

Note − For every single-track Turing Machine S, there is an equivalent multi-track Turing Machine M such that L(S) = L(M).

Non-Deterministic Turing Machine:

In a Non-Deterministic Turing Machine, for every state and symbol, there are a group of actions the TM can have. So, here the transitions are not deterministic. The computation of a non-deterministic Turing Machine is a tree of configurations that can be reached from the start configuration.

An input is accepted if there is at least one node of the tree which is an accept configuration, otherwise it is not accepted. If all branches of the computational tree halt on all inputs, the non-deterministic Turing Machine is called a Decider and if for some input, all branches are rejected, the input is also rejected.

A non-deterministic Turing machine can be formally defined as a 6-tuple (Q, X, ∑, δ, q0, B, F) where −

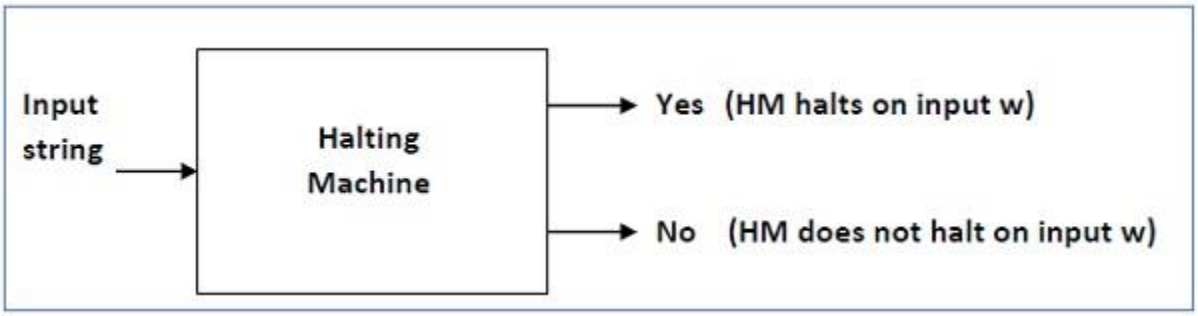
- Q is a finite set of states
- X is the tape alphabet
- ∑ is the input alphabet
- δ is a transition function; $\delta : Q \times X \rightarrow P(Q \times X \times \{Left\_shift, Right\_shift\})$.
- q0 is the initial state
- B is the blank symbol
- F is the set of final states

Turing Machine Halting Problem:

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.

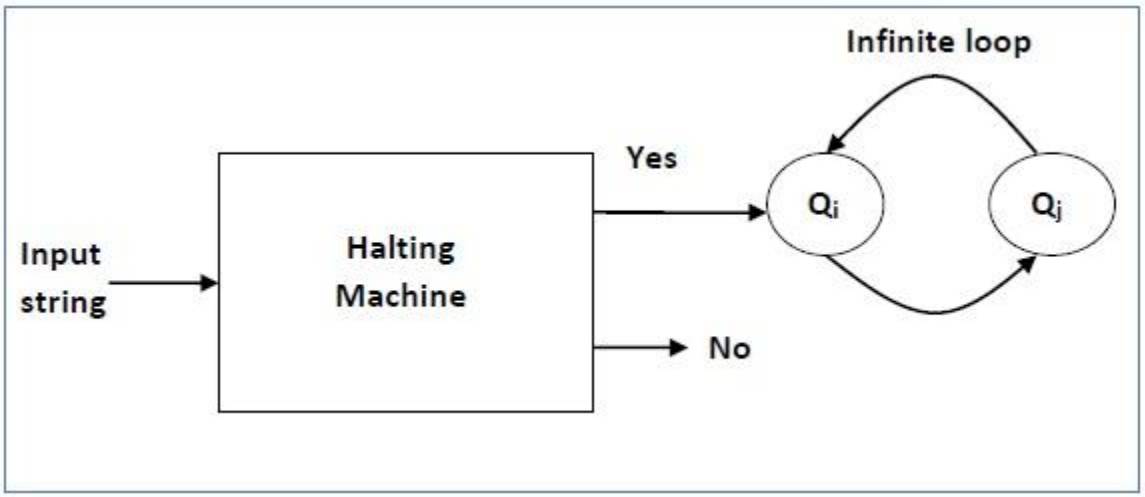
Proof − At first, we will assume that such a Turing machine exists to solve this problem and then we will show it is contradicting itself. We will call this Turing machine as a Halting machine that produces a 'yes' or 'no' in a finite amount of time. If the halting machine finishes in a finite amount of time, the output comes as 'yes', otherwise as 'no'. The following is the block diagram of a Halting machine −



Now we will design an inverted halting machine (HM)' as −

- If H returns YES, then loop forever.
- If H returns NO, then halt.

The following is the block diagram of an 'Inverted halting machine' −



Further, a machine (HM)2 which input itself is constructed as follows −

- If (HM)2 halts on input, loop forever.
- Else, halt.

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

Undecidable Problems

A problem is undecidable if there is no Turing machine which will always halt in finite amount of time to give answer as 'yes' or 'no'. An undecidable problem has no algorithm to determine the answer for a given input.

Examples

- Ambiguity of context-free languages: Given a context-free language, there is no Turing machine which will always halt in finite amount of time and give answer whether language is ambiguous or not.
- Equivalence of two context-free languages: Given two context-free languages, there is no Turing machine which will always halt in finite amount of time and give answer whether two context free languages are equal or not.
- Everything or completeness of CFG: Given a CFG and input alphabet, whether CFG will generate all possible strings of input alphabet (∑*)is undecidable.
- Regularity of CFL, CSL, REC and REC: Given a CFL, CSL, REC or REC, determining whether this language is regular is undecidable.

Note: Two popular undecidable problems are halting problem of TM and PCP (Post Correspondence Problem).

Undecidable Problems about Turing machine:

- Membership problem of a Turing Machine?
- Finiteness of a Turing Machine?
- Emptiness of a Turing Machine?
- Whether the language accepted by Turing Machine is regular or CFL?

Recursive Enumerable (RE) or Type -0 Language

RE languages or type-0 languages are generated by type-0 grammars. An RE language can be accepted or recognized by Turing machine which means it will enter into final state for the strings of language and may or may not enter into rejecting state for the strings which are not part of the language. It means TM can loop forever for the strings which are not a part of the language. RE languages are also called as Turing recognizable languages.

Recursive Language (REC)

A recursive language (subset of RE) can be decided by Turing machine which means it will enter into final state for the strings of language and rejecting state for the strings which are not part of the language. e.g.; L= {anbncn|n>=1} is recursive because we can construct a turing machine

which will move to final state if the string is of the form anbncn else move to non-final state. So the TM will always halt in this case. REC languages are also called as Turing decidable languages. The relationship between RE and REC languages can be shown in Figure 1.

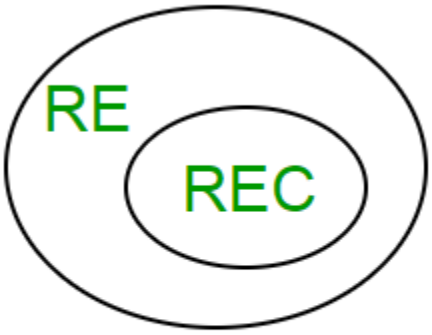


Figure 1

Closure Properties of Recursive Languages:

- **Union:** If L1 and If L2 are two recursive languages, their union L1∪L2 will also be recursive because if TM halts for L1 and halts for L2, it will also halt for L1∪L2.
- **Concatenation:** If L1 and If L2 are two recursive languages, their concatenation L1.L2 will also be recursive. For Example:
  L1= {anbncn|n>=0}
  L2= {dmemfm|m>=0}
  L3= L1.L2 = {anbncndm emfm|m>=0 and n>=0} is also recursive.
  L1 says n no. of a's followed by n no. of b's followed by n no. of c's. L2 says m no. of d's followed by m no. of e's followed by m no. of f's. Their concatenation first matches no. of a's, b's and c's and then matches no. of d's, e's and f's. So it can be decided by TM.
- **Kleene Closure:** If L1is recursive, its kleene closure L1* will also be recursive. For Example:
  L1= {anbncn|n>=0}
  L1*= { anbncn||n>=0}* is also recursive.
- **Intersection and complement:** If L1 and If L2 are two recursive languages, their intersection L1 ∩ L2 will also be recursive. For Example:
  L1= {anbncndm|n>=0 and m>=0}
  L2= {anbncndn|n>=0 and m>=0}
  L3=L1 ∩ L2 = { anbncndn |n>=0} will be recursive.
  L1 says n no. of a's followed by n no. of b's followed by n no. of c's and then any no. of d's. L2 says any no. of a's followed by n no. of b's followed by n no. of c's followed by n no. of d's. Their intersection says n no. of a's followed by n no. of b's followed by n no. of c's followed by n no. of d's. So it can be decided by turing machine, hence recursive.

Similarly, complement of recursive language L1 which is ∑*-L1, will also be recursive.

Note: As opposed to REC languages, RE languages are not closed under complement which means complement of RE language need not be RE.

. . .

Post Correspondence Problem:

Post Correspondence Problem is a popular undecidable problem that was introduced by Emil Leon Post in 1946. It is simpler than Halting Problem.

In this problem we have N number of Dominos (tiles). The aim is to arrange tiles in such order that string made by Numerators is same as string made by Denominators.

In simple words, lets assume we have two lists both containing N words, aim is to find out concatenation of these words in some sequence such that both lists yield same result.

Let's try understanding this by taking two lists A and B

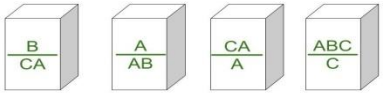
A=[aa, bb, abb] and B=[aab, ba, b]

Now for sequence 1, 2, 1, 3 first list will yield aabbaaabb and second list will yield same string aabbaaabb.

So the solution to this PCP becomes 1, 2, 1, 3.

Post Correspondence Problems can be represented in two ways:

1. Domino's Form :



2. Table Form :

	Numerator	Denominator
1	B	CA
2	A	AB
3	CA	A
4	ABC	C

Lets consider following examples.

Example-1:

	A	B
1	1	111
2	10111	10
3	10	0

Explanation –

- Step-1:

  We will start with tile in which numerator and denominator are starting with same number, so we can start with either 1 or 2.
  Lets go with second tile, string made by numerator- 10111, string made by denominator is 10.

- Step-2:

  We need 1s in denominator to match 1s in numerator so we will go with first tile, string made by numerator is 10111 1, string made by denominator is 10 111.

- Step-3:

  There is extra 1 in numerator to match this 1 we will add first tile in sequence, string made by numerator is now 10111 1 1, string made by denominator is 10 111 111.

    - Step-4:

  Now there is extra 1 in denominator to match it we will add third tile, string made by numerator is 10111 1 1 10, string made by denominator is 10 111 111 0.
  Final Solution - 2 1 1 3

    - String made by numerators: 101111110

- String made by denominators: 101111110

  As you can see, strings are the same.

Example-2:

	A	B
1	100	1
2	0	100
3	1	00

Explanation –

- Step-1:

  We will start from tile 1 as it is our only option, string made by numerator is 100, string made by denominator is 1.

- Step-2:

  We have extra 00 in numerator, to balance this only way is to add tile 3 to sequence, string made by numerator is 100 1, string made by denominator is 1 00.

- Step-3:

  There is extra 1 in numerator to balance we can either add tile 1 or tile 2. Lets try adding tile 1 first, string made by numerator is 100 1 100, string made by denominator is 1 00 1.

- Step-4:

  There is extra 100 in numerator, to balance this we can add 1st tile again, string made by numerator is 100 1 100 100, string made by denominator is 1 00 1 1 1. The 6th digit in numerator string is 0 which is different from 6th digit in string made by denominator which is 1.

We can try unlimited combinations like one above but none of combination will lead us to solution, thus this problem does not have solution.

Other Undecidable Problems:

- Whether a 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.
- Whether two CFG L and M equal?

  Since we cannot determine all the strings of any CFG, we can predict that two CFG are equal or not.
- Ambiguity of CFG?

  There exist no algorithm which can check whether for the ambiguity of a CFL. We can only check if any particular string of the CFL generates two different parse trees then the CFL is ambiguous.
- Is it possible to convert a given ambiguous CFG into corresponding non-ambiguous CFL?

  It is also an Undecidable Problem as there doesn't exist any algorithm for the conversion of an ambiguous CFL to non-ambiguous CFL.
- Is a language Learning which is a CFL, regular?

  This is an Undecidable Problem as we can not find from the production rules of the CFL whether it is regular or not.

Counter Machines:

Counter machine has the same structure as the multi-stack machine but in place of each stack is a counter. Counters hold any non-negative integer,but we can only distinguish between zero and nonzero counters. That's the move of the counter machine depends on its state,input symbol and which if any of the counters are zero. In one move,the counter machine can

a)change state

b)Add or subtract 1 from any of its counters,independently. However a counter is not allowed to become negative,so it cant subtract from a counter that is currently 0.

Counter Machine may also be regarded as a restricted multi-stack machine. The restrictions are as follows,

a)There are only two stack symbols,which we shall refer to as Z0(the bottom of stack marker) and X.

b)Z0 is initially on each stack.

c)We may replace Z0 only by a string of the form X^iz0 for some

i >=0.

d)We may replace X only by X^i for some i >= 0. That's Z0 appears

only on the bottom of each stack and all other stack symbols if any are X.