6 Decidability

What is a decision problem:
A decision problem is a set of related statements, each of which must be either true or false.
For example:
"For context-free grammar (G), the language L(G) is ambiguous or not?
For some context-free grammar (G), this is true, and for some, it is false.
But we can clearly say that the statement is either true or false.
Basically, we have two kinds of problems:

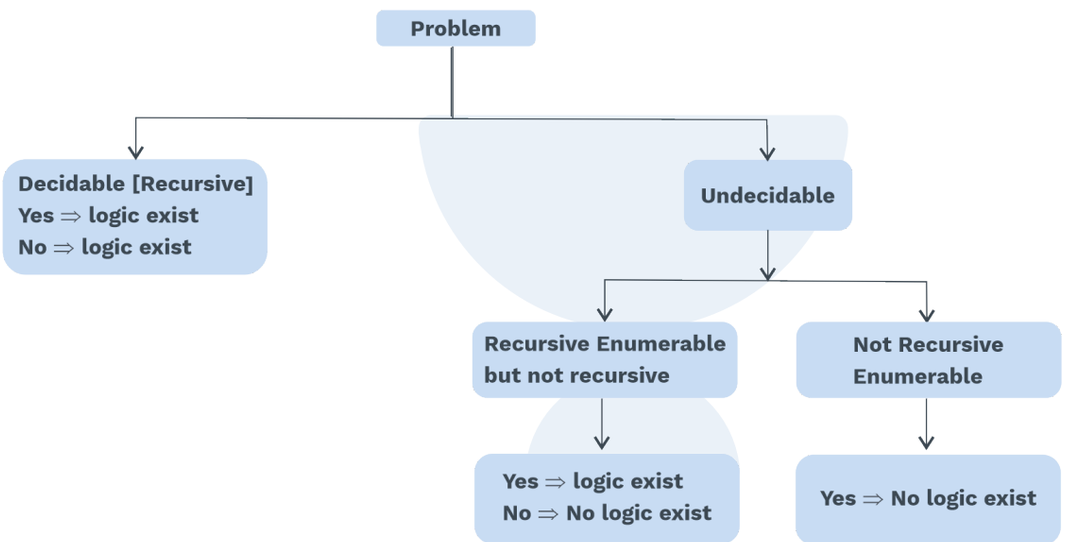


Fig. 6.1 Decidability

Note:

- For decidable problem, the Halting Turing Machine (HTM) exist.
- For undecidable problem\s, the Halting Turing Machine (HTM) does not exist, but Turing Machine (TM) may or may not exist.
- If a problem is not recursive enumerable (RE), then no Turing Machine exists for that problem.

Language:
The set of all strings that can be generated from the grammar.
Language can be finite or infinite. If a language consists of a finite number

of strings, then it is called a Finite Language; otherwise, we can say it is an infinite language.

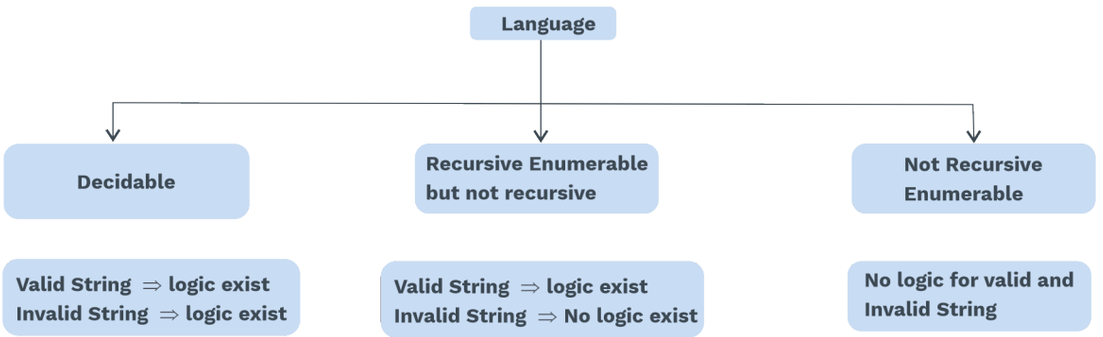


Fig. 6.2

Example:

Consider a String w and a language L

1) If $w \in L$ or $w \notin L$

logic exist logic exist

For both the cases, we have logic, then the language is decidable.

2) If $w \in L$ or $w \notin L$

logic exist No logic exist

Logic exists for valid strings. So, the language is Recursively Enumerable but not Recursive.

3) If $w \in L$ or $w \notin L$

No logic exist No logic exist

For both the cases, we do not have logic, then the language is not Recursively Enumerable.

Encoding a turing machine (TM):

Each Turing Machine with input alphabet {0, 1} may be thought of as a binary string.

To represent a Turing Machine $(M) = (Q, \Sigma, \Gamma, \delta, q, B, F)$ as a binary string where $\Sigma = \{0, 1\}$.

First, assign integer to the tape symbols, states and directions Left (L) and Right (R).
Let the states be: q_1, q_2, q_3, q_4 ...q_k.
Here q_1 is a Starting state, q_2 is a Final State and q_3, q_4 ... q_k are other states.
Let tape symbols are: X_1, X_2, X_3, ...X_m.

X_1 = 0
X_2 = 1
X_3 = B (Blank)

Let the direction left be D_1 and the direction Right be D_2.

Encoding of transition function (δ):
one transition rule is,

$\delta(q_i, X_j) = (q_k, X_l, D_m)$ for some integers i, j, k, l and m.
Now, code this rule by the string- $0^i10^j10^k10^l10^m$
where i, j, k, l, m should be greater than 0 (i, j, k, l, m > 0).
There should not be two consecutive 1's within the code for a single transition.
The code for the entire Turing Machine will be:

$$Q_1\ 11\ Q_2\ 11\ Q_3\ 11\ \ldots\ Q_n$$

Each transition is separated by two consecutive 1's and $Q_1, Q_2 \ldots Q_n$ are the transition of Turing Machine.

Example:
Consider the Turing Machine (TM)

$M = (\{q_1, q_2, q_3, q_4, q_5\}, \{0, 1\}, \{0, 1, B\}, \delta, q_0, B, \{q_2\})$

where δ consists of the rules:

$\delta(q_1, B) = (q_5, B, L)$

$\delta(q_4, 1) = (q_3, B, R)$

$\delta(q_5, B) = (q_4, 1, R)$

$\delta(q_3, 1) = (q_2, 1, L)$

Codes for each transition:

$(q_i, X_j, q_k, X_i, D_m)$

1) $\delta(q_1, B) = (q_5, B, L)$
$(q_1, X_3, q_5, X_3, D_1)$
So,

$0^1 10^3 10^5 10^3 10^1$
01000100000100010

2) $\delta(q_4, 1) = (q_3, B, R)$
$(q_4, X_2, q_3, X_3, D_2)$
So,
$0^4 10^2 10^3 10^3 10^2$
000010010001000100

3) $\delta(q_5, B) = (q_4, 1, R)$
$(q_5, X_3, q_4, X_2, D_2)$
So,
$0^5 10^3 10^4 10^2 10^2$
00000100010000100100

4) $\delta(q_3, 1) = (q_2, 1, L)$
$(q_3, X_2, q_2, X_2, D_1)$
So,
$0^3 10^2 10^2 10^2 10^1$
00010010010010
A code for Turing Machine (M):
0100010000010001011000010010001000100**11**0000100010000100100**11**0
0010010010010
Here, two consecutive 1's are used as Separators.

The diagonalization language (l_d):
It is the set of strings (w_i) such that w_i is not in $L(M_i)$.

$$L_d = \{w_i \in (0+1)^+ \mid w_i \notin L(M_i)\}$$

Let,

$$\Sigma = \{a, b\}$$

And we know, Σ^* is countable; now we must prove 2^{Σ^*} is uncountable.

Prove by contradiction:

Let 2^{Σ^*} is countable and it has one-one mapping.
We know,

$$\Sigma^* = \in, a, b, aa, ab, ba, bb, aaa \ldots$$

Let,

$$L_1 = \{\in, a\}$$

$$L_2 = \{a, b\}$$

$$L_3 = \{\in, a, aa\}$$

$L_4 = \{b, ab, ba\}$

$L_5 = \{aa, ba, bb\}$

$L_6 = \{\in, aaa\}$ and so on.

Now, mapping L_1, L_2, L_3, L_4, L_5 and L_6 with Σ^*.

	∈	a	b	aa	ab	ba	bb	aaa	...
L_1 :	1	1	0	0	0	0	0	0	...
L_2 :	0	1	1	0	0	0	0	0	...
L_3 :	1	1	0	1	0	0	0	0	...
L_4 :	0	0	1	0	1	1	0	0	...
L_5 :	0	0	0	1	0	1	1	0	...
L_6 :	1	0	0	1	0	0	0	1	...
; ;	;	;	;	;	;	;	;	;	...
; ;	;	;	;	;	;	;	;	;	...
; ;	;	;	;	;	;	;	;	;	...

Diagonal

Fig. 6.3 The Table That Represents M Accepts W

To construct Diagonal Language (L_d), we complement the diagonal.
Complement of diagonal would begin:

0 0 1 1 1 1...

∈ a b aa ab ba...

Thus, L_d would contain w_i = {b, aa, ab, ba...} except '∈' and 'a'.
Since L_d is not present in Diagonal Matrix or given set of languages. So, the initial assumption was wrong.

By contradiction, 2^{Σ^*} is uncountable.

Proof that L_d is not Recursively Enumerable (RE):
By the above intuition, there is no Turing Machine that accepts the language L_d. So, L_d is not Recursive Enumerable (RE).

An undecidable problem that is recursively enumerable (RE):
It has been already established that diagonal languages are not recognized by Turing machines.
Let's have insight into the area of recursively enumerable languages.

Case 1:
A Turing machine will necessarily halt for strings that are not inclusive to a given language. However, it may or may not enter into an accepting state.

Case 2:
Let us consider a scenario where recursively enumerable languages are involved. The Turing machine will necessarily halt if the input string is inclusive to the language. However, for excluded input strings, the Turing machine may get inter-wined into an infinite loop or halt.

Recursive languages:
A recursive language L If L = L(M) for some Turing Machine (M) such that:
1) $w \in L \Rightarrow$ Halts and accepts
2) $w \notin L \Rightarrow$ Halts and rejects

Note:

Recursive language is always decidable. If language is not recursive, then it is undecidable/semi decidable.

Compliment of recursive and recursive enumerable language (RE):
If a language L is Recursive Enumerable but the complement of recursive enumerable language $(\overline{L})$ is not Recursive Enumerable (RE), then we can say L can not be Recursive.
If L is Recursive, then the complement of $L(\overline{L})$ would also be recursive, which implies L is Recursive Enumerable(RE).

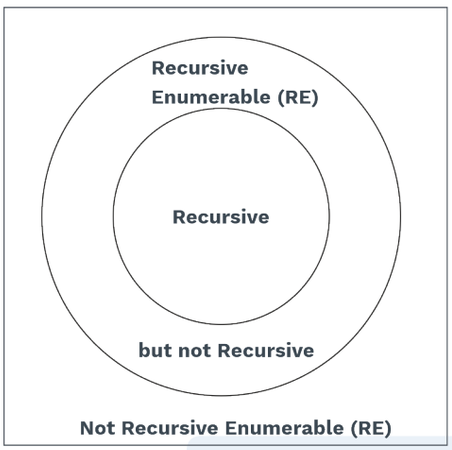


Fig. 6.4 Relationship between recursive and RE and not RE languages

Note:

The Recursive languages are closed under complement.

If L is language that is Recursive, so complement of L ($\overline{L}$) is also recursive:
Proof:

Let,

L = L(M) for some Turing Machine (M) that always halts. Now, we will construct another Turing Machine (M_1) such that:

$$\overline{L} = L(M_1)$$

where M_1 behaves just like M, we will have to modify M to create M_1:

1) All accepting states of M are changed as non-accepting states of M_1 with no transition. It means M_1 will halt without accepting.
2) M_1 has a new accepting state R; there is no transition from R.
3) For each combination of a non-accepting state of the Turing Machine (M) and a tape symbol of M such that M has no transition, add transitions to R.

M is guaranteed to halt. So, we can say M_1 will also halt. M_1 accept only those strings that M will never accept.

So, M_1 accepts the

complement of $L(\overline{L})$.

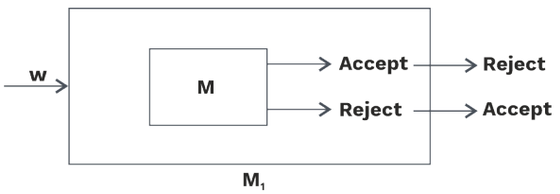


Fig. 6.5 Construction Of A Tm Accepting The Compliment Of Recursive Language.

If both the language L and its complement are Recursive Enumerable (RE), then language L is Recursive.

Proof as consider:

M_1 and M_2 are Turing Machines which are simulated in parallel by a Turing Machine (M).
Let,

$$L = L(M_1) \text{ and } \overline{L} = L(M_2)$$

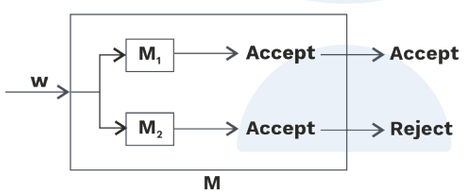


Fig. 6.6 Construction Of A Turing Maschine Accepting A Compliment Of A Recurisve Language

We can make Turing Machine (M) a two-tape Turing Machine and then it is converted into a single tape Turing Machine for simplicity.
Tape one of the Turing machines (M) simulate the tape of M_1, while tape two of the Turing machine (M) simulate the tape of M_2.
If

$$\text{Input } w \in L(M) \rightarrow M_1 \text{ will accept} \rightarrow M \text{ accepts and halts}$$

$$\text{Input } w \notin L(M) \rightarrow \overline{L} \rightarrow M_2 \text{ will accept} \rightarrow M \text{ halts without accepting}$$

Thus, for all inputs, M halts, and L(M) is the same as L. Since M always halts, and L(M) = L. So, we can say that L is Recursive.

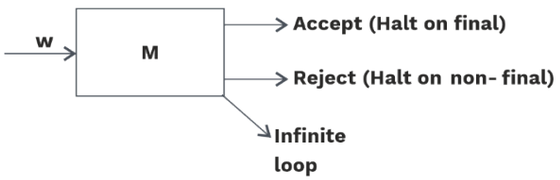
Turing machine halting problem:

"Does a Turing Machine (M) halt on input w?"

OR

"Is $w \in L(M)$?"

We know that Turing Machine has three possibilities.



If $w \in L(M)$. So, it will halt on accepting states.

If $w \notin L(M)$. So, it may halt on the non-final state, or it can get into an infinite loop.

Halt: The program/algorithm on certain input will halt on accepting state or halt on the non-final state, but it would never go into an infinite loop.

"Can we design an algorithm that tells whether the given program will halt or not?"

"The Answer will be NO". Because we cannot design any generalized algorithm which can surely say that a given program will halt or not."

Only a single way we have, just run the program and check whether it will halt or not.

So,

"Halting problem is an undecidable problem because we cannot have any algorithm that tells whether the given program will halt or not in a generalized way."

Table for decidable and undecidable problems:

Problems	FA	DPDA	PDA	LBA/HTM	TM
Halting	D	D	D	D	UD
Membership	D	D	D	D	UD
Emptiness	D	D	D	UD	UD
Finiteness	D	D	D	UD	UD
Totality	D	D	UD	UD	UD
Equivalence	D	D	UD	UD	UD
Disjoint	D	UD	UD	UD	UD
Set containment	D	UD	UD	UD	UD

Table 6.1

D $\Rightarrow$ Decidable

UD $\Rightarrow$ Undecidable

Note:

- For Regular Languages, all the given problems are decidable.
- For Recursive Enumerable Language (REL), all given problems are undecidable.

Rice theorem part-I:

- Any non-trivial property of Recursively Enumerable Language (REL) is undecidable.
- It is a negativity test. It does not prove if the language is Recursive or Recursive Enumerable.
- It Proves If a language is not recursive, it may or may not be recursively enumerable.

Definitions

Property : A property of language is simply a set of languages. We say 'L' satisfies the property 'P' is $L \in P$.

Examples:

- {<M>|M has 15 states}

This is a Property of Recursive Enumerable Language (REL) but not a language.

- {<M>|L(M) is infinite}
- {<M>|L(M) is regular.}

Trivial and non-trivial properties:

We know that set of all Turing Machines a (string of 0's and 1's) are countably infinite.

∴ Recursive Enumerable (RE) set is countably infinite like Recursive Enumerable Language (REL) set (RE_1, RE_2, RE_3, ...}

Trivial property:

A property is trivial if it is satisfied by all Recursive Enumerable Languages or it is not satisfied by any Recursive Enumerable Language (REL).

- If all the elements of the Recursive Enumerable Language (REL) set have the property → Trivial property

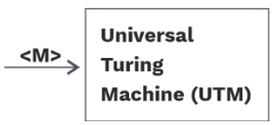
- If all the elements of the Recursive Enumerable Language (REL) set do not have the property → Trivial property

Non-trivial property:

A property is non-trivial if it is satisfied by some Recursive Enumerable Languages (REL) and are not satisfied by others.

- If at least one Recursive enumerable language (RE_i) has the property and one Recursive enumerable language (RE_i) does not have the property, then the property is said to be non-trivial.

Example:



$L = \{<M> \mid L(M) \text{ is infinite}\}$

Here, 'M' is the arbitrary Turing Machine Code which acts as an input to the Universal Turing Machine (UTM).

Let's find a Recursive Enumerable Language (RE) that has the property:

$\{L(RE_i) \text{ is infinite}\}$

RE_i_Yes = {1, 11, 111, ...}.

Turing Machine accepts languages:

$$L = \{W \mid W \in 1^*\}$$

Let's find a RE_j that does not have the property:

$\{L(RE_j) \text{ is finite}\}$

RE_j_NO = {1},

Turing Machine accepts language,

$$L = \{W \mid W \in 1\}$$

∴ Hence, we can find RE_i_YES (RE_i that has the property) and RE_i_NO(RE_i that does not have the property).

Hence, it is a Non-Trivial property of Recursively Enumerable Language. So, by Rice Theorem, we can say that the language is undecidable.

Rice theorem part-II:

Any Non-Monotonic property of Recursively Enumerable Language is not even semi-decidable.

Note:

- Recursive $\Rightarrow$ Decidable
- Semi decidable $\Rightarrow$ Recursive Enumerable but not REC.

This theorem is used for proving if the language is not semi-decidable (Not Recursive Enumerable).

Let's discuss the non-monotonic properties.

A property of recursively enumerable language is non-monotonic if any set (RE_i) having the property is PROPER SUBSET of any set (RE_j) that does not have the property.

Example:

where <M> is the arbitrary Turing Machine code.

Language of the Universal Turing Machine (UTM) can be defined as:

L(Universal Turing Machine) = {M | L(M) has at most 10 strings}

Let's see

If we can find Recursive Enumerable Language (REL_i); from REL_Set that has the property.

REL_i_YES = $\{\phi\}$ (Turing Machine accepts language $\Rightarrow$ L = $\{\phi\}$)

Can we find REL_j from REL_Set that does not have the property?

"Answer is YES"

REL_j_NO = {0, 1, 00, 000, 1111, ...0011,...}

(Turing Machine accepts language L $\Rightarrow$ $L = \left\{\Sigma^+\right\}, \Sigma = \{0,1\}$)

We can observe now,

$$REL_i \_ YES \subset REL_j \_ NO$$

So, it is a non-monotonic property.

Hence, By Rice Theorem, the language is not even semi-decidable.

Example:

L = {<M>|L(M) is infinite}

This is monotonic property since an infinite set can never be a subset of a finite set. So, we cannot apply Rice Theorem Part-II here.

We can use reduction to prove language is not Recursive Enumerable (RE).

It proves RICE Theorem Part-II is not the necessary condition for language to be NOT Recursive Enumerable (RE). It is a sufficient condition.

Reducibility:

Let us have two problems, A and B. If we have an algorithm to convert instance of problem A to instance of problem B that have the same answer, then we say

A reduces to B

or

A is polynomially reduced to B

or

A is many to one reducible to B

or

$A \leq B$

or

$A \propto_p B$

or

$A \leq_m B$

or

B is at least as hard as A.

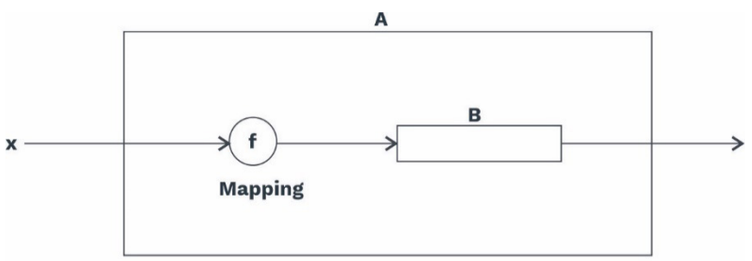


Fig. 6.7 A Is Reducible To B With The Help Of Mapping

Let, $L_1 \leq L_2$

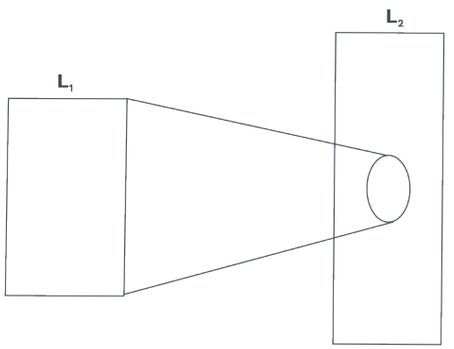


Fig. 6.8 Every Member Of L_1 Is Also Mapped To Some Member Of L_2

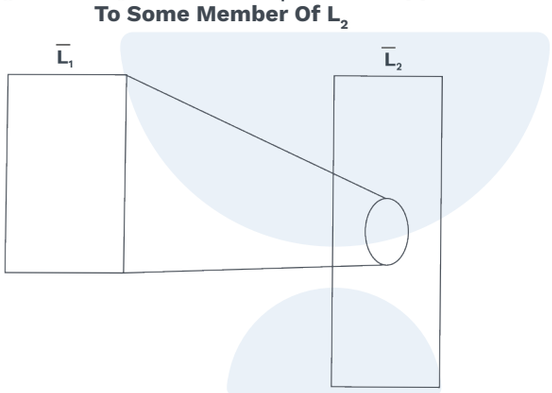


Fig. 6.9 Every Instance Of L_1 Is Mapped To Some Instance Of L,

Grey Matter Alert!

Let L_1 reducible to L_2 ($L_1 \leq L_2$)

- If L_2 is decidable, then L_1 is decidable.
- If L_1 is undecidable, then L_2 is undecidable.
- If L_2 is Recursive Enumerable (RE), then L_1 is Recursive Enumerable (RE).
- If L_1 is not Recursive Enumerable (RE), then L_2 is not Recursive Enumerable (RE).
- If $L_1 = \theta(n^2)$, then $L_2 = \Omega(n^2)$.
- If $L_2 = \theta(n^2)$, then $L_1 = O(n^2)$.

Note:

- If $A \leq B$ and $B \leq C$, then A is also reducible to C $(A \leq C)$.
- If $A \leq B$ and $C \leq B$, then
  - If B is decidable, then A and C both are decidable.
  - If A is decidable, then we cannot answer about B and C.
- If $A \leq B$ and $B \leq C$, then
  - If C is decidable, then B is decidable as well as A is decidable.
  - If B is decidable, then A is also decidable but we cannot answer about C.
  - If A is undecidable, then B and C are also undecidable.
  - If B is undecidable, then C is also undecidable.

Previous Years' Question

Choose the correct alternatives (More than one may be correct).
It is undecidable whether:

a) An arbitrary Turing machine halts after 100 steps.
b) A Turing machine prints a specific letter.
c) A Turing machine computes the products of two number.
d) None of the above.

Sol: b)

Previous Years' Question

Which of the following statements is false?

a) The Halting problem of Turing machines is undecidable
b) Determining whether a context-free grammar is ambiguous is undecidable.
c) Given two arbitrary context-free grammars G_1 and G_2 it is undecidable whether $L(G_1) = L(G_2)$
d) Given two regular grammar G_1 and G_2 it is undecidable whether $L(G_1) = L(G_2)$

Sol: d)

Previous Years' Question

Which one of the following is not decidable?

a) Given a Turing machine M, a string s and an integer k, M accepts s within k steps.

b) Equivalence of two given Turing machines.

c) Language accepted by a given finite state machine is not empty.

d) Language generated by context free grammar is non-empty.

Sol: b) **(GATE - 1997)**

Previous Years' Question

Consider the following decision problems:

(P_1) : Does a given finite state machine accept a given string?

(P_2) : Does a given context free grammar generate an infinite number of strings?

Which of the following statements is true?

a) Both (P_1) and (P_2) are decidable.

b) Neither (P_1) nor (P_2) are decidable.

c) Only (P_1) is decidable.

d) Only (P_2) is decidable.

Sol: a) **(GATE - 2000)**

Previous Years' Question

Consider the following problem X.

Given a Turing machine M over the input alphabet Σ, any state q of M and a word $w \in \Sigma$, does the computation of M on w visit the state of q?

Which of the following statements about X is correct?

a) X is decidable

b) X is undecidable but partially decidable.

c) is undecidable and not even partially decidable.

d) X is not a decision problem.

Sol: b) **(GATE - 2001)**

SOLVED EXAMPLES

Q1 **Membership problem is decidable on (More than one maybe correct option):**
a) Regular language
b) Context-free language
c) Context-sensitive language
d) Deterministic context-free language.

Sol: **a), b), c) & d)**

Explanation:
Membership problem is decidable for all languages except recursively enumerable language because all the machine always halts except Turing machine.

Q2 **Consider the following statement:**
S1: Recursive language is also called decidable language.
S2: Recursively Enumerable languages are also called as semi-decidable language.
S3: Partially decidable and semi-decidable both are the same.
Number of correct statement(s) is/are_____.

Sol: **3 to 3**

Explanation:
All the statements is correct.
Recursive language is called decidable language because, for the recursive language, we have halting Turing Machines.
For recursively enumerable language, we have a Turing machine and Turing machine may or may not halt, but TM always halts for string belongs to the language.
Hence, Recursively enumerable languages are semi-decidable.
Partially decidable and semi decidable means the same only.

Q3 **CFL are decidable on:**
a) Membership problem only **b) Emptiness problem only**
c) Finiteness Problem only **d) All the above**

Sol: **d)**

CFL are decidable on membership problem, emptiness problem, finiteness problem because we have analgorithm for all the problems.
Push down automata or CYK is analgorithm for membership problems.
For Emptiness problem algorithm is the simplification of CFG.
The Dependency tree lets us know whether the grammar is finiteness or not.
Hence, d) is the correct option.

Chapter Summary

- **Decision problem:**
  A decision is a set of related statements, each of which must be either true or false.
  i) For decidable problems, Halting Turing Machine (HTM) exists.
  ii) For undecidable problem, Halting Turing Machine (HTM) does not exist, but Turing Machine (TM) may exist.
  iii) If a problem is not recursive enumerable (RE), then no Turing Machine exists for that problem.

- **Language:**
  The set of all strings that can be generated from the grammar.

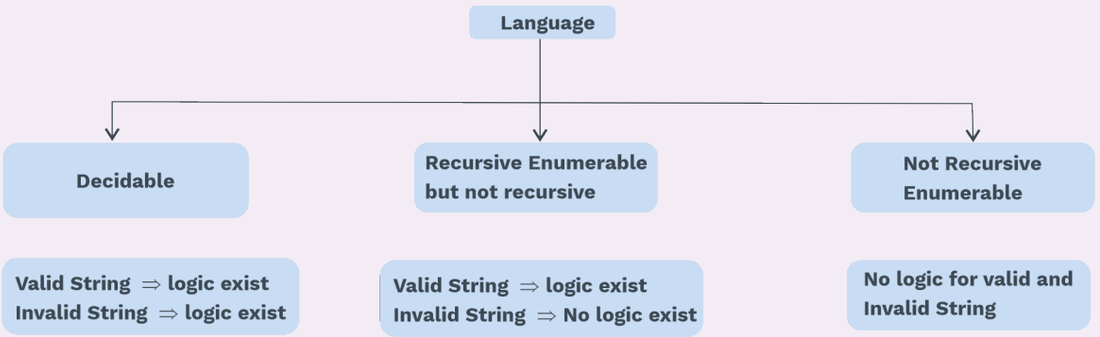


Fig. 6.10

- **Encoding a Turing Machine (TM)**
  Each Turing Machine with input alphabet {0, 1} may be thought of as a binary string.

- **The diagonalization language (L_d):**
  It is the set of strings (w_i) such that w_i is not in $L(M_i)$.

$$L_d = \{w_i \in (0+1)^+ \mid w_i \notin L(M_i)\}$$

- **An undecidable problem that is recursive enumerable (RE):**
  We have already seen that there is no Turing machine for Diagonal Language (L_d) to accept it.

- **Recursive languages:**
  A recursive language L If L = L(M) for some Turing Machine (M) such that:
  1) $w \in L \Rightarrow$ Halt and accept
  2) $w \notin L \Rightarrow$ Halt and reject

- **Compliment of recursive and recursive enumerable language (RE):**
  If a language L is Recursive Enumerable but the complement of recursive enumerable language $(\overline{L})$ is not Recursive Enumerable (RE), then we can say L can not be Recursive.

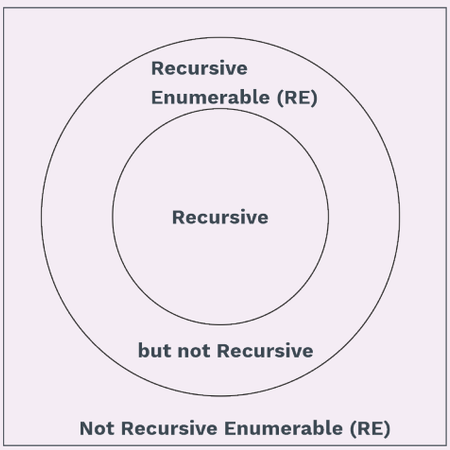
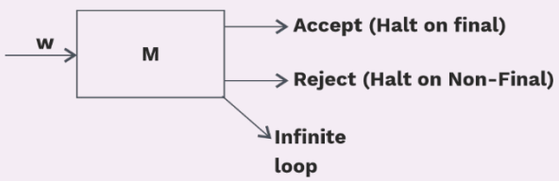


Fig. 6.11 Relationship between Recursive and Recursive Enumerable (RE)and not Recursive Enumerable (RE) language.

- **T uring machine halting problem:**
  "Does a Turing Machine (M) halt on input w?"
  OR
  "Is $w \in L(M)$?"
  We know that Turing Machine has three possibilities.



  If $w \in L(M)$. So, it will halt on accepting states.
  If $w \notin L(M)$. So, it may halt on the reject state, or it can get into an infinite loop.

- **Rice theorem part-I:**
  Any non-trivial property of Recursively Enumerable Language (REL) is undecidable.

- **Trivial property:**
  A property is trivial if either it is satisfied by all Recursive Enumerable Languages or it is not satisfied by any Recursive Enumerable Language (REL).

- **Non-trivial property:**
  A property is non-trivial if it is satisfied by some Recursive Enumerable Languages (REL) and are not satisfied by others.
- **Rice theorem part-II:**
  Any Non-Monotonic property of Recursively Enumerable Language is not even Semi-decidable.
- **Reducibility:**
  Let us have two problems, A and B. If we have an algorithm to convert an instance of problem A to instance of problem B that has the same answer, then we say A reduce to B