Unit - 8: Undecidability - Short Notes

1) Post's Correspondence Problem (PCP)

✓* **Definition:** An instance of Post's Correspondence Problem (PCP) consists of two lists of strings over some alphabet Σ. The two lists must be of equal length

- We generally refer to the A and B lists where $A = w_1, w_2, \ldots, w_k$ and $B = x_1, x_2, \ldots x_k$ for some integer k.
- For each i, the pair (w_i, x_i) is said to be a corresponding pair.
- This instance of PCP has a solution if there is a sequence of one or more integers $i_1, i_2, \ldots i_m$ that when interpreted as indexes for strings in A and B lists. will yield the same string. That is -

$w_{i_1}, w_{i_2}, \ldots w_{i_m} = x_{i_1}, x_{i_2}, \ldots x_{i_m}$. We say the sequence $i_1, i_2, \ldots i_m$ is a solution to this instance of PCP, if so.

✓* **Eg:** The post's correspondence problem is
Given an instance of PCP, tell whether this instance has a solution

i	List A w_i	List B x_i
1	110	110110
2	0011	00
3	0110	110

Let $\Sigma = \{0,1\}$, and let the A and B lists be defined as above. In this case, PCP has a solution. For instance, let $m=3$, $i_1=2$, $i_2=3$ and $i_3=1$. The solution is the list 2, 3, 1. We verify that this list is a solution by Concatenating the corresponding strings in order for the two lists that is,
$w_2 w_3 w_1 = x_2 x_3 x_1 = 0011011 0110$. This solution is not unique. For eg: 2, 1, 1, 3, 2, 1, 1, 3 is another solution

✓*

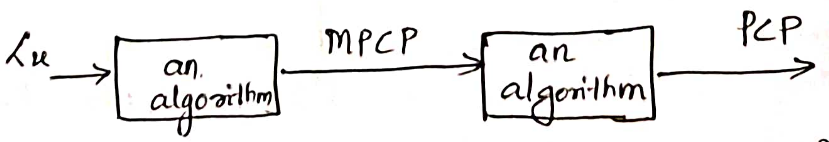


Fig: Reductions proving the undecidability of Post's Correspondence Problem

2

2) Recursive Languages:

[Question can be even asked [redacted] / [redacted]
Write short note on Recursive language and it's relationship with RE and non-RE languages]

A Language L is recursive if $L = L(M)$ for some Turing Machine M Such that:

a) If w is in L then M accepts (and therefore halts)

b) If w is not in L, then M eventually halts, although it never enters an accepting state.

Explanation:

A TM of this type corresponds to our informal notion of an "algorithm"; a well defined sequence of steps that always finishes and produces an answer. If we think of the language L as a "problem", then problem L is called decidable if it is a recursive language, and it called undecidable if it not recursive language

- The existence or non existence of an algorithm to solve a problem is often of more importance than the existence of TM to solve the problem.
- The TM's that are not guaranteed to halt may not give us enough information to conclude that a string is not in the language. Thus, dividing problems or languages between the decidable - those whose are solved by an algorithm and those that are undecidable is more important than division between recursively enumerable languages (RE) and non recursively enumerable languages.

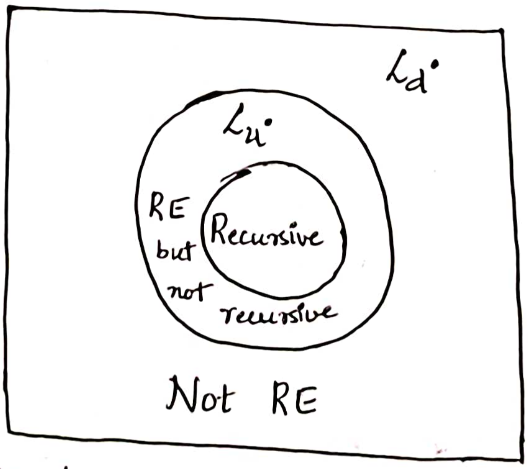


Fig: Relationship between the recursive, RE and non-RE languages

Here L_u is universal language, L_d is non RE language.

3) Universal Language:

Universal Language, L_u is defined to be the set of binary strings that encode, a pair (M, w) where M is a TM with the binary input alphabet and w is a string in $(0+1)^*$, such that w is in L(M).

- That is, L_u is the set of strings representing a TM and an input accepted by that TM.
- We show that there is a TM U, often called as universal Turing Machine, such that $L_u = L(U)$.

* For Universal Language
1) Write Definition
2) Write diagram & explanation

* For Universal Turing Machine
1) Write the diagram
2) Working of U

Universal TM which accepts Universal language:

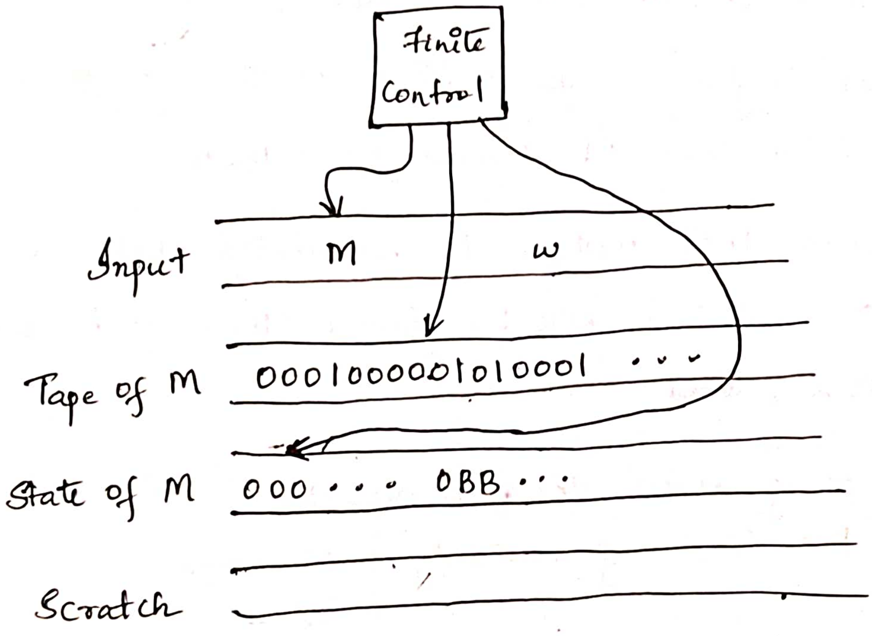


Fig: Organization of Universal Turing Machine.

Explanation:
Here, The transitions of M are stored initially on the first tape, along with the string w. A second tape will be used to hold the simulated tape of M, using the same format as for the code of M. That is, tape symbol X_i of M will be represented by 0^i and tape symbols will be separated by single 1's. The third tape of U holds the state of M, with state q_i represented by i 0's.

Note * working of Universal Turing Machine you write only when the question is Write short Note on Universal Turing Machine.

Operation (working) of Universal Turing Machine (U):

If the question is Universal Language then you need not write.

1) Examine the input to make sure that the code for M is a legitimate code for TM. If not, U halts without accepting.

2) Initialize second tape to contain the input w, in its encoded form. That is, for each 0 of w, place 10 on the second tape and for each 1 of w, place 100. Note that the blanks on the simulated tape of M which are represented by 1000, will not actually appear on that tape.

3) Place 0, 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 ~~stape~~ tape for a transition $0^i 1 0^j 1 0^k 1 0^l 1 0^m$, such that 0^i is the state on tape 3, and 0^j is the tape symbol of M that begins at the position on tape 2 scanned by U. This transition is the one M would next make.

5) If M has no transition that matches the simulated state and tape symbol in (4), no transition will be found. Thus M halts in the simulated configuration.

6) If M enters its accepting state, then U accepts.
U accepts the coded pair (M, w) if and only if M accepts w.

4) The Halting Problem:

The Halting Problem for TM is a problem similar to L_u - One that is RE but not recursive.

- We define halting problem H(M) for TM M to be set of inputs w such that M halts given input w, regardless of whether or not M accepts w. Then, the halting problem is the set of pairs (M, w) such that w is in H(M).

- This problem/language is another example of one that is RE but not recursive;

- It can be informally stated as "Given a Turing Machine M and an input w with the initial Configuration q_0, after some computations do the machine M halts ?"

In other words we have to identify whether (M, w) halts or not when w is applied as the input.

- It is not possible to find the answer for Halting problem by simulating the action of M on w by a universal Turing Machine, because there is no limit on the length of the computation. If M enters into an infinite loop, then no matter how long we wait, we can never be sure that M is in fact in a loop. The machine may be in loop because of very long computation. What is required is an algorithm that determines the correct answer for any M and w by performing some analysis on machine's description and the input.
- The domain of this problem is to be taken as the set of all Turing Machines and all w. We must always know what the domain is, because this may affect the conclusion. The problems may be decidable on some domain but not on another.

5) Recursively Enumerable Language (RE):

Definition: A Language L is recursively enumerable language if it is accepted by turing machine.

Given TM $M = (Q, \Sigma, \Gamma, \delta, q_0, B, F)$, the language accepted by TM i.e $L(M)$ is given as the set of strings w in Σ^* such that $q_0 w \vdash^* \alpha p \beta$ for some state p in F and any tape strings α and β.

- The language accepted by the TM is recursively enumerable language i.e given a string w which is input to TM, the machine halts and outputs Yes if it belongs to the language. If w does not belong to the language L, the TM halts and outputs NO.

- The languages with TM which will always halts and output yes if it belongs to the language or output no if it does not belong to the language are called decidable languages. If an algorithm exists to solve a given problem, then the problem is decidable otherwise it is undecidable problem.

Note:

Other short Notes you refer Notes or Text Book which is discussed in the class:

1) Applications of CFG's
2) Regular expression in Unix
3) Applications of RE (Regular Expressions)
4) Multitape TM's
5) Non-deterministic TM's
6) Inherent ambiguity of CFL (context-free languages)
7) Chomsky hierarchy

6) Language which is not Recursively Enumerable:

A language L which is not accepted by a Turing Machine is a language that is not Recursively Enumerable.

The decision problems that have decision algorithms the output of which is yes/no are called solvable problems. Any instance of a problem for which the Turing Machine halts whether the input is accepted/rejected is called decidable or solvable. There are problems that are not solvable. Our long-range goal is to prove undecidable the language consisting of pairs (M, w) such that:

a). M is a TM (coded in binary) with the input alphabet {0,1}
b) w is a string of 0's and 1's
c) M accepts input w.

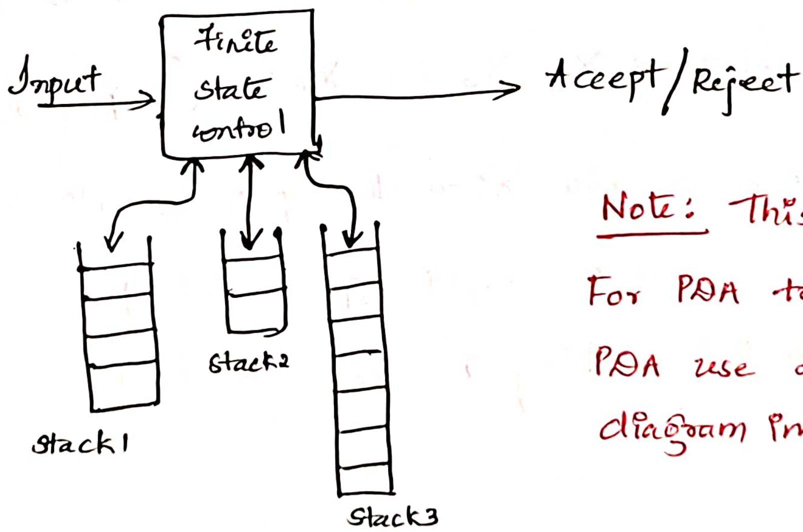
We must give coding for TM, that uses only 0's and 1's regardless of how many states the TM has. Once we give this coding, we can treat any binary string as if it were a TM.

- It involves the language called "diagonalization language" L_d, which consists of all those strings w such that TM does not accept the input w.

7) Multistack Machines: [Multistack PDA]

We know that the TM can accept languages that are not accepted by any PDA with one stack. If we give PDA two stacks, then it can accept any language that a TM can accept.

- The multi-stack machine is generalized model of PDA. A machine with three stacks is shown below:



Note: This is 3 stack PDA
For PDA to be 2 stack PDA use 2 stacks in the diagram instead of 3

Fig: A machine with three stacks

- A k-stack machine is a deterministic PDA with k stacks. It obtains its input from the input source i.e input alphabets Σ

Similar to the PDA. The multistack machine has a finite control which is in one of a finite set of states. It has a finite stack alphabet, which it uses for all its stacks.

- A move of the multistack machine is based on:

1) The state of the finite control.
2) The input symbol which is chosen from the input alphabet. The multistack machine can make a move using ϵ input, but to make the machine deterministic, there cannot be a choice of an ϵ-move or a non-ϵ move in any situation.
3) The top stack symbol on each of its stacks.

In one move, the multistack machine can:

a) change to a new state
b) Replace the top symbol of each stack with a string of zero or more stack symbols. There can be different replacement string for each stack.

A typical transition rule for a k-stack machine looks like

$$\delta(q, a, X_1, X_2, \dots X_k) = (p, \gamma_1, \gamma_2, \dots \gamma_k)$$

That is, in state q, with X_i on the top of the ith stack, the machine may consume a from its input, go to state p, replace X_i on the top of the ith stack by string γ for each $i = 1, 2, \dots k$

Other Topics to study in Turing Machines: [Refer Padma Reddy or Standard Text Book]

1) Restricted Turing Machines — a) Turing Machine with Semi-Infinite Tape
   b) Multistack Machines [Already given]
   c) Counter Machines
   d) Off-line Turing Machines
   e) Linear Bounded Automata

2) Programming Techniques for Turing Machine.
   - a) Storage in the state
   - b) Multiple Tracks
   - c) Subroutines

Please Refer all the previous year question papers.

ALL THE BEST, DO WELL