6. UNDECIDIBALITY

Theory of computation

Lecture Slides by:
Er. Shiva Ram Dam, PEC

Syllabus:

6.1 Church's Thesis

6.2 Halting Problem

6.3 Universal TM

6.4 Undecidable problem about TM

6.5 Recursive Function Theory

6.6 Properties of Recursive and Recursive Enumerable Language

6.1 Decidable and Undecidable Problem

Decidable problem:

- A problem is decidable if we can construct a TM that:
  - Will halt in finite amount of time for every input, and
  - Give answer as YES or NO.
- A decidable problem has an algorithm to determine to answer for a given input.
- Examples of decidable problems:
  1. Equivalence of two regular language
  2. Finiteness of regular language
  3. Emptiness of CFL

6.1 Decidable and Undecidable Problem

Undecidable problem:

- A problem is undecidable if **there is no TM** that:
  - Will halt in finite amount of time for every input, and
  - Give answer as YES or NO.
- An undecidable problem has no algorithm to determine to answer for a given input.
- Examples of undecidable problems:
  - Ambiguity of CFL
  - Equivalence of two CFL
- Two popular undecidable problems are :
  - The Halting problem,
  - PCP (Post Correspondence Problem)

6.2 The Halting Problem

- Basically, halting means terminating.
- Halting means that the program on certain input:
  - will accept it and halt, or
  - reject it and halt, and never go into an infinite loop.
- Halting problem is undecidable.
- It asks- “Is it possible to tell whether a given machine will halt for some given input?”
  - The answer is NO.
- We cannot design a generalized algorithm which can appropriately say the given a program, the machine will ever halt or not.

Halting problem: Example

- Input:

  A TM and input string w

- Problem:

  Does the TM finish computing of the string w in a finite no. of steps?

- Proof:

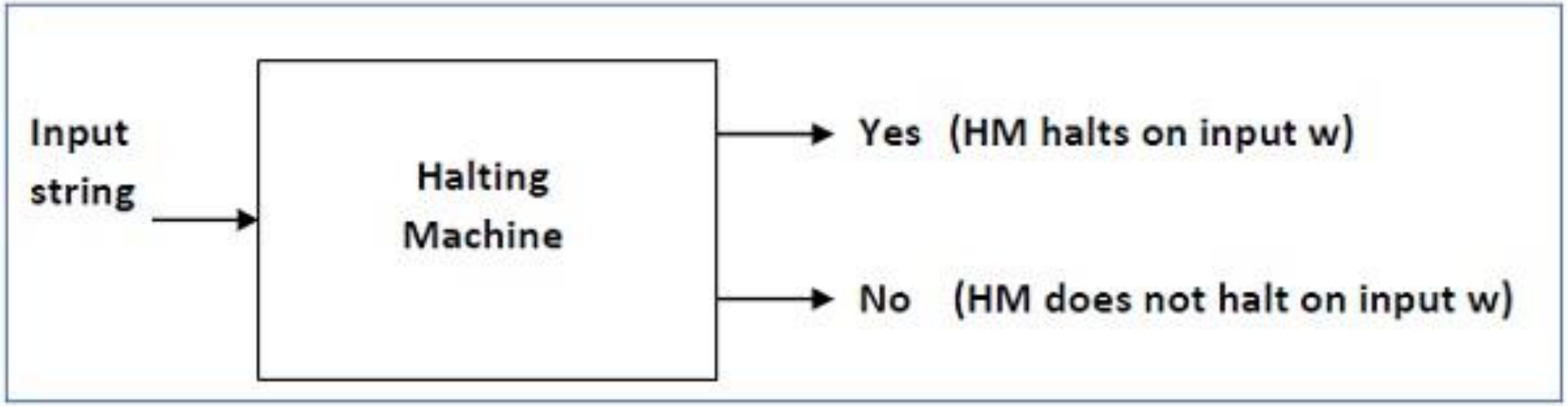
  Assume TM exits to solve this problem and then we will show it is contradicting itself.

Halting problem: Example (contd.)

Let the Machine be called HALTING MACHINE (HM).

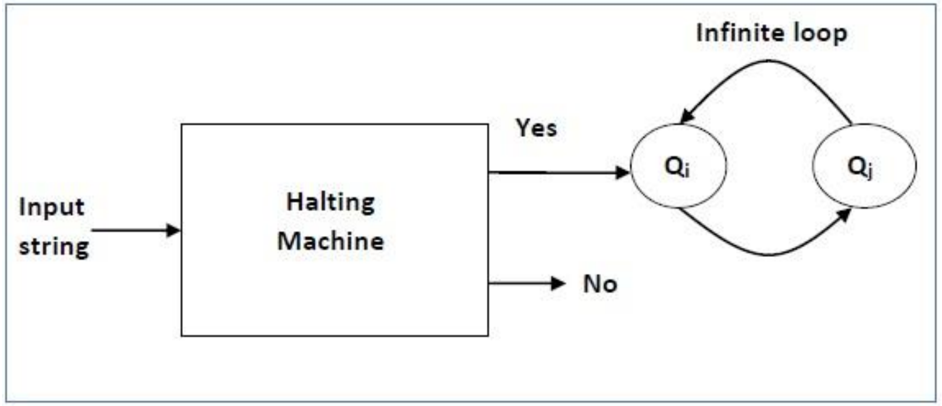
Let it produce a YES or NO in a finite amount of time.

Block diagram:



Halting problem: Example (contd.)

- Now lets design an INVERTED HALTING MACHINE
  - If HM returns YES, then loop forever
  - If HM returns NO, then halt
- Block diagram:



- Here,
  - If HM halts on given input, then it loops forever.
  - But if HM doesn’t halt, it returns NO and hence halts.
- Hence, by contradiction, the Halting Problem is undecidable.

6.3 Church's Thesis

- Church Thesis submitted by Alonzo Church (1936)
- Church's Thesis considers the TM as Ultimate Calculating Mechanism
- States that:

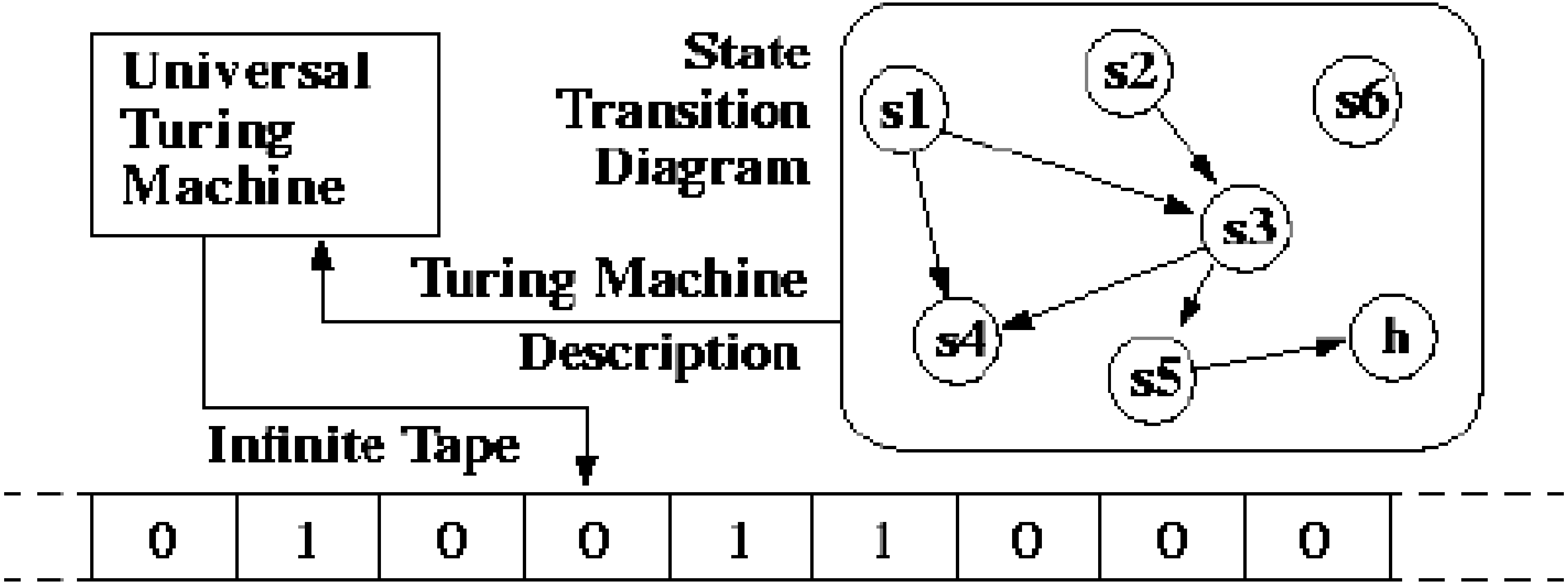
  No computational procedure will be considered an algorithm unless it can be represented by a TM.

- Tied together the idea of Recursive Functions and Computable Functions.
- Church Thesis , however, cannot be a theorem.

6.4 Universal TM

- Introduced by Alan Turing in 1936-1937
- A UTM can simulate the behavior of an arbitrary TM over any set of input symbols.
- Reads both the description of machine and the input from its own tape.
- Thus it is possible to create a single machine that can be used to compute any computable sequence.

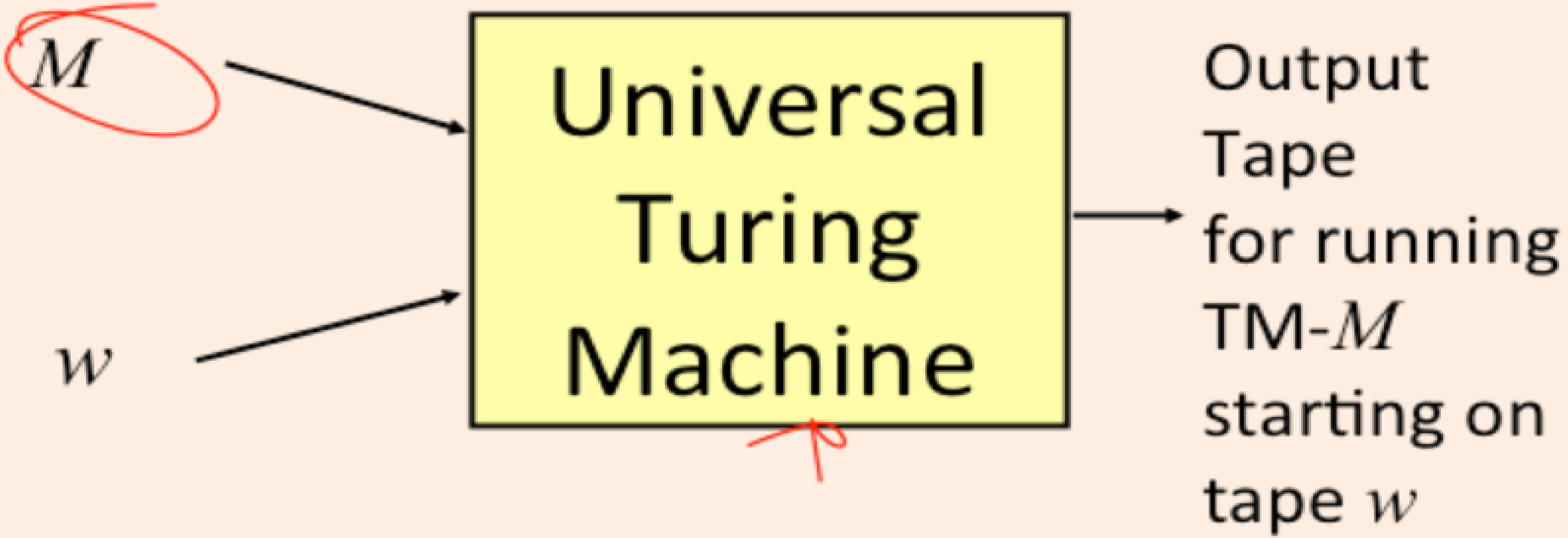
6.4 Universal TM (contd.)



6.4 Universal TM (contd.)

Input: < Description of some TM *M*, *w* >

Output: result of running *M* on *w*



6.5 Undecidable Problems about TM

The problems for which no algorithms exist are called Undecidable or Unsolvable.

One famous Undecidable Problem is the HALTING problem

Some undecidable problems about TM are:

1. Given Turing machine M and input string w for a TM, does M halt on input w?
2. Given M:
   - Does M halt on empty tape?
   - Is there any string at all on which M halts?
   - Does M halt on every input string?
   - Is the language the M semidecides regular? Is it Context free? Is it recursive?
3. Given two TMs (M1 and M2), do they halt on same input string?

6.6 Encoding of TM

- Is the process of formulating a notation system where we can encode both an arbitrary Turing machine T1 and an input string x over an arbitrary alphabet as strings e(T1) and e(x) over some fixed alphabet.
- The Encoding must not destroy any original information.
- For encoding, we use alphabets {0,1} although TM may have much larger alphabet.

Steps for Encoding

1. Start by assigning positive integer to each state, each tape symbol and each of three directions in TM that we want to encode.
2. Represent a state or a symbol by a string of 0's of appropriate length. Here 1's are used as separator.
3. For transition rule, use encoding function(S).

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

   encoded as:

   $S(q_i)\ 1\ S(a_j)\ 1\ S(q_k)\ 1\ S(a_l)\ 1\ S(D_m)$ .. say m1
4. Separate the entire transition rules by pair of 1's.

   ie. $m_1\ 11\ m_2\ 11\ m_3\ 11\ \dots\ m_n$

5. Now code for TM and i/p string x will be formed by separating them by three consecutive 1's.

   ie. e(TM) 111 e(x)

Encoding of TM: Example

Example:

Encode a TM, $T = (Q, \Sigma, \tau, \delta, q_1, F, B)$ where

$Q = \{q_1, q_2, q_3\}$, $\Sigma = \{a, b\}$, $\tau = \{a, b, B\}$, $q_1 = \{q_1\}$,

$F = \{q_3\}$, $B = \{B\}$, and δ is:-

$\delta(q_1, b) = (q_3, a, R)$

$\delta(q_3, a) = (q_1, b, R)$

$\delta(q_3, b) = (q_2, a, R)$

$\delta(q_3, B) = (q_3, b, L)$ and input string $x = ab$

Sol^n: Here,

Step 1:

~~Let~~ Here, $\Sigma = \{a, b\}$

Let $a_1 = a$

$a_2 = b$

Encoding of TM: Example

Step 2:

Let us represent states and symbols by a string of 0's of appropriate length.

$s(B) = 0$

$s(a_i) = 0^{i+1}$ for each $a_i \in \Sigma$

$s(q_i) = 0^{i+2}$ for each $q_i \in Q$

$s(N) = 0$

$s(L) = 00$

$s(R) = 000$

Using encoding function s as defined above:

$s(q_1) = 000$ $\quad s(a_1) = s(a) = 00$ $\quad s(N) = 0$

$s(q_2) = 0000$ $\quad s(a_2) = s(b) = 000$ $\quad s(L) = 00$

$s(q_3) = 00000$ $\quad s(B) = 0$ $\quad s(R) = 000$

5

Encoding of TM: Example

Step 3: Using encoding function for transition rules

$e(m_1) = S(q_1)\ 1\ S(b)\ 1\ S(q_3)\ 1\ S(a)\ 1\ S(R)$

$= 000\ 1\ 000\ 1\ 00000\ 1\ 00\ 1\ 000$

$e(m_2) = S(q_3)\ 1\ S(a)\ 1\ S(q_1)\ 1\ S(b)\ 1\ S(R)$

$= 00000\ 1\ 00\ 1\ 000\ 1\ 000\ 1\ 000$

$e(m_3) = S(q_3)\ 1\ S(b)\ 1\ S(q_2)\ 1\ S(a)\ 1\ S(R)$

$= 00000\ 1\ 000\ 1\ 0000\ 1\ 00\ 1\ 000$

$e(m_4) = S(q_3)\ 1\ S(B)\ 1\ S(q_3)\ 1\ S(b)\ 1\ S(L)$

$= 00000\ 1\ 0\ 1\ 00000\ 1\ 000\ 1\ 00$

Encoding of TM: Example

Step 4: Code for TM, T is:

$e(T) = e(m_1)\ 11\ e(m_2)\ 11\ e(m_3)\ 11\ e(m_4)$

$= 0000\ 1\ 000\ 1\ 000000\ 1\ 00\ 1\ 000\ 11$

$000001\ 00\ 1\ 000\ 1\ 0000\ 1\ 000\ 11$

$00000\ 1\ 000\ 1\ 0000\ 1\ 00\ 1\ 000\ 11$

$00000\ 1\ 0\ 1\ 000000\ 1\ 000\ 1\ 00$

Step 5: Now for T and any input string x where $x = ab$,

Code will be: $e(T)\ 111\ e(x)$

Here,

$e(x) = s(a)\ 1\ s(b)$

$= 00\ 1\ 000$

Hence,

$e(T)\ 111\ e(x) = 00001\ 00001 \ldots\ldots 1000\ 111\ 001000$

6.7 Recursive and Recursively Enumerable Language

When a TM executes an input, there are four possible outcomes of execution. Then Tm:

1. Halts and accept the input
2. Halts and rejects the input
3. Never halts(fall into loop), or
4. Crash

Reference : https://www.geeksforgeeks.org/recursive-and-recursive-enumerable-languages-in-toc/

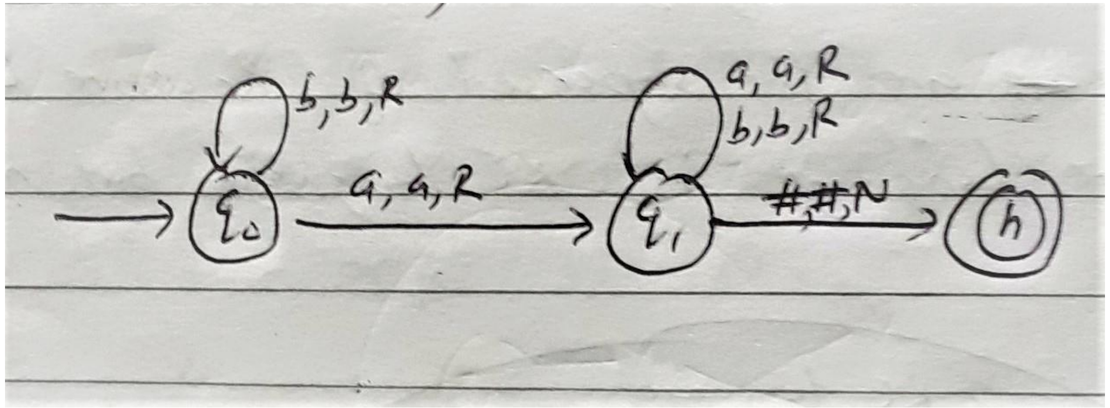
6.7.1 Recursive Enumerable Language (RE)

- Also called Type-0 language or TM Recognizable Language
- RE languages are those languages that can be accepted by TM.
- A language is RE if :
  - There exists a TM that accepts every string of the language, and
  - Does not accept strings that are not in the language.
- String that are **not in the language** may be **rejected** or may cause the TM to **go into an infinite loop.**
- RE language are superset of Recursive Language
- Every Recursive language is RE language, but not vice-versa.

Example: of RE language

Let L={w∈{a,b}* : w contains at least one ‘a’

Then we can design a TM for L as:



- This machine scans to the right to find one ‘a’.
- If no **‘a’** is found, it goes forever, never halting.
- It halts only if there is at least one ‘a’.

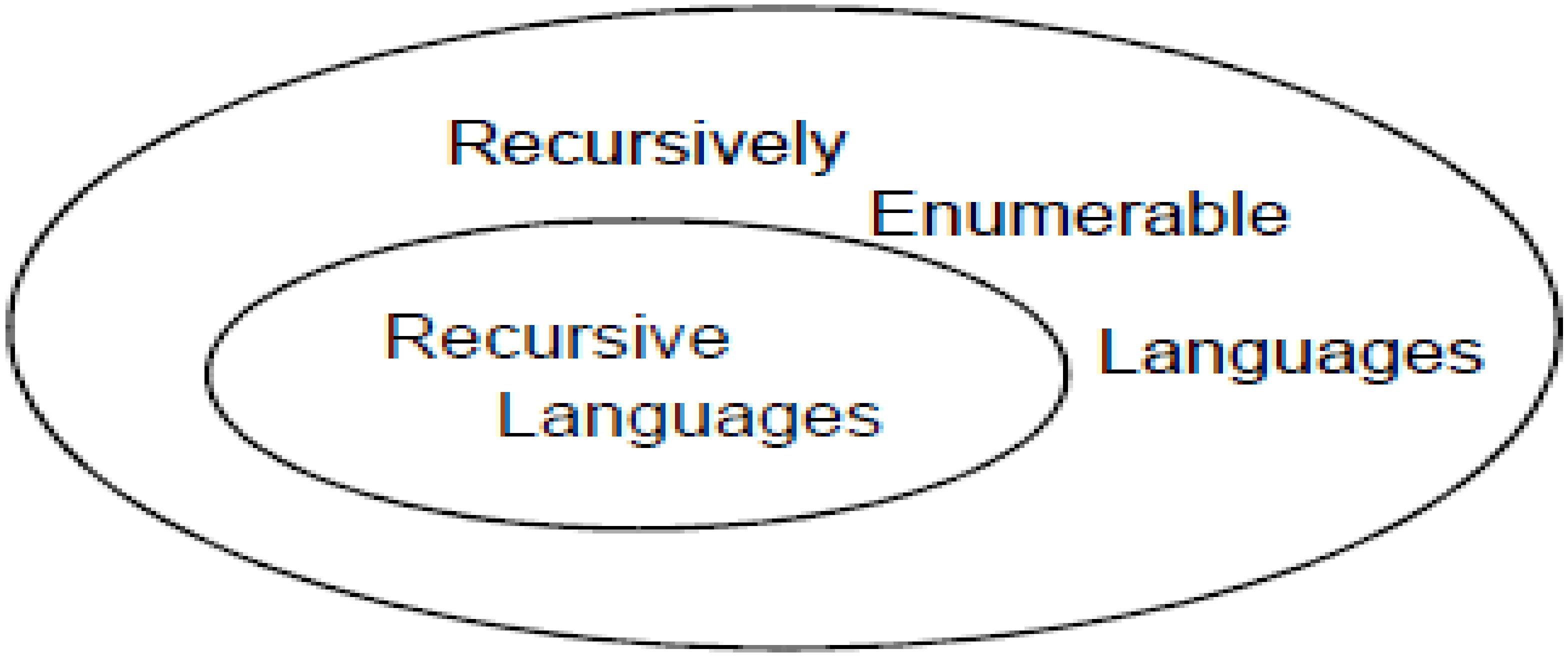
Therefore, given language is recursively enumerable.

6.7.2 Recursive Language (REC Language)

- A language is recursive if:
  - There exists a TM that accepts every sting of the language and
  - rejects every strings that are not in the language.
- A REC language is decidable by TM. It means it will **enter into final state for acceptable strings** and **into rejecting state for non-acceptable strings.**
- So, the TM will always halt in this case.
- A REC language is the subset of RE language.
- For eg:

  $L=\{a^n b^n c^n | n>=1\}$ is recursive language because we can construct a TM which will move to final state if the string is of the form $a^n b^n c^n$, else move to non-final state.

Relationship between RE and REC language



Relationship between RE and REC language

6.8 Turing Recognizable Language

- Can run forever without deciding
- A language L is Turing recognizable if there exists a Turing machine M such that for all srings **w**:
  - If $w \in L$, eventually M enter q_{accept}.
  - If $w \notin L$,either M enters q_{reject} or M never terminates.

6.9 Turing Decidable Language

- Always terminates
- A language L is Turing decidable if there exists a Turing machine M such that for all strings w:
  - If $w \in L$, M enter q_{accept}.
  - If $w \notin L$, M enters q_{reject}

6.10 Theorems Proof:
If a language is recursive then it is recursively enumerable.

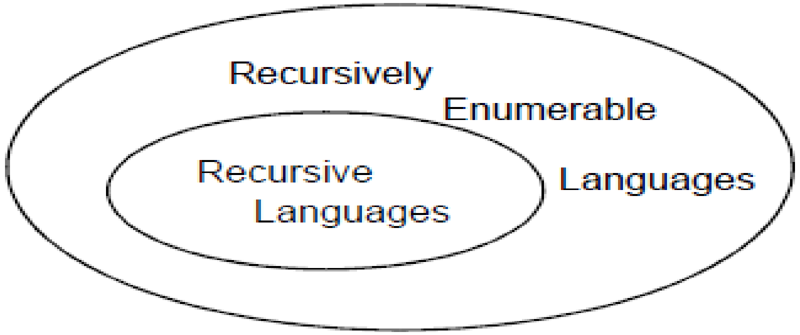
- A language is recursive if :

  TM accepts every stings of the language then **enters into final state**, and rejects every strings that are not in the language, then **enters into rejecting state.**

- A language L is recursively enumerable if:

  TM accepts every strings of the language, then **enters into final state**, and rejects every string that are not in the language, then **may enter into rejecting state or may loop forever.**

Hence, we can say that a recursive language is also a recursively enumerable. This is shown with a relationship diagram.



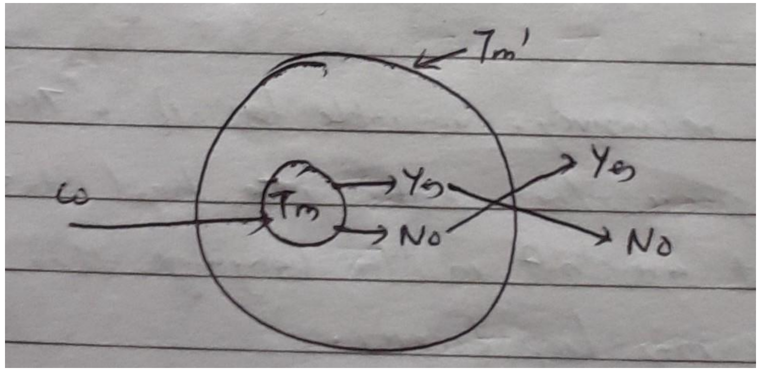
6.11 Properties of Recursive and Recursive Enumerable Language

1. The complement of a Recursive Language is recursive.
2. The union of two Recursive language is recursive.
3. The union of two RE language is recursively enumerable
4. If a language L and its complement L’ are both recursively enumerable, then L (and hence L’) is recursive.
5. If L is recursive language then $\sum^*$ - L is recursive.

6.12 Theorem Proof:

1. The complement of a Recursive Language is recursive

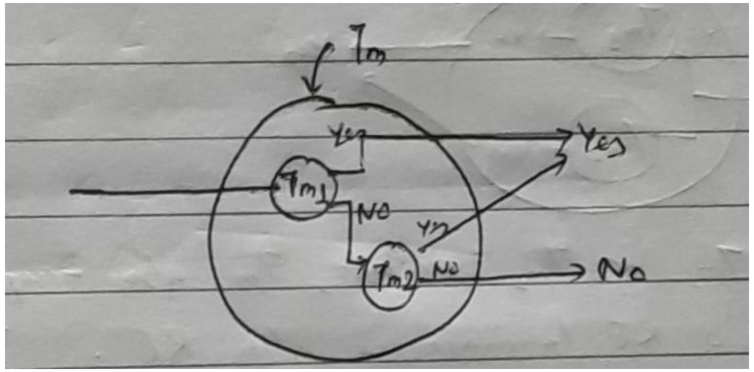
- Let L be recursive language.
- Tm be Turing machine that halts on all inputs and accepts L
- Let us construct a Turing machine Tm’ from Tm so that if Tm enters a final state on input w, then Tm’ halts without accepting.
- If Tm halts without accepting, Tm’ enters a final state.



- Since one of these two events occurs, Tm’ is an algorithm.
- So, clearly, T(Tm’) is the complement of L and thus the complement of L is recursive language.

2. The union of two recursive language is recursive.

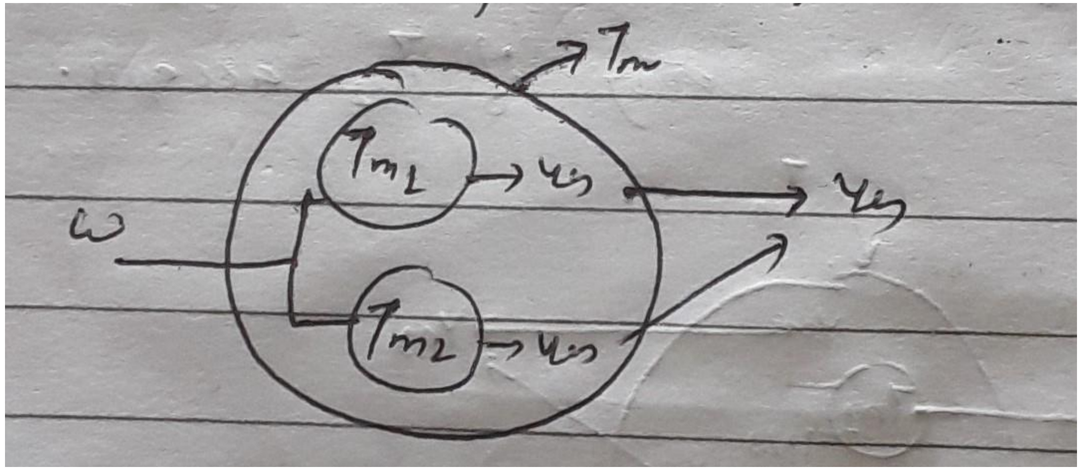
- Let L1 and L2 be two recursive languages accepted by Tm1 and Tm2 respectively.
- Let us construct a turing machine Tm that first simulates Tm1 and then Tm2.
- If Tm1 accepts, then Tm accepts and halts.
- If Tm1 rejects, then Tm simulates Tm2 and accepts iff Tm2 accepts.



- Here, since both Tm1 and Tm2 are algorithm, and Tm is guaranteed to halt in either the case.
- Hence, clearly, Tm accepts L1 U L2.
- Thus, the union of two recursive language is also recursive.

3. The union of two recursively enumerable language is recursively enumerable.

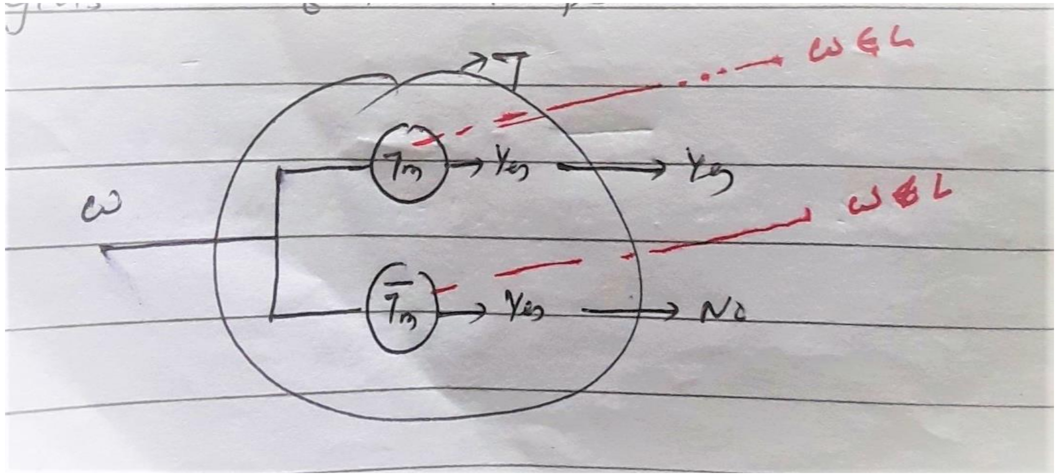
- Let L1 and L2 be recursively enumerable language and their enumerative TM are Tm1 and Tm2 respectively.
- Let us construct a Turing machine Tm which can simulate Tm1 and Tm2 simultaneously on separate tape.



- Here, if either Tm1 or Tm2 accepts, Tm also accepts.
- Thus, union of two RE languages is also RE.

4. If a language L and its complement L' are both recursively enumerable, then L (and hence L') is recursive.

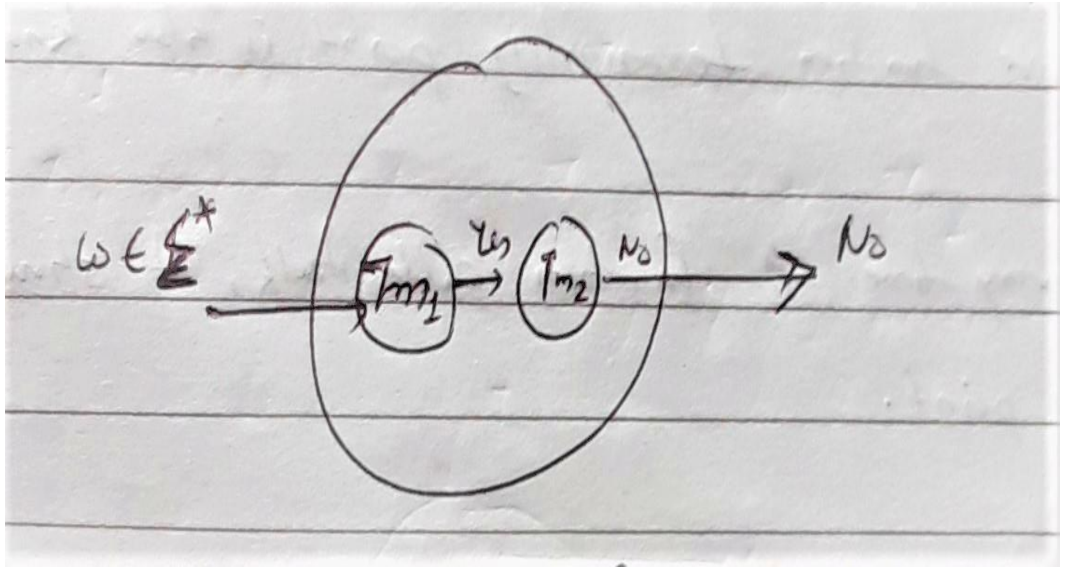
- Let Tm and Tm' accept L and L' respectively.
- Let us construct a Turing machine T which simulate Tm and Tm' simultaneously.
- T accepts w if Tm accepts w, and
- T rejects w if Tm' accepts w



- Thus, T will always say either YES or NO, but never says both.
- Since T is algorithm that accepts L, it follows that L is recursive.

5. If L is recursive language then ∑* - L is recursive.

- The required Tm-complement can be represented as:



∑* - L means Recursive of L

- When a string w ∈ ∑* is given as input to Tm-complement, its control passes the string to Tm1 as input.
- As Tm1 decides the language L, therefore, for w ∈ L after a finite no. of moves, Tm1 outputs YES which is given as input to Tm2, which in turn returns NO.
- Similarly for w ∉ L , Tm2 return YES
- Hence, there exists a Tm-complement for ∑* - L
- So it is Turing decidable, that is recursive.

Also Refer:
https://www.geeksforgeeks.org/recursive-and-recursive-enumerable-languages-in-toc/

Other properties:

1. Intersection of two recursive language is also recursive.
2. Intersection of two recursive enumerable language is also recursive enumerable.

3. References videos:
    1. https://www.youtube.com/watch?v=_4asJqA2xTI
    2. https://www.youtube.com/watch?v=BwIRBVM_P0E
    3. https://www.youtube.com/watch?v=VhK0p3QSE4A

6.13 Recursive Function Theory

- Recursive function theory is a functional or declarative approach to computation.
- In this approach, computation is described in terms of **"what is to be accomplished"** instead of **"how to accomplish".**
- Recursive function theory begins with some very elementary functions that are intuitively effective.
- Then it provides a few methods for building more complicated functions from simpler functions.

Reference:
https://legacy.earlham.edu/~peters/courses/logsys/recursiv.htm
https://www.youtube.com/watch?v=7vgXBspFVh4

- **E.g.: Given the recursive function defined by:**

  f(1)=1

  f(2)=2

  f(n)=2f(n-1)+f(n-2)   for n>=3

  What is the value of f(5)?

The answer is 29

Computation is as:

n	value
1	f(1)=1
2	f(2)=2
3	f(3)=2f(n-1)+f(n-2) =2f(3-1)+f(3-2) =2f(2)+f(1) =2*2+1 =5
4	f(4)=2f(n-1)+f(n-2) =2f(4-1)+f(4-2) =2f(3)+f(2) =2*5+2 =12
5	f(5)=2f(n-1)+f(n-2) =2f(5-1)+f(5-2) =2f(4)+f(3) =2*12+5 =29

6.13.1 Initial Functions for Natural Numbers

- All the elementary functions are all functions of natural numbers.
- They make take zero as input, nut no negative number, and not any rational or irrational numbers.
- Let N={0,1,2,....} be a set of natural numbers, we have three initial function over N defined as below:

1. **Zero function**

   The Zero function returns zero regardless of its argument.

   Denoted by Z and defined as:

   Z(n) = 0 for ∀ n ∈ N

   Eg: Z(2) = 0

2. **Successor function**

   The Successor functions returns the successor of its arguments.

   Denoted by S and defined as:

   S(n) = n+1 for ∀ n ∈ N

   Eg: S(2) = 2 +1 = 3

3. **Projection function**

   Defined as $P_i^n(a_1,a_2,.....a_n)= a_i$ where $a_i \in N$ for i=1,2,3, …n and i<=n

   Projection function takes n arguments and returns their i^{th} argument.

   Eg: $P_2^3(7,8,9)=8$

Reference:
https://www.youtube.com/watch?v=_cswfIQg0Ss

16.13.2 The Building Operations

We can build more complex and interesting functions from the initial set using three methods:

1. Composition
2. Primitive recursion
3. Minimization

References:

- https://www.youtube.com/watch?v=twHp7IrPJEs
- https://www.youtube.com/watch?v=cjq0X-vfvYY
- https://www.youtube.com/watch?v=bFkU-qV2Ioo

6.13.3 Composition of function

- We can define a new function by the combination of two or more functions.
- Such defined functions are called **composition functions.**
- For e.g.: given S(n)=n+1

$$
\begin{aligned}
S(Z(a)) &= S(0) \\
&= 1
\end{aligned}
$$

$$
\begin{aligned}
S(S(Z(n))) &= S(S(0)) \\
&= S(1) \\
&= 2
\end{aligned}
$$

Reference:
https://www.youtube.com/watch?v=twHp7IrPJEs

6.13.4 Primitive Recursive Function

- A function is primitive if:
  - It is an initial function
  - It is obtained from recursion or composition of initial functions.
- The factorial function is derived by primitive recursion from the functions for multiplication and subtraction.
- Eg:

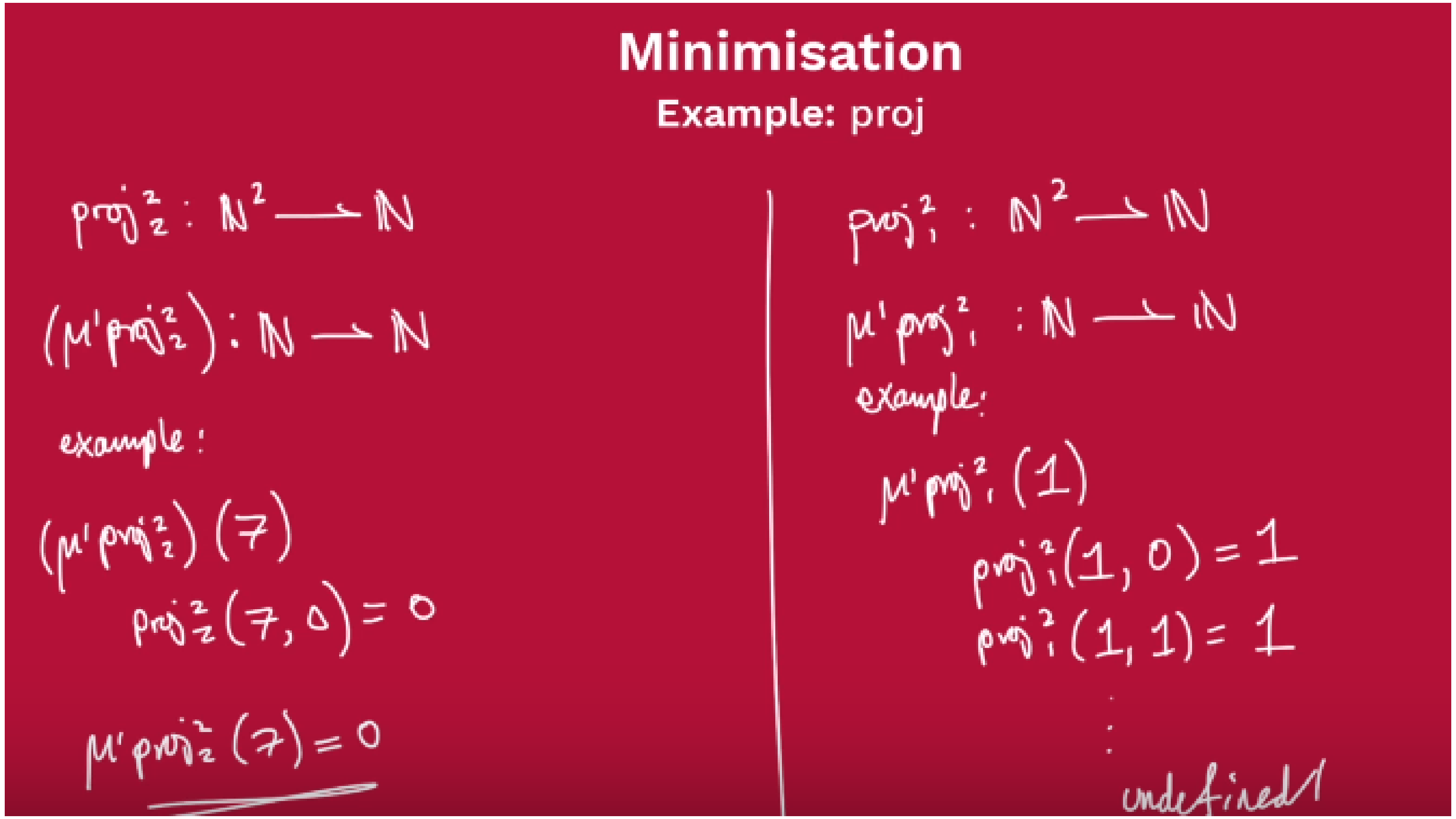
$$f(n) \begin{cases} f(n) = 1 & \text{when } n = 1 \\ f(n) = n(f(n-1)) & \text{when } n > 1 \end{cases}$$

Reference:

- https://www.youtube.com/watch?v=cjq0X-vfvYY
- https://people.cs.clemson.edu/~goddard/texts/theoryOfComputation/16.pdf

6.13.5 Minimisation

It provides the way to find the least value.



Reference: https://www.youtube.com/watch?v=bFkU-qV2Ioo

6.13.6 Recursive functions

- A function which calls itself directly or indirectly and terminates after infinite no. of steps is known as Recursive function.
- In recursive function, terminating point is also known as base point.
- Each and every time, the function calls itself, it should be nearer to the base point.
- Recursive function are built up from basic functions by some operations.

Examples of recursive definitions

$$\begin{cases} f_1(0) & \equiv 0 \\ f_1(x+1) & \equiv f_1(x) + (x+1) \end{cases}$$

$f_1(x) =$ sum of $0, 1, 2, \ldots, x$

$$\begin{cases} f_2(0) & \equiv 0 \\ f_2(1) & \equiv 1 \\ f_2(x+2) & \equiv f_2(x) + f_2(x+1) \end{cases}$$

$f_2(x) = x$th Fibonacci number

6.13.7 Partial Recursive Function

- A function **f**(a1, a2, ….an)computed by a TM is known as partial recursive function if **f** is defined for some but not all values of a_1, a_2, ….a_n.
- Let $f(a_1,a_2,a_3, \ldots,a_n)$ be a function and defined on function $g(b_1,b_2,b_3,\ldots b_m)$, then; **f** is partial function if some element of **f** is assigned to almost one element of function **g**.
- A partial function is recursive if:
  - It is an initial function over N, or
  - It is obtained by applying recursion or composition or minimization on initial function over N.

Refer:

- **Pandey A.K., An Introduction to Automata Theory and Formal Language, Page 280**
- **https://www.youtube.com/watch?v=_RlkwPCN4yQ**

6.13.8 Total Recursive Function

- A function is said to be total recursive function if it is defined for all of its arguments.
- Let $f(a_1,a_2,a_3, \ldots..,a_n)$ be a function and defined on function $g(b_1,b_2,b_3,\ldots..b_m)$, then; **f** is total function if every element of **f** is assigned to some unique element of function **g**.

Refer:

- https://www.youtube.com/watch?v=_RlkwPCN4yQ
- **Pandey A.K., An Introduction to Automata Theory and Formal Language, Page 280**

References:

- Pandey A.K., An Introduction to Automata Theory and Formal Language
- https://www.youtube.com/watch?v=0Q9qAM2htII
  https://www.youtube.com/watch?v=macM_MtS_w4
  https://www.youtube.com/watch?v=2PaOjhnyQ9o
  https://www.youtube.com/watch?v=NbrnomQkc2U
- https://www.youtube.com/watch?v=_RlkwPCN4yQ
- https://legacy.earlham.edu/~peters/courses/logsys/recursiv.htm
- https://www.youtube.com/watch?v=7vgXBspFVh4
- https://www.youtube.com/watch?v=twHp7IrPJEs
- https://www.youtube.com/watch?v=cjq0X-vfvYY
- https://www.youtube.com/watch?v=bFkU-qV2Ioo
- https://www.youtube.com/watch?v=yaDQrOUK-KY
- https://www.youtube.com/watch?v=_cswfIQg0Ss

End of chapter