Chapter-5

Undecidability

Church-Turing thesis (or Church's Thesis)

Turing machines that decides language and comput. functions and therefore halts on every input are useful computational devices.

Turing machine corresponding to formal notion of algorithm must halt on all inputs and therefore such machines are called algorithms.

This principle is called "Church-Turing thesis". This is not a theorem and so, cannot be proved mathematically.

It also says that problems unsolvable by Turing machine are impossible problems.

Universal Turing Machine:-

The turing machine that takes other turing machines (M) and their inputs (w), in encoded form and is capable to run 'M' on 'w' is called universal turing machine. This turing machine halts only if 'M' halts on 'w'. In other words, considering a turing machine as a program written in any programming language, programs written in this programming language can be interpreted by another program written in same language called universal turing machine.

Halting Problem:-

"For an arbitrary given turing machine, M and input w, there is no algorithm that decides whether or not 'M' accepts 'w'". Such problems for which no algorithm exists are known as undecidable or unsolvable problems.

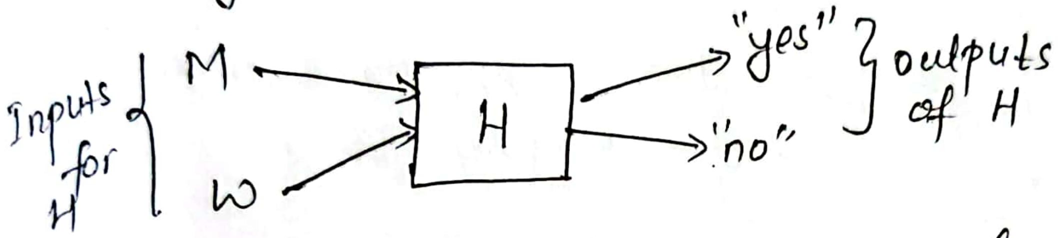
Telling whether a given Turing machine halts on given inputs is also an ~~unpr~~ undecidable problem and is called "Halting problem" for Turing machine.

* Proof that halting problem is unsolvable:-

Let's suppose, there exist a turing machine 'H' that takes other turing machine 'M' and input 'w' as input and decides whether or not 'M' halts on 'w' say,

'H' says "yes" if 'M' halts on 'w'

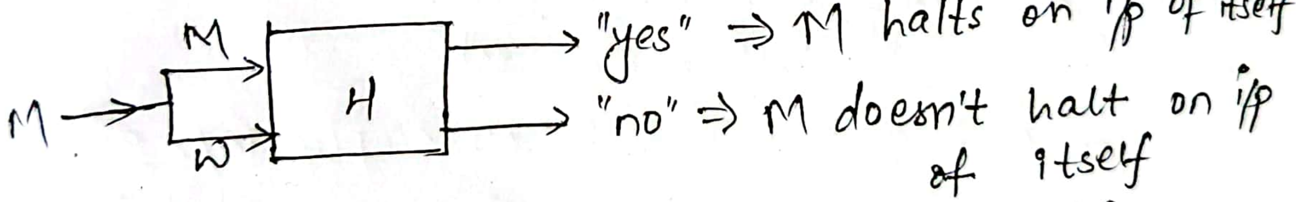
and 'H' says "no" if 'M' ~~halts~~ does not halt on 'w'.



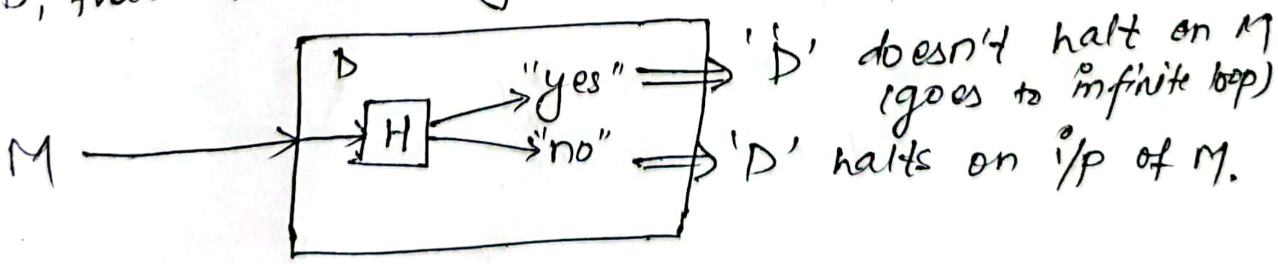
Using this algorithm, we can know if any machine halts on input of itself.

[By taking 'M' itself as input 'w' of 'M' (w=M)]

So, H can be modified as:-



This valuable machine 'H' that solves halting problem can be used to construct another machine 'D', that halts only if 'H' says "no".

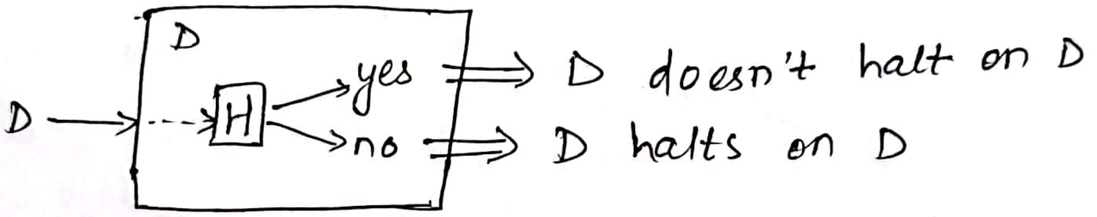


That means, if Machine 'M' with input 'M' is given to D then D halts iff 'M' doesnot halt on 'M'.
i.e, D accepts all turing machines that donot accept themselves.

Now, the unanswerable question is
"does 'D' halt on input of 'D'?"

D $\longrightarrow \boxed{D}$ $\Rightarrow$ will D halt?

The answer would be:
"D" halts on 'D' iff 'D' doesnot halt on D.



The statement is self contradictory which means our assumption that 'H' exists is wrong.
Hence, H doesn't exist i.e. Halting problem is unsolvable.
proved

Undecidable Problems about Turing Machine:-

1) Given a turing machine (TM) 'T' and input 'w', does 'T' halt on 'w'? (Halting problem)
2) Given a T.M, T does T halt on empty tape?
3) Given a TM, T is there any string on which 'T' halts?
4) Given a TM, T does T halt on every input string?
5) Does two T.Ms, T_1 and T_2 halts on same i/p strings?
$T_1 = T_2$?
6) Given a T.M, T is the language that T semidecides regular? contextfree? recursive?

The unsolvability of halting problem (that we proved earlier) implies the unsolvability of many other problems in mathematics and computer science such problems are proved unsolvable by reducing unsolvability of halting problem to these problems. PCP (Post correspondence Problem), Tiling problem etc are some examples.

[Halting problem $\leq$ PCP i.e, PCP is atleast as hard as halting problem]

Undecidable Problems about Grammars (Unrestricted)

1) For given grammar G and string 'w' to determine whether $w \in L(G)$?

2) For given grammar G, does $e \in L(G)$?

3) For two grammars G_1 and G_2, is $L(G_1) = L(G_2)$?

4) For given grammar G, is $L(G) = \phi$?

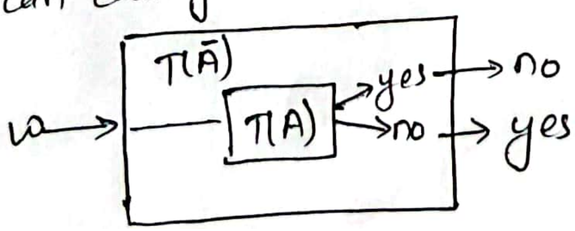
* Rice's Theorem:-

Every non-trivial property of a recursively enumerable language are undecidable.

Properties of Recursive and Recursively enumerable language

* If A is recursive, its complement is also recursive

⇒ For recursive language, T.M T(A) always halts and says "yes" if $w \in A$, and "no" if $w \notin A$. We can easily construct T.M for $\bar{A}$, using T(A) as,

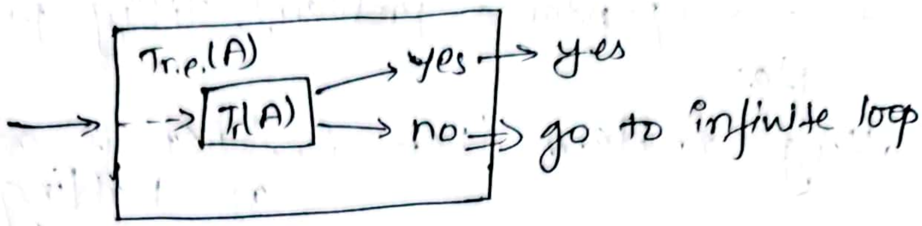


Then, this T.M, $T(\bar{A})$ say "yes" if $w \notin A$ i.e. $w \in \bar{A}$ and,

"no" if $w \in A$ i.e. $w \notin \bar{A}$.

$\therefore$ $\bar{A}$ is also recursive

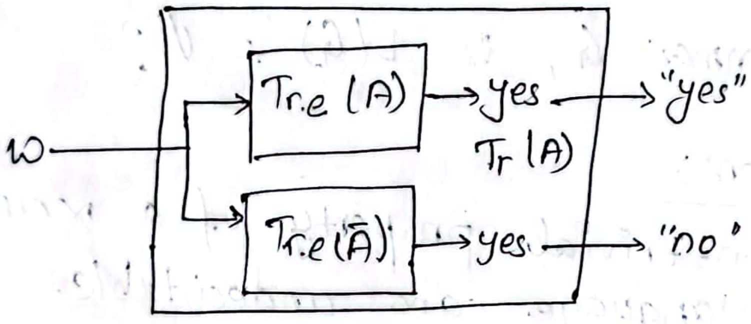
* If A is recursive, it is also recursively enumerable.



Here,

$T_r(A)$, Turing machine that decides A can be used to construct Tr.e.(A) that is turing machine that semidecides A. So, A is also recursively enumerable.

* If A and $\bar{A}$ both are recursively enumerable, then A is recursive.



As A and $\bar{A}$ are recursively enumerable, let Turing machine's Tr.e.(A) and Tre $(\bar{A})$ semidecides languages A and $\bar{A}$ respectively, for any input w, either Tre (A) accepts w if $w \in A$ or Tre $(\bar{A})$ accepts.

We can make use of these two machines to design machine Tr (A) as shown in figure above that decides language A.

$\therefore$ A is recursive.

Turing Machine as an enumerator.

→ Machine that enumerates / lists out all the strings of a language is termed as enumerator and the language is said to be turing enumerable iff such enumerator exists for that language.

→ This machine works like a generator / grammar.

→ Start with empty tape and one by one list out all the strings belonging to language into the tape.

A language is recursively enumerable iff it is turing enumerable.

Proof:-

case I:- If language is recursively enumerable, then it is turing enumerable.

⇒ Let T be turing machine that semidecides the recursively enumerable languages, L. Then, to show L is turing enumerable, we need to construct an enumerator, E for L.

Simple approach for E would be to feed every possible input string, w into T one-by-one and whenever T accepts w, just print w in the tape. But the big problem in this approach is that the machine T is not guaranteed to halt for $w \notin L$. T might go to infinite loop and so will our enumerator E.

This problem can be solved by using the simple, yet powerful technique of dovetailing.

→ Arrange the input strings in lexicographic order (increasing no. of alphabet symbols. eg:- e, 0, 1, 00, 01, 10, 11, 000, 001, ...)

In 1st phase, carry out 1st step of computation of T on 1st input string.

In 2nd phase, carryout 2nd step of computation of T on 1st input string and 1st step on 2nd input string. continue in similar pattern so that,

in nth phase, nth step of computation of T on 1st i/p string is carried out, (n-1)th step on 2nd and so on.

In the process if for some string w, if T accepts and halts, just write 'w' in the tape and continue processing for other strings.

In this way, sooner or later, all the input strings will be processed and all $w \in L$ will be printed in the tape by our enumerator, E.

→ A point to be noted here is that the order in which enumerator, E prints the straight w is not necessarily lexicographic. Longer strings might get printed earlier than the shorter ones.

<u>case II</u>: If language is turing enumerable, it is recursively enumerable.

⇒ Let E be enumerator for turing enumerable language L, T be a turing machine that takes input w and compares 'w' with each strings generated / printed by E. If a match is found, T accepts w and halts as $w \in L$, otherwise just keep on comparing 'w' with enumerator output. So, T semidecides L, hence L is recursively enumerable.

<u>proved</u>

A language is recursive iff it is lexicographically turing enumerable.

Case I:- Language, L is recursive $\Rightarrow$ L is lexicographically turing enumerable

Let T be turing machine that Decides the recursive language, L. Construct an enumerator, E that feeds all possible input strings 'w' to T in lexicographic order (increasing no. of alphabet symbols; eg: e, 0, 1, 00, 01, ...)

As, T is guaranteed to halt on every input and decide whether or not $w \in L$, print $w \in L$ into tape and skip all $w \notin L$.

This way, E will print on its tape, all $w \in L$ in lexicographic order and hence L is lexicographically turing enumerable.

Case II:- L is lexicographically turing enumerable $\Rightarrow$ L is recursive.

Let E be enumerator that lexicographically prints all the strings belonging to L.

T be turing machine that takes string 'w' as its input and compares 'w' with the outputs of E one-by-one from the begining. When a match is found, T simply accepts 'w'. As the order in which strings of L are enumerated by E is lexicographic, T rejects 'w' if it reads the string that should appear later than 'w' in lexicographic order.

Hence, as T decides the language L, L is recursive.

proved

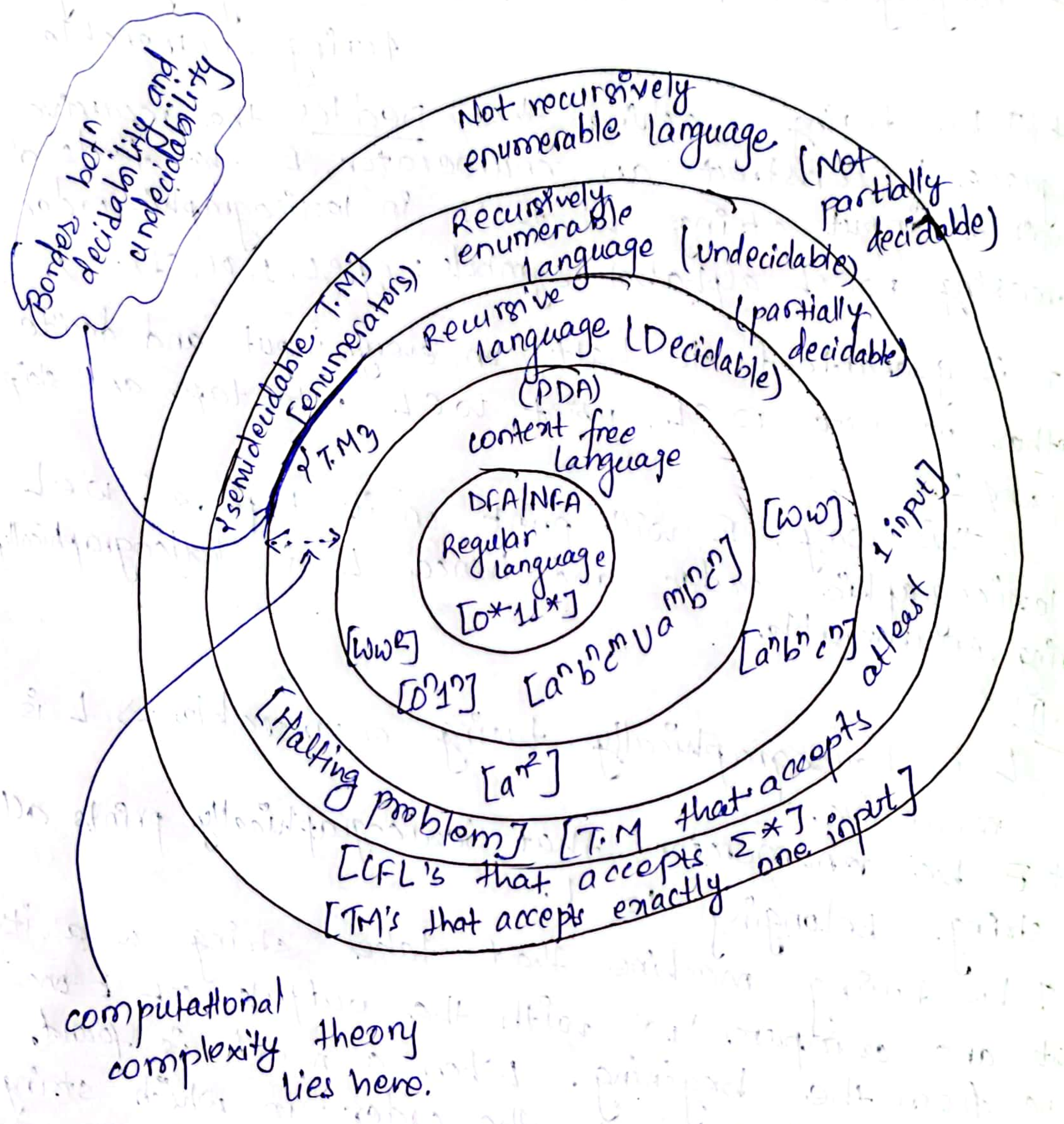


fig:- Different levels of languages, corresponding machines ~~that~~ accepting them with examples.