(2) An undecidable problem that is RE

(2.1) Recursive languages

we call a language L recursive if $L = L(M)$ for some TM (M) such that:

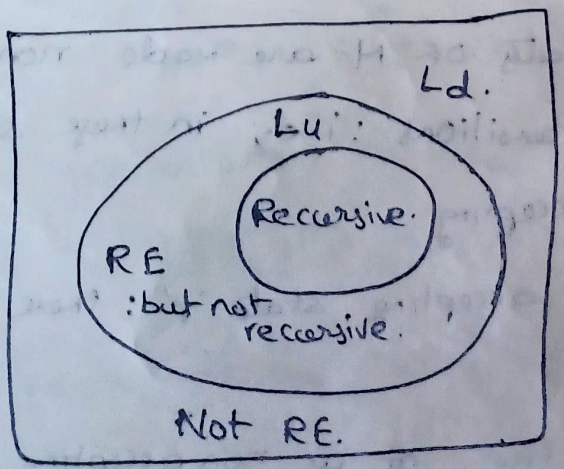
1. If w is in L, then M accepts (halts)
2. If w is not in L, then M eventually halts, although it never enters an accepting state

If we consider any problem 'L' is decidable then it is a "recursive language" and if 'L' is undecidable then it is "not a recursive language".

for dividing the problems & languages between the decidable and undecidable :-

decidable - those that are solved by an algorithm.

undecidable is often more important than the division between the recursively enumerable languages (those that have TM's of some sort) and the "non recursively enumerable languages" (which have no TM at all).



RE - Recursively Enumerable.

fig: Relation b/w Recursive, RE and NOT RE.

(2.2) Complements of Recursive and RE languages

Recursive languages are closed under complementation.

⇒ If a language L is RE, but $\bar{L}$ (complement of L) is not RE, then we know that L can not be recursive;

⇒ For if 'L' is recursive, then $\bar{L}$ would also be recursive and thus surely RE.

Theorem: If L is a recursive language, so is $\bar{L}$.

Proof:- Let $L = L(M)$ for some TM, M halts always. We construct a TM $\bar{M}$ such that $\bar{L} = L(\bar{M})$ by the construction suggested in the below figure.

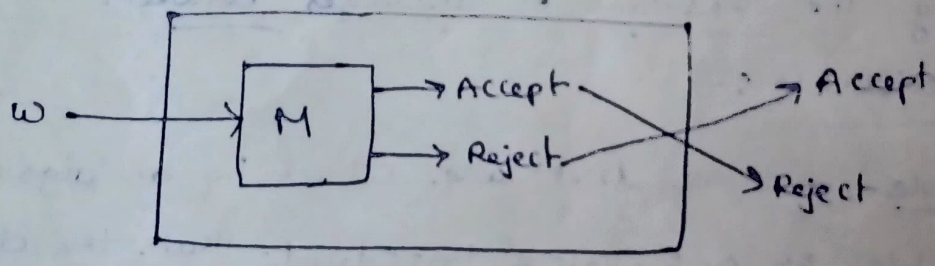


fig: construction of a TM accepting the complement of a recursive language.

That is, $\bar{M}$ behaves just like M. However, M is modified as follows to create $\bar{M}$:

1. The accepting states of M are made non accepting states of $\bar{M}$ with no transitions: i.e., in these states $\bar{M}$ will halt without accepting.
2. $\bar{M}$ has a new accepting state 'r'; there are no transitions from r.
3. For each combination of a non accepting state of M and a tape symbol of M such that M has no transition, add a transition to the accepting state r.

since M' is guaranteed to halt, we know that M is also guaranteed to halt.

M accepts exactly those strings that M does not accept.

Thus M accepts L.

(2.3) Universal language (&) Universal Turing Machine.

A Universal TM M_U is an automaton that, given as input the description of any TM 'M' and a string w, can simulate the computation of M for input w.

To construct such a M_U, we first choose a standard way of describing TMs.

$$M = (Q, \{0,1\}, \{0,1,B\}, \delta, q_1, B, q_2)$$ where $Q = \{q_1, \cdots q_n\}$

q_1 is the initial state and 'q_2' is the single final state.

The alphabet $\{0,1,B\} \in T$ are represented as a_1, a_2, a_3.

and direction left and right are represented as D1 and D2.

The transitions of TM are encoded in a special binary representation where each symbol is ~~represented~~ seperated by '1'.

In a transition,

$$\delta(q_i, a_j) = (q_k, a_l, D_m)$$

the binary representation for the transition is given as

$$0^i 1 0^j 1 0^k 1 0^l 1 0^m$$

The binary code for the Turing machine 'M' that has transitions $t_1, t_2, t_3 \cdots t_n$ is represented as

$$111\ t_1\ 11\ t_2\ 11\ t_3\ 11\ t_4 \cdots 11 t_n 111.$$

If a string to be verified, then the problem is represented as a tuple $\langle M, w \rangle$ where M is the definion of TM, and w is the input string

→ let $M = (\{q_1, q_2, q_3\}, \{0,1\}, \{0,1,B\}, \delta, q_1, B, \{q_2\})$ have moves defined as

$$\delta(q_1,1) = (q_3,0,R)$$
$$\delta(q_3,0) = (q_1,1,R)$$
$$\delta(q_3,1) = (q_2,0,R)$$
$$\delta(q_3,B) = (q_3,1,L)$$

Give the problem representation for the string $w = 1011$

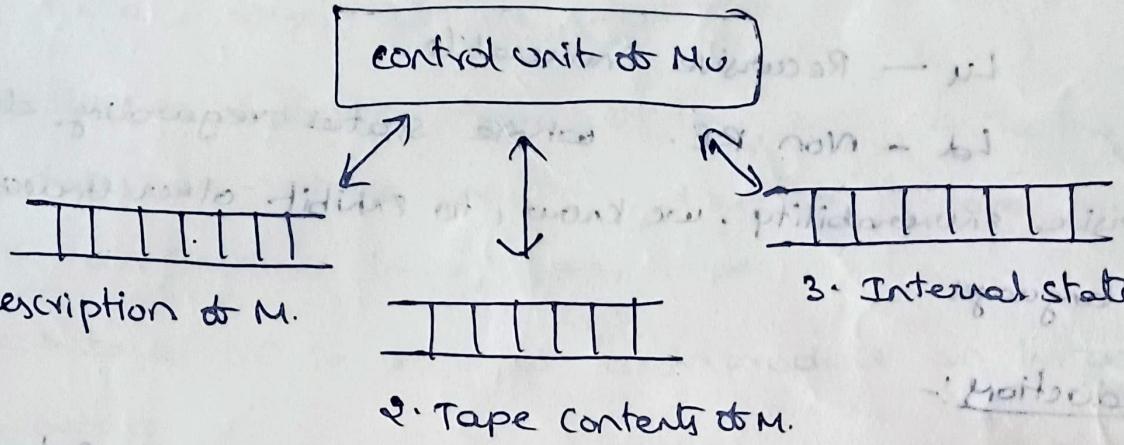
Soln: Let Binary representation for states $\{q_1, q_2, q_3\}$ be $\{0, 00, 000\}$. for alphabet $\{0,1,B\}$ be $\{0, 00, 000\}$ and for directions $\{R, L\}$ be $\{0, 00\}$. The transitions are represented as follows

Transition	Binary representation
$\delta(q_1,1) = (q_3,0,R)$	0100100010100
$\delta(q_3,0) = (q_1,1,R)$	000101010010
$\delta(q_3,1) = (q_2,0,R)$	000100100101000
$\delta(q_3,B) = (q_3,1,L)$	000100010001001000

The problem instance $\langle M, 1011 \rangle$ is represented as

111 0100100010100 11 000101010010 11 000100100101000 11 000100010001001000 111 1011

The following figure shows the organization of a universal TM that has a control unit and three tapes



for any input M and w, Tape1 will keep an Encoded definition of M; Tape 2 will contain the tape contents of M and Tape # 3, the internal state of M.

(2.4) Undecidability of the universal language.

Universal language (Lu) is RE but not Recursive language. Knowing that Lu is undecidable is in many ways more valuable than our previous discovery that Ld En is not RE. The reason is that the reduction of Lu to another problem P can also be used to show that there is no algorithm to solve P, regardless of whether or not P is RE. However, reduction of Ld to P is only possible if P is not RE, so Ld cannot be used to show undecidability for those problems that are RE but not recursive. On the other hand if we want to show a problem not to be RE, then only Ld can be used; Lu is useless since it is RE.

(3) Undecidable problems about Turing Machines

We shall now use the languages

L_u — Recursive Enumerable

L_d — Non RE. whose stated regarding decidable and recursive enumerability. we know, to exhibit other undecidable non RE languages

(3.1) Reductions:-

In general, if we have an algorithm to convert instances of a problem P_1 to instances of a problem P_2 that have the same answer, then we say that P_1 reduces to P_2.

→ we can use this proof to show that P_2 is at least as hard as P_1. Thus if "P_1 is not recursive", then "P_2 cannot be recursive".

→ If P_2 is non-RE, then P_1 cannot be RE.

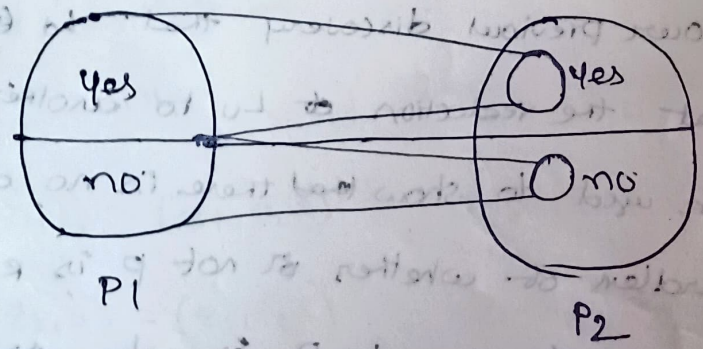


fig:- Reductions turn positive instances into positive and negative to negative

In the figure, a reduction must turn any instance of P_1 that has a "yes" answer into an instance of P_2 with a "yes" answer and every instance of P_1 with a "no" answer must be turned into an instance of P_2 with a "no" answer

* Note that, it is not essential that every instance of P_2 be the

be the target of one or more instances of P_1, and in fact it is quite common that only a small fraction of P_2's is a target of the production.

Formally, a reduction from P_1 to P_2 is a Turing Machine that takes an instance of P_1 written on its tape and halts with an instance of P_2 on its tape.

The instance of P_1 as input and produce an instance of P_2 as output.

** If there is a reduction from P_1 to P_2, then:

a) If P_1 is undecidable then so is P_2.

b) If P_1 is non-RE, then so is P_2.

(3.2) Turing Machines that accept the Empty language:

Consider 2 languages L_e and L_{ne}. Each consists of binary strings.

If w is a binary string, then it represents some TM, M_i.

If $L(M_i) = \phi$: that is, M_i does not accept any input, then w is in L_e. Thus, L_e is the language consisting of all those encoded TM's whose language is empty.

On the other hand, if $L(M_i)$ is not the empty language, then w is in L_{ne}. Thus, L_{ne} is the language of all codes for TMs that accept atleast one input string.

* $L_e = \{M \mid L(M) = \phi\}$, it is non-RE.

** $L_{ne} = \{M \mid L(M) \neq \phi\}$; it is RE.

3.3 Rice's Theorem and properties of RE languages:-

* The property of the RE languages is simply a set of RE languages. Thus, the property of being context free is formally the set of all CFL's.

* The property of being Empty is the set $\{\phi\}$ consisting of only the Empty language.

* The property is trivial if it is either Empty (i.e satisfied by no language at all) or is all RE languages. Otherwise, it is "non-trivial"

* Note that the Empty property, ϕ, is different from the Property of being an Empty language, $\{\phi\}$.

If P is a property of the RE languages, the language L_P is the set of codes for Turing machines M_i such that $L(M_i)$ is a language in P.

When we talk about the decidability of a property P, we mean the decidability of the language L_P.

Theorem: (Rice's Theorem) Every non-trivial property of the RE languages is undecidable.

Proof: Let 'P' be a non trivial property of the RE languages, Assume to begin that ϕ, the Empty language is not in P. we can prove by contradiction procedure.

$\Rightarrow$ P is nontrivial, there must be some non-Empty language L that is in P. Let M_L be a TM accepting L.

we shall reduce L_U to L_D, thus proving that L_D is undecidable, ∵ L_U is undecidable.

→ The Algorithm to perform the reduction takes as input a pair (M, w) and produces a TM M'.

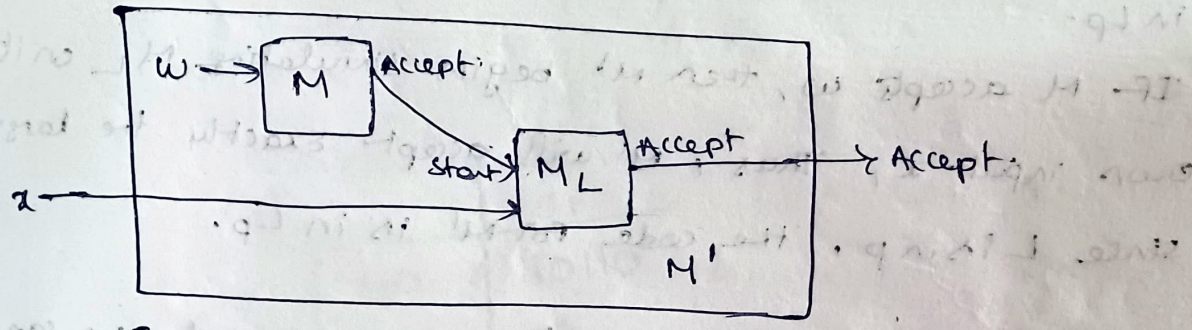


Fig: Construction of M' for the proof of Rice's Theorem.

The design of M' is suggested in figure; $L(M')$ is ϕ if M doesnot accept w, and $L(M') = L$ if M accepts w.

→ M' is a two-tape TM. One tape is used to simulate M on w. The simulation of M on w is "built in to" M'; the latter TM doesnot have to read the transitions of M from on a tape of its own.

→ The other tape of M' is used to simulate M_L on the input x to M', if necessary.

Again, the transitions of M_L are known to the reduction algorithm and may be "built into" the transitions of M'.

The Turing Machine M' is constructed to do the following:

1. Simulate M on input w. Note that w is not the input to M'; rather, M' writes M and w onto one of its tapes and simulates the Universal TM 'U' on

on that pair.

2. If M doesnot accept w, then M' does nothing else. M' no accepts its own input, x; so $L(M') = \phi$. Since we assu ϕ is not in property P; that means code for M' is no in L_P.

3. If M accepts w, then M' begins simulating M on its own input x. Thus, M' will accept exactly the language L. since L is in P, the code for M' is in L_P.

→ Observe that constructing M' from M and w can be carried out by an algorithm. since algorithm turns (M, w) in to an M' that is in L_P if and only if (M, w) is in L_M, this algorithm is a reduction of L_M to L_P and proves that the property P is undecidable.

(4) Post Correspondence problem

An instance of post correspondance 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, and write $A = w_1, w_2 \cdots w_k$ and $B = x_1, x_2 \cdots x_k$ for some integer k. for each i, the pair (w_i, x_i) is said to be a corresponding pair.

we say this instance of PCP has a solution, if there is a sequence of one or more integers $i_1, i_2 \cdots i_m$ in that, when interpreted as indexes for strings in the A and B lists

yield the same string.

i.e. $w_{i_1} w_{i_2} \cdots w_{i_m} = x_{i_1} x_{i_2} \cdots x_{i_m}$. we say the sequence $i_1, i_2, i_3 \cdots i_m$ is a solution to this instance of PCP, if so.

The POST correspondance 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

fig: An instance of PCP

(1) let $\Sigma = \{0, 1\}$ and let the A and B lists be defined in fig. In this case PCP has a solution. for instance let $m=3$, $i_1=2$, $i_2=3$ and $i_3=1$

solution sequence 2 3 1

w_i	0011	0110	110
x_i	00	110	110110

After elimination of blanks and concatenation of strings in the top and bottom rows seperately we get,

0011 0110 110 } same.
00110110110

∴ these selections from a solution to the PCP problem

(2) Problem:- consider the following sequence and find whether it has a solution or not.

i	g_i	h_i
1	100	1
2	0	100
3	1	00

let us consider the results of selection of sequence $\{1,3,1,1,3,2,2\}$ accordingly. They are shown in the following table and each selection is shown in table.

solution sequence	1	3	1	1	3	2	2
string g	100	1	100	100	1	0	0
string h	1	00	1	1	00	100	100

after elimination of blanks and concatination of strings in the top and bottom rows seperately,

G — 100 1 100 100 100
H — 1001 00100 100

} same.

∴ these selections form a solution to the PCP problem.

(3) Problem : consider the following sequence and find whether it has a solution or not

i	g	h
1	01	0
2	110010	0
3	1	1111
4	11	01

Let us consider the tuple $(i_1, i_2, i_3, i_4, i_5) = (1, 3, 2, 4, 4$

is a witness for a positive solution because

$x_1 x_3 x_2 x_4 x_4 x_3 = y_1 y_3 y_2 y_4 y_4 y_3 = 011110010111111.$

⇒ so it has a solution.

(4) Consider :

i	g	h.
1	0	10
2	01	1

This problem has no solution as the strings formed by g's would always start with 0. and the strings formed by h's would always start with 1.

***

PCP is very useful for showing the undecidability of many other problems by means of reducibility

1. Interactable problems

Now focus on problems that are decidable, and ask which of them can be computed by Turing Machines that run in an amount of time that is polynomial in the size of the input.

* The problems solvable in polynomial time on a typical computer are exactly the same as the problems solvable in polynomial time on a Turing Machine.

Here, they introduce the theory of "intractable" "intractability", that is, techniques for showing problems not to be solvable in polynomial time.

(1.1) The classes P and NP

Basic concepts of intractability theory: the classes of P and NP of problems solvable in polynomial time by deterministic and non-deterministic TM's respectively and the technique of polynomial time reduction.

NP-completeness, a property that certain problems in NP have; they are atleast as hard as any problem in NP.

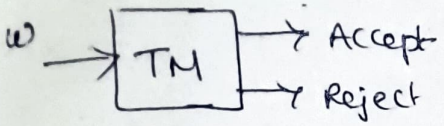
(1.1.1) Problems solvable in polynomial Time

A TM 'M' is said to be of time complexity T(n) if when ever M is given an input w of length n, M halts after making atmost T(n) moves, regardless of whether or not M accepts.

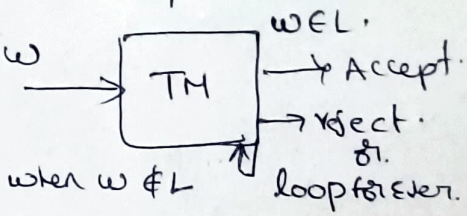
T(n) is a polynomial in n. we say a language L is in class P if there is some polynomial T(n) such that $L = L(M)$ for some deterministic TM M of Time complexity T(n).

Ex: Kruskal's algorithm.

Recursive language



Recursively Enumerable language



(*) If we consider any language L and M be the Turing machine represented the language L then any string $w \in L(M)$ which is either accept or Reject by Turing Machine (M) then the language 'L' is called Recursive language.

1. A language that is not Recursively Enumerable

Consider the language L_d the "diagnolization language". which consists of all those strings 'w' such that the TM represented by 'w' doesnot accept the input 'w'.

we shall show that L_d has no TM at all for a language is showing something stronger than that language is undecidable.

(1.1) codes for Turing machine