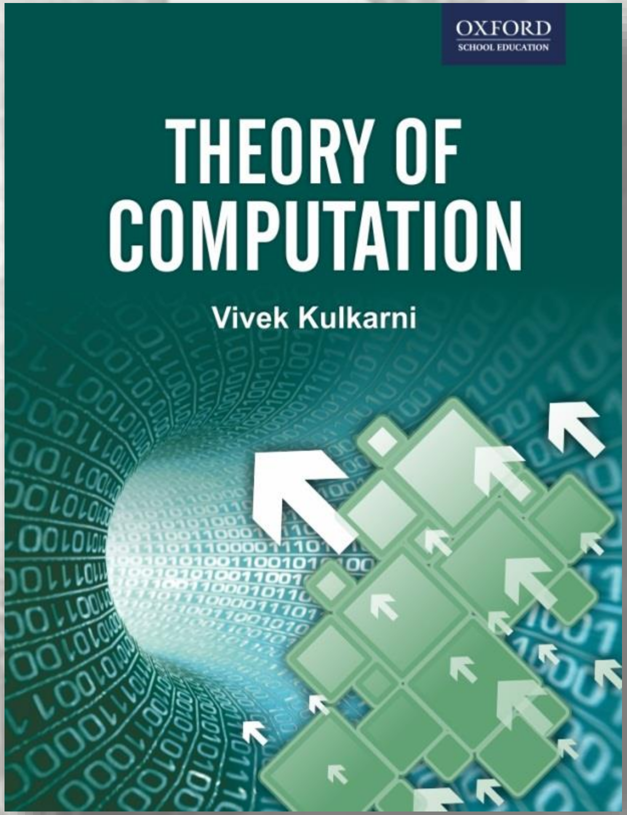
OXFORD
HIGHER EDUCATION

THEORY OF COMPUTATION

Vivek Kulkarni

Slides for Faculty Assistance



Chapter 4

Turing Machine

Author: Vivek Kulkarni

vivek_Kulkarni@yahoo.com

Outline

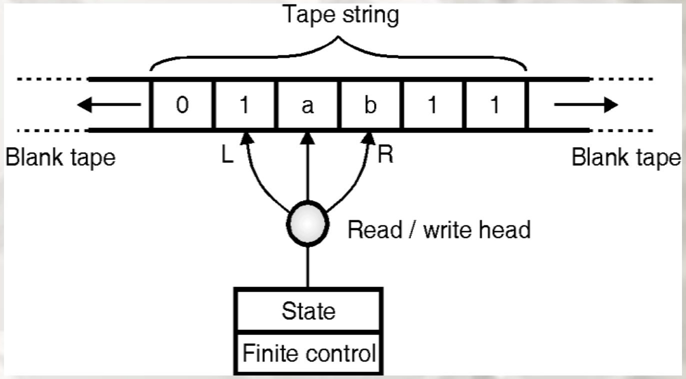
- Following topics are covered in the slides:
  - Formal model of Turing machine
  - Comparison of finite automata (FA) and TM on the basis of their relative computational powers
  - Recursively enumerable languages and recursive languages
  - Concept of composite TM and iterative TM
  - Multi-tape TM, multi-stack TM, and multi-track TM
  - Concept of universal Turing machine (UTM)
  - Halting problem and limits of computability
  - Church's Turing hypothesis
  - Solvable, semi-solvable, and unsolvable problems
  - Total and partial recursive functions
  - Post's correspondence problem (PCP)
  - Linear Bounded Automata (LBA)

Turing Machine

- Turing Machine (TM)
  - Complete algorithmic solution
  - Visualized as a head / tape assembly
  - Program based on only 3 actions :
    - Change in machine state
    - Change in direction of the head to move one cell at a time {left=L, right=R, no-movement=N}
    - Erase and write a new symbol on the tape cell
  - These 3 actions are coded into a functional matrix (δ) as quintuples of the form,

$$\delta: (Q \times \Gamma) \rightarrow (Q \times \Gamma \times \{L, R, N\})$$

Where, ‘Q’ is the finite set of states and ‘Γ’ is the finite set of tape symbols



TM Formal Definition

- A Turing machine (TM) is denoted by: $M = (Q, \Sigma, \Gamma, \delta, q_0, B, F)$,
  where,
  Q : Finite set of states
  Γ: Finite set of allowable tape symbols, including blank character
  Σ : The set of input symbols, which is a subset of Γ, excluding blank character
  B : A symbol for blank character
  q_0: Start (or initial) state $\in Q$
  F : Set of final states (or halt states) $\subseteq Q$
  δ : Functional matrix, such that $\delta: (Q \times \Gamma) \rightarrow (Q \times \Gamma \times \{L, R, N\})$

- In order to have a deterministic TM, there should be a unique triple (b, β, d) for every state α on some symbol a, for $\delta(\alpha, a)$ in the functional matrix.

Instantaneous Description

- Instantaneous description (ID) of a TM M is denoted by: '$\alpha_1\ q\ \alpha_2$', where q is the current state of the TM, i.e., $q \in Q$, and '$\alpha_1\ \alpha_2$' is the string in Γ^*
- For example, let '$a_0\ a_1\ \ldots\ a_{i-1}\ q\ \ a_i\ \ldots\ a_n$' be an ID of M. This ID indicates that the current state of the machine is q and the machine head is about to read symbol a_i from the tape.
- A transition is defined as moving from one ID to the other performing one quintuple. For example, consider the move $\delta\ (q, a_i) = (*, p, R)$ that can be shown as,

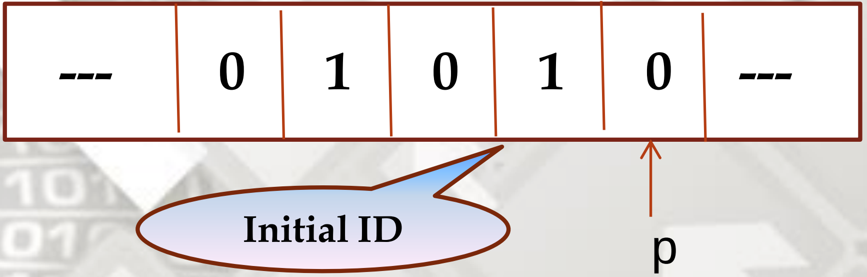
$$a_0\, a_1 \ldots a_{i-1}\, q\, a_i \ldots a_n \ \vdash_{M}\ a_0\, a_1 \ldots a_{i-1}\, {*}\, p\, a_{i+1} \ldots a_n$$

Here, a_i is replace by * and the machine has changed the state to p and moved one position to the right on tape.

TM Example

2's complement of 01010
=
10101 --- 1's complement
+ 1

10110



State p : Keep moving left till the leftmost '1' is found

Do the state transition to q once the leftmost '1' is read

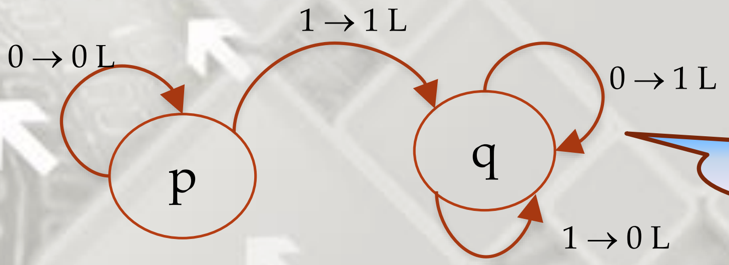
State q : While moving left replace every '1' by '0' and every '0' by '1'

δ	0	1
p	0pL	1qL
q	1qL	0qL

Functional Matrix

Quintuples

$\delta \quad (p, 0) = (0\ p\ L)$
$\delta \quad (p, 1) = (1\ q\ L)$
$\delta \quad (q, 0) = (1\ q\ L)$
$\delta \quad (q, 1) = (0\ q\ L)$



Transition Graph

Complexity of a TM

- The complexity of a TM is directly proportional to the size of the functional matrix. In other words, we can say that the complexity of a TM depends on the number of symbols that are being used and the number of states of the TM. Hence:
- Complexity of a TM = $|\Gamma| \times |Q|$ (or $|I| \times |S|$),

  where,

  $|\Gamma|$ = Cardinality of tape alphabet (i.e., number of tape symbols), and

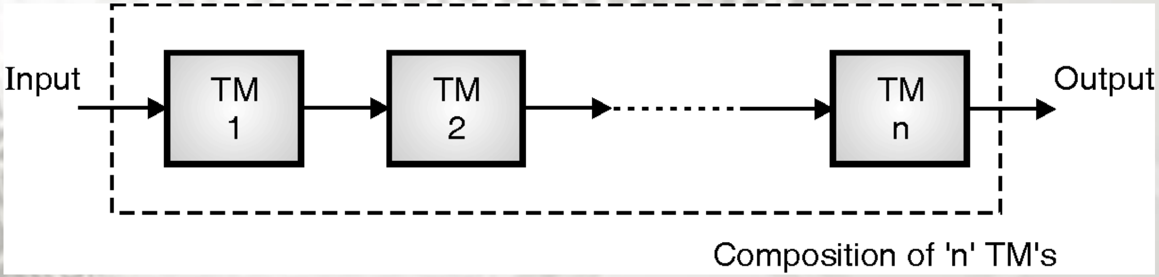
  $|Q|$ = Number of states of the TM.
- For example,

  If $\Gamma = \{1, 0, a, c, ;, ,\}$ and $Q = \{q_0, q_1, q_2, q_3, q_4 = \text{halt}\}$

  Then, the complexity of the TM = $|\Gamma| \times |Q| = 6 \times 5 = 30$

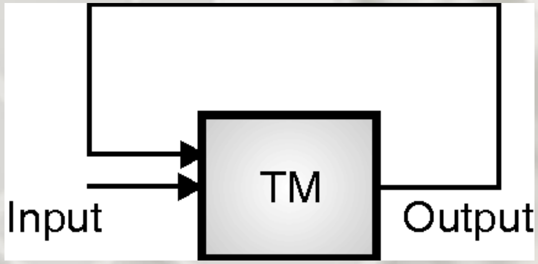
Composite TM and Iterative TM

- Two or more Turing machines can be combined to solve a complex problem, such that the output of one TM forms the input to the next TM, and so on. This is called *composition*.



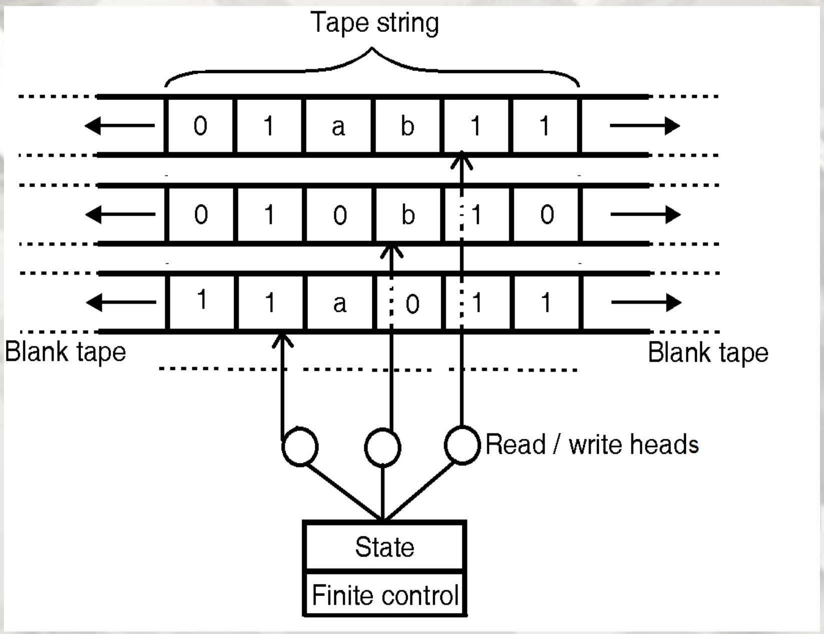
Composition of 'n' TM's

- Another way of having a combination TM is by applying its own output as input repetitively. This is called *iteration* or *recursion*, and the TM is said to be iterative TM (or ITM).



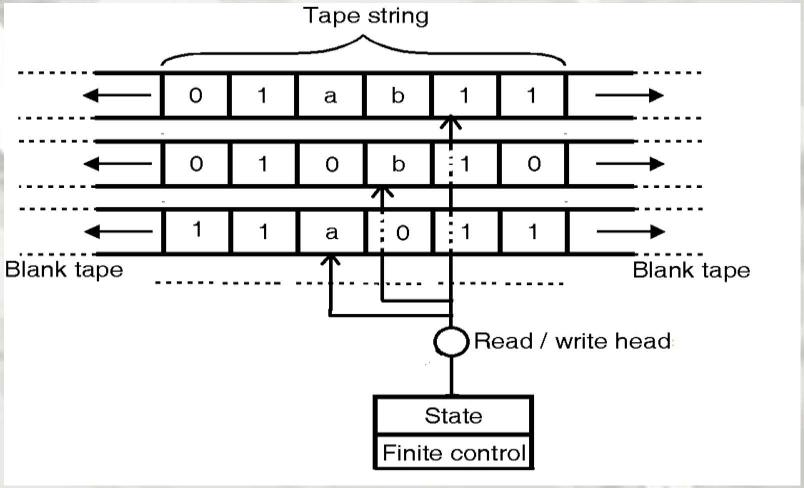
Multi-tape Turing Machine

- Multi-tape Turing machines have *k* number of independent tapes, having their own read/write heads. These machines have independent control over all the heads—any of these can move and read/write their own tapes. All these tapes are unbounded at both the ends just as in the single-tape TM.
- Multi-tape TM and single-tape TM are equivalent in power (except for some difference in execution time

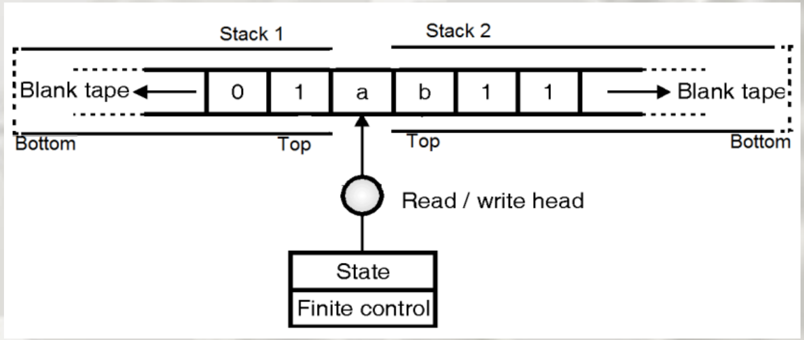


Multi-track and Multi-stack TM

- In a *k*-track Turing machine, one head reads and writes on all tracks simultaneously. A tape position in a *k*-track Turing machine denotes *k* symbols from the tape alphabet. This is equivalent to the standard single-tape Turing machine except that it reads/writes *k* symbols at one go.



- In multi-stack TM symbols to the left of the head of the TM can be stored onto one stack, while the symbols on the right of the head can be placed on the other stack.



Solvable and Semi-solvable Problems

- **Solvable** problem: TM when applied to such a problem, always eventually terminates with the correct "yes" or "no" answer
  - A class of all such problems is called as **Recursive language**
  - The mathematical functions that denote these type of problems are called as **Total Recursive Functions**
  - Simple Examples - multiplication, addition, concatenation and many other

- **Semi-solvable** problem: TM when applied to such a problem, always eventually terminates with correct answer when answer is "yes" and may or may not terminate when the correct answer is "no"
  - A class of all such problems is called as **Recursively Enumerable language**
  - The mathematical functions that denote these type of problems are called as **Partial Recursive Functions**
  - Simple Examples - division, factorial and many other

Recursively Enumerable and Recursive Languages

- **Recursively Enumerable Set**
  - A set S of words over Σ is said to be recursively enumerable, if there is a TM over Σ, which accepts every word in S and either rejects or loops for every word in $\sim S$ ($\sim S = \Sigma^* - S$). This can be represented as:
    - Accept TM = S
    - Reject (TM) $\cup$ loop (TM) = $\Sigma^* - S$
- **Recursive Set**
  - A set S of words over Σ is said to be recursive, if there is a TM over Σ, which accepts every word in S and rejects every word in $\sim S$ ($\sim S = \Sigma^* - S$). This can be represented as:
    - Accept (TM) = S
    - Reject (TM) = $\Sigma^* - S$
    - Loop (TM) = ϕ

Total and Partial Recursive Functions

- If $f(i_1, i_2, \dots i_k)$ is defined for all values of arguments, $i_1, \dots, i_k$, then f is said to be a **total recursive function**. These total recursive functions correspond to the recursive languages, since they are computed by a TM that always halts.
  - Total recursive functions are analogous to recursive languages.
  - For example, all common arithmetic functions on integers, such as multiplication, $[\log_2 n]$, and 2^{2n}, are total recursive functions.

- If $f(i_1, i_2, \dots, i_k)$ is not defined for all values of arguments $i_1, \dots, i_k$, then f is said to be a **partial recursive function**. In other words, a function $f(i_1, \dots, i_k)$ computed by a TM, which may or may not halt on a given input, is said to be a partial recursive function.
  - Partial recursive functions are analogous to recursively enumerable languages.
  - For example, factorial ($n!$) is only defined for integers, $n \geq 0$; it is undefined for negative integers.

Gödel Numbering

- Assign a unique natural number to each basic symbol
- Encoding of an entire sequence of symbols is obtained as the product of the first 'n' primes raised to their corresponding values in the sequence
- For example, given a sequence

  (x1, x2, ….., xn)

  its encoding can be written as,

  $2^{x1} . 3^{x2} . 5^{x3} . …. p_n^{xn}$"

  Note that, ‘x1’, ‘x2’ and so on are the encodings of the individual symbols and ‘2’, ‘3’, ‘5’,…. ‘p_n’ are the first ‘n’ primes

- Based on Number theory - any integer greater than ‘1’ can be written as a unique product of prime numbers.
  - For example, $10 = 1^1 * 2^1 * 5^1$, $24 = 1 * 2 * 2 * 2 * 3 = 1^1 * 2^3 * 3^1$

Turing Machine Encoding

- Binary encoding of a sample Turing Machine

{p, q}	== {0, 00}
{L, R, N}	== {0, 00, 000}
{0, 1}	== {0, 00}

	0	1
p	0pL	1qL
q	1qL	0qL

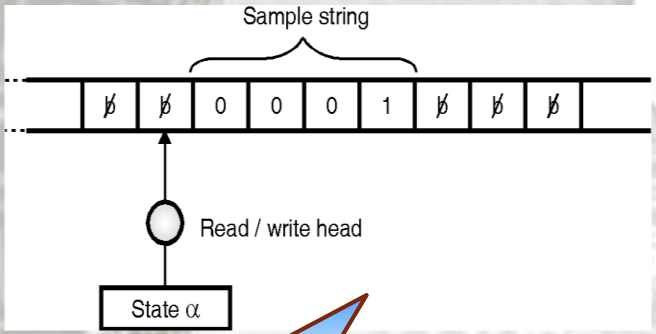
(p, 0) = (0,p,L)	==	010101010
(p, 1) = (1,q,L)	==	010010010010
(q, 0) = (1,q,L)	==	001010010010
(q, 1) = (0,q,L)	==	001001010010

- Symbol '1' is used as a separator

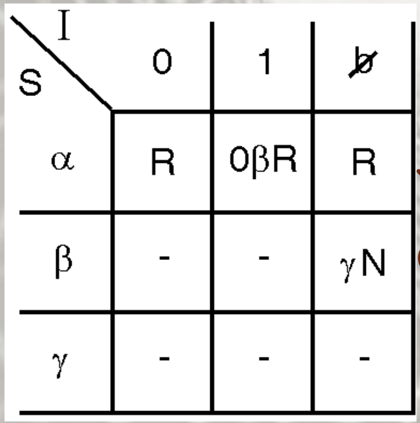
Encoding of the TM (Entire functional matrix considered in row-major order)

010101010 11 010010010010 11 001010010010 11 001001010010

TM Simulation



Initial ID



I / S	0	1	$\not{b}$
α	R	0βR	R
β	-	-	γN
γ	-	-	-

SFM

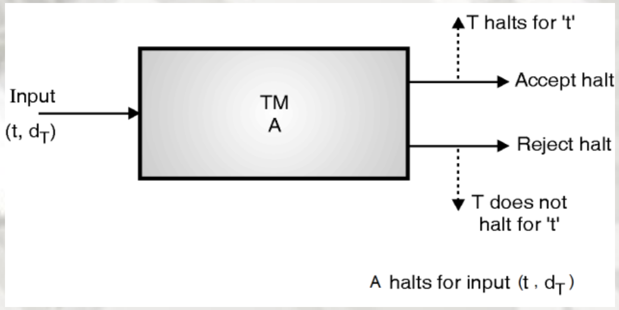
$\alpha\,\not{b}\,0\,0\,0\,1\,\not{b}$	,	initial configuration
$\vdash_M \not{b}\,\alpha\,0\,0\,0\,1\,\not{b}$	,	$\delta(\alpha, \not{b}) = (R)$
$\vdash_M \not{b}\,0\,\alpha\,0\,0\,1\,\not{b}$	,	$\delta(\alpha, 0) = (R)$
$\vdash_M \not{b}\,0\,0\,\alpha\,0\,1\,\not{b}$	,	$\delta(\alpha, 0) = (R)$
$\vdash_M \not{b}\,0\,0\,0\,\alpha\,1\,\not{b}$	,	$\delta(\alpha, 0) = (R)$
$\vdash_M \not{b}\,0\,0\,0\,0\,\beta\,\not{b}$	,	$\delta(\alpha, 1) = (0\,\beta\,R)$
$\vdash_M \not{b}\,0\,0\,0\,0\,\gamma\,\not{b}$	,	$\delta(\beta, \not{b}) = (\gamma\,N)$

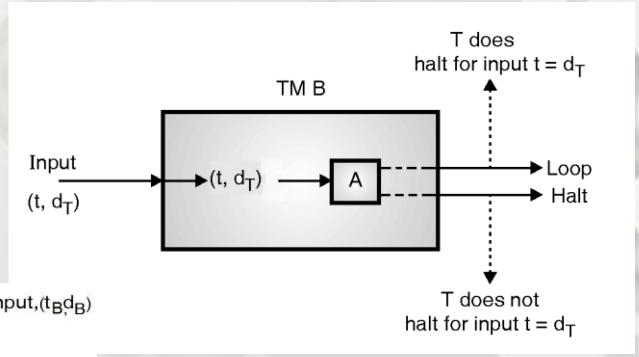
Universal Turing Machine (UTM)

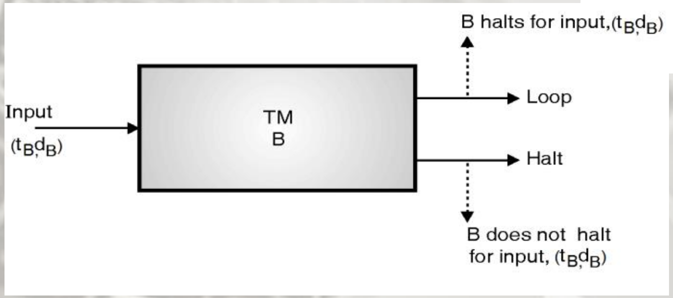
- **Universal Turing Machine (UTM)**
  - Turing machine that can simulate any other Turing Machine
  - It accepts the *encoded* Functional Matrix of any other TM as input on its tape (**Program area**)
  - It also accepts the data on which the other TM needs to be simulated (**Data area** of the tape)
  - UTM needs an **imitation algorithm** that can simulate the functional matrix of any other TM (**System area**)
  - Functional Matrix of such a UTM is analogous to an Operating System
  - **UTM is analogous to a modern Computer!**

Halting Problem and Unsolvability

- For a given input for any general TM two cases arise,
  - The machine may halt after a finite number of steps
  - The machine may not ever halt no matter how long it runs
- Given any TM, problem of algorithmically determining whether it ever halts or not, is called as the **Halting Problem**
- The halting problem is **unsolvable**




TM B
Input
(t, d_T)
(t, d_T)
A
T does halt for input t = d_T
Loop
Halt
T does not halt for input t = d_T


Input
($t_B d_B$)
TM B
B halts for input,($t_B d_B$)
Loop
Halt
B does not halt for input, ($t_B d_B$)

Church's Turing Hypothesis

- This hypothesis is also known as *Church–Turing conjecture, Church's Thesis*, or *Church's conjecture*. It essentially states that everything that is algorithmically computable is computable by a Turing machine.

- **Statement**: The intuitive notion of a computable function can be identified with the class of partial recursive functions.
- In other words, every problem having an algorithmic solution can be solved using a machine having the foregoing set of instructions.

Post's Correspondence Problem (PCP)

- Let $A = w_1, w_2, \ldots, w_k$, and $B = x_1, x_2, \ldots, x_k$ be strings over some alphabet. Post's correspondence problem (PCP) is to find the correspondence sequence of integers, $i_1, i_2, \ldots, i_m$, for $m \geq 1$ such that:

$$w_{i1}, w_{i2}, \ldots, w_{im} = x_{i1}, x_{i2}, \ldots, x_{im}$$

- The sequence, '$i_1, i_2, \ldots, i_m$' is considered as the solution for the PCP instance. Each PCP instance is constituted by some set of values for A and B.
- If we consider the PCP as a generic class of all such instances, then it is *unsolvable*. Furthermore, there exists no generic algorithm that can find a solution for any such PCP instance; hence, it is also an *undecidable* problem. However, for a few values of A and B, it might have a solution.

Linear Bounded Automata (LBA)

- A linear bounded automaton (LBA) is a restricted form of Turing machine, which follows three main restrictions:
  - The tape alphabet includes two special symbols, serving as left and right end-markers (just as in the case of two-way finite automata; refer to Chapter 2).
  - The end-markers cannot be erased. Hence, one cannot write anything at the end-marker positions.
  - Transitions do not make the head move to the left of the left end-marker or to the right of the right end-marker.

- These limitations make the LBA a somewhat more *realistic* model of computers than a Turing machine.