*Pharos University in Alexandria*
*Faculty of Computer Science & Artificial Intelligence*
*Course Title: Theory of Computation*
*Code: CS 307*


# Theory of Computation

**Lecturer: Sherine Shawky**

Text Books

1. Introduction to formal languages and automata, Peter Linz, 6th edition, 2017.

# Week 11

# Turing Machine

# Introduction

- Till now we have seen machines which can move in one direction: left to right.
- But Turing machine can move in both directions and also it can read from TAPE as well as write on it.
- Turing machine can accept recursive enumerable languages
- Turing machine(Automata Machine) was proposed in 1931 by Alan Turing. Back then it was termed as "The Computer".

# Turing Machine

- Here we introduce the important concept of a *Turing machine*, a finite-state control unit to which is attached a one dimensional, unbounded tape.
- Even though a Turing machine is still a very simple structure, it turns out to be very powerful and lets us solve many problems that cannot be solved with a pushdown automaton.

# Turing Machine

- The regular languages form a proper subset of the context-free languages and, therefore, that pushdown automata are more powerful than finite automata.
- The context-free languages, while fundamental to the study of programming languages, are limited in scope.
- Some simple languages are not context-free. This prompts us to look beyond context-free languages and investigate how one might define new language families that include these examples.

# Turing Machine

- To do so, we return to the general picture of an automaton.
- If we compare finite automata with pushdown automata, we see that the nature of the temporary storage creates the difference between them.
- If there is no storage, we have a finite automaton; if the storage is a stack, we have the more powerful pushdown automaton.
- Extrapolating from this observation, we can expect to discover even more powerful language families if we give the automaton more flexible storage.

# Turing Machine

- For example, what would happen if we used two stacks, three stacks, a queue, or some other storage device?
- Does each storage device define a new kind of automaton and through it a new language family?
- This approach raises a large number of questions, most of which turn out to be uninteresting.
- It is more instructive to ask a more ambitious question and consider how far the concept of an automaton can be pushed.
- What can we say about the most powerful of automata and the limits of computation? This leads to the fundamental concept of a Turing machine and, in turn, to a precise definition of the idea of a mechanical or algorithmic computation.

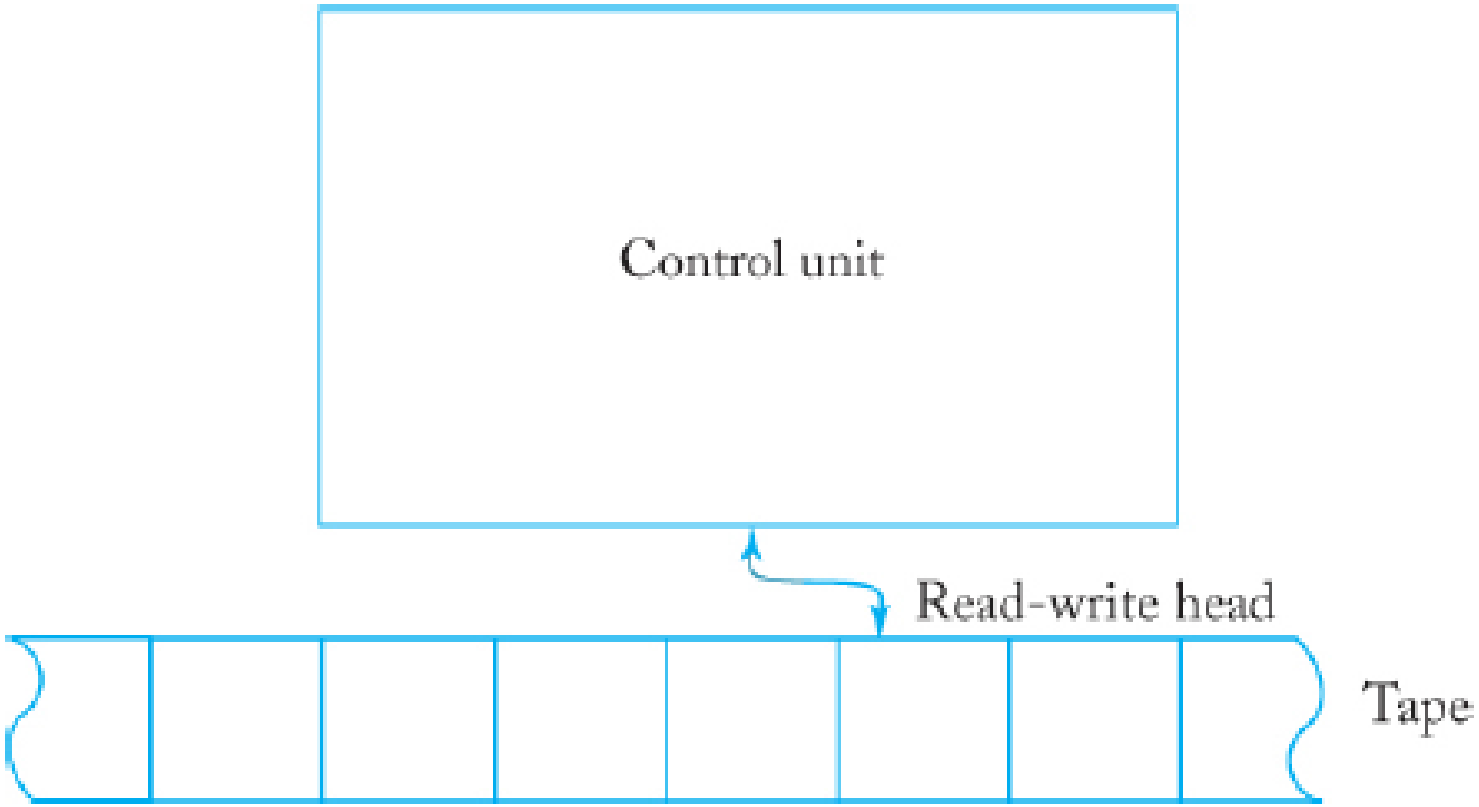
# Turing Machine

- Although we can envision a variety of automata with complex and sophisticated storage devices, a Turing machine's storage is actually quite simple.
- It can be visualized as a single, one-dimensional array of cells, each of which can hold a single symbol.
- This array extends indefinitely in both directions and is therefore capable of holding an unlimited amount of information.
- The information can be read and changed in any order.
- We will call such a storage device a tape because it is analogous to the magnetic tapes used in older computers.

# Turing Machine

- A Turing machine is an automaton whose temporary storage is a tape.
- This tape is divided into cells, each of which is capable of holding one symbol.
- Associated with the tape is a read-write head that can travel right or left on the tape and that can read and write a single symbol on each move.
- The automaton that we use as a Turing machine will have neither an input file nor any special output mechanism.
- Whatever input and output is necessary will be done on the machine’s tape.
- We could retain the input file and a specific output mechanism without affecting any of the conclusions we are about to draw, but we leave them out because the resulting automaton is a little easier to describe.

# Turing Machine



# Turing Machine

- A Turing machine $M$ is defined by

  $M = (Q, \Sigma, \Gamma, \delta, q0, \square, F)$, where

  - $Q$ is the set of internal states,
  - $\Sigma$ is the input alphabet,
  - $\Gamma$ is a finite set of symbols called the tape alphabet,
  - $\delta$ is the transition function,
  - $\square \in \Gamma$ is a special symbol called the blank,
  - $q0 \in Q$ is the initial state,
  - $F \subseteq Q$ is the set of final states.

# Turing Machine

- In the definition of a Turing machine, we assume that $\Sigma \subseteq \Gamma - \{\square\}$, that is, that the input alphabet is a subset of the tape alphabet, not including the blank.
- Blanks are ruled out as input for reasons that will become apparent shortly. The transition function $\delta$ is defined as $\delta : Q \times \Gamma \rightarrow Q \times \Gamma \times \{L, R\}$.
- In general, $\delta$ is a partial function on $Q \times \Gamma$; its interpretation gives the principle by which a Turing machine operates.

# Turing Machine

- The arguments of δ are the current state of the control unit and the current tape symbol being read.
- The result is a new state of the control unit, a new tape symbol, which replaces the old one, and a move symbol, *L* or *R*.
- The move symbol indicates whether the read-write head moves left or right one cell after the new symbol has been written on the tape.

# Turing Machine

- Turing Machines as Language Accepters
- Turing machines can be viewed as accepters in the following sense.
- A string *w* is written on the tape, with blanks filling out the unused portions.
- The machine is started in the initial state *q*0 with the readwrite head positioned on the leftmost symbol of *w*.
- If, after a sequence of moves, the Turing machine enters a final state and halts, then *w* is considered to be accepted.

# Turing Machine

- DEFINITION 9.3
- Let $M = (Q, \Sigma, \Gamma, \delta, q0, \square, F)$ be a Turing machine.
- Then the language accepted by $M$ is

  $L(M) = \{w \in \Sigma+ : q0w* \vdash x1qf\, x2$ for some $qf \in F, x1, x2 \in \Gamma*\}$.

# Turing Machine

- This definition indicates that the input *w* is written on the tape with blanks on either side.
- The reason for excluding blanks from the input now becomes clear: It assures us that all the input is restricted to a welldefined region of the tape, bracketed by blanks on the right and left.
- Without this convention, the machine could not limit the region in which it must look for the input; no matter how many blanks it saw, it could never be sure that there was not some nonblank input somewhere else on the tape.

# Turing Machine

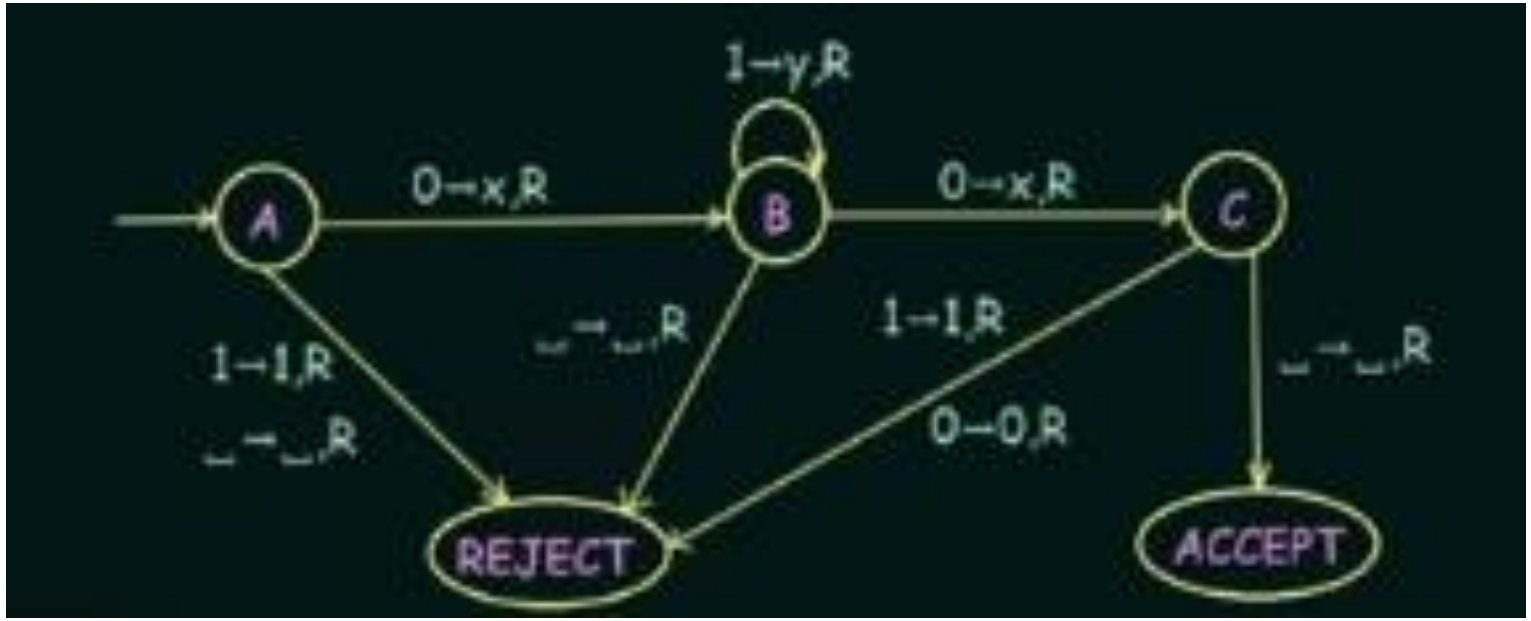
- Definition 9.3 tells us what must happen when $w \in L(M)$.
- It says nothing about the outcome for any other input.
- When $w$ is not in $L(M)$, one of two things can happen: The machine can halt in a nonfinal state or it can enter an infinite loop and never halt.
- Any string for which $M$ does not halt is by definition not in $L(M)$.

# Turing Machine

- Design a TM for language that accepts strings of the form 01*0

# Turing Machine

- Design a TM for language that accepts strings of the form 01*0
- Answer

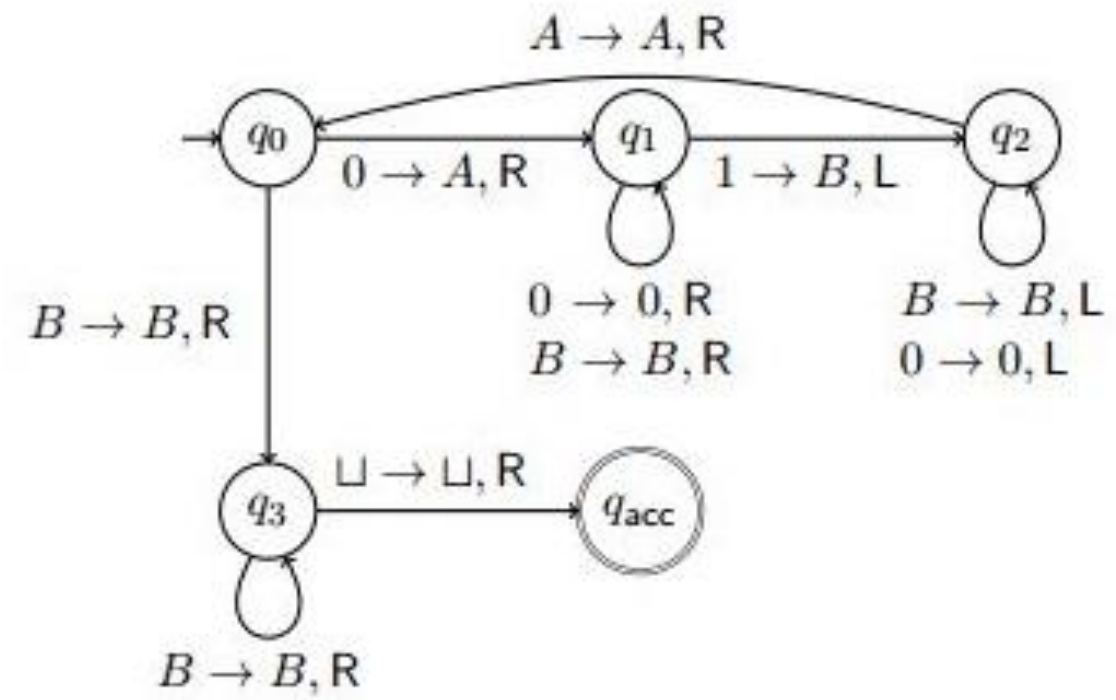


# Turing Machine

- For $\Sigma = \{a, b\}$, design a Turing machine that accepts $L = \{a^n b^n : n \geq 1\}$.

# Turing Machine

- For $\Sigma = \{a, b\}$, design a Turing machine that accepts $L = \{a^n b^n : n \geq 1\}$.
- Answer



# TM for Complicated Tasks

- We have shown explicitly how some important operations found in all computers can be done on a Turing machine.
- Since, in digital computers, such primitive operations are the building blocks for more complex instructions, let us see how these basic operations can also be put together on a Turing machine.
- To demonstrate how Turing machines can be combined, we follow a practice common in programming.
- We start with a high level description, then refine it successively until the program is in the actual language with which we are working.
- We can describe Turing machines several ways at a high level; block diagrams or pseudocode are the two approaches we will use most frequently in subsequent discussions.

# TM as Adder

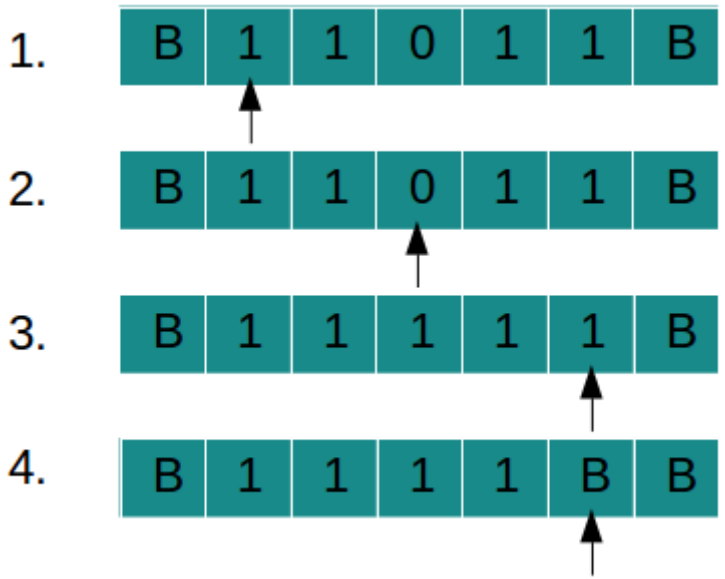
- Design a Turing machine that computes the function
- $f(x, y) = x + y$

# TM as Adder

- Approach for Addition
- Numbers are given in Uniary form
- Example: 3 = 111, 2 = 11, 5 = 11111 etc.
- For addition of 3 and 4, numbers will be given in TAPE as "B B 1 1 1 0 1 1 1 1 B B".
- Convert '0' to '1' and reduce '1' from right
- Hence output will be "B B 1 1 1 1 1 1 1 B B B"
- Total number of '1' in output is = 7 which is addition of 3 and 4

# Turing Machine

- TAPE movement for string "110111":

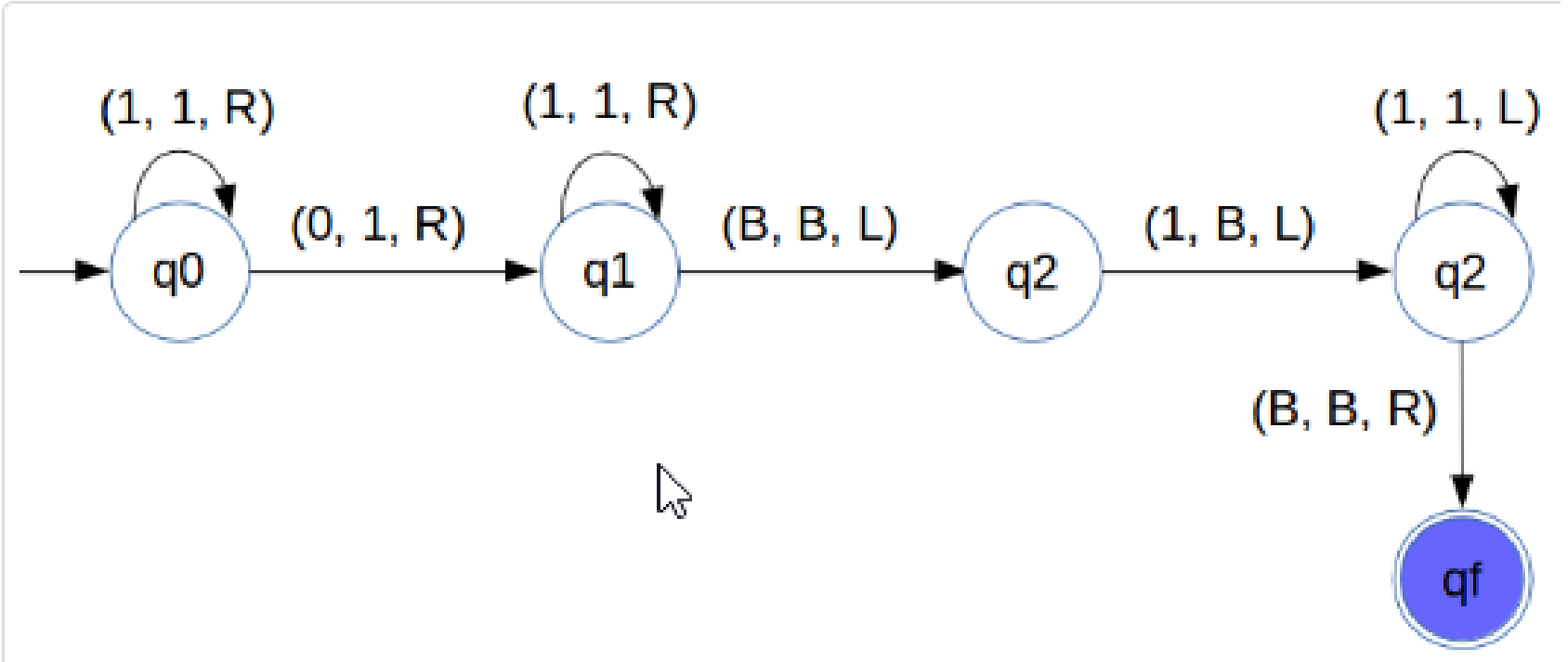


# Turing Machine

- Explanation of TAPE movement
- Input is given as "11011" (2 + 2)
- Scan string from left to right
- Pass all '1'SSS and convert '0' to '1'
- Reach BLANK in right
- Convert rightmost '1' to BLANK
- **Addition of 2 and 2 = 4 is written on TAPE**
  Input String : 11011
  Output String : **1111**

# Turing Machine

- State Transition Diagram



# Turing Machine

- We have designed state transition diagram for adder as follows:
  Start scanning string from left to right
- Pass all '1's and keep moving right
- Convert '0' to '1' and keep moving right
- When BLANK(in right) is reached move one step left **convert '1' to BLANK**
- Keep moving left after that to point start of string
- Number of '1's is reduced because number of '1' was increased by one while converting '0' to '1'
- ***Uniary addition is like string concatination.***