Turing Machines

Lecture 7

Strathmore UNIVERSITY

Lecture Outline

- *Turing Machines (TM's)*
- *Turing Machine Operation*
- *TM Formal definition*
- *Variations of TM's*
- *Application of TM's*
- ***Alan Turing (Optional)***

Turing Machines

- Just like Finite Automata, Turing Machine (TM) is an abstract model of computation designed by Alan Turing
- Turing Machines (TM’s) are most powerful model of a computer
- In contrast to Finite Automata (FA) , TM’s have ***infinite tape for storage*** which act as a memory

Turing Machines

- A Turing Machine can do everything that a real computer can do
- Nonetheless, even a Turing machine cannot solve certain problems which are beyond the theoretical limits of computation
- Initially the tape contains only the input string and is blank everywhere else

Turing Machines

- If the machine needs to store information, it may write this information on the tape
- To read the information that it has written, the machine can move its head back over it
- The machine continues computing until it decides to produce an output.
- The outputs accept and reject are obtained by entering designated accepting and rejecting states

Turing Machines

- If it doesn't enter an accepting or a rejecting state, it will go on forever, never halting

Turing Machines vs Finite Automata

- The simplest automata used for computation is a finite automaton which can compute only very primitive functions hence not an adequate computation model
- Further, a finite-state machine’s inability to generalize computations hinders its power

Turing Machines vs Finite Automata

Strathmore UNIVERSITY

- [*Source: Stanford*] The key difference between a finite-state machine and a Turing Machine are as follows:
  - *"Imagine a Modern CPU. Every bit in a machine can only be in two states (0 or 1). Therefore, there are a finite number of possible states"*

Turing Machines vs Finite Automata

- The key difference between a finite-state machine and a Turing Machine as as follows:
  - *“In addition, when considering the parts of a computer a CPU interacts with, there are a finite number of possible inputs from the computer's mouse, keyboard, hard disk, different slot cards, etc”*

Turing Machines vs Finite Automata

- The key difference between a finite-state machine and a Turing Machine as as follows:
  - *“As a result, one can conclude that a CPU can be modeled as a finite-state machine*
  - *Now, consider a computer. Although every bit in a machine can only be in two different states (0 or 1),”*

Turing Machines vs Finite Automata

- The key difference between a finite-state machine and a Turing machine as as follows:
  - *" there are an infinite number of interactions within the computer as a whole. It becomes exceeding difficult to model the workings of a computer within the constraints of a finite-state machine"*

Turing Machines vs Finite Automata

Strathmore UNIVERSITY

- The key difference between a finite-state machine and a Turing machine as as follows:
  - *" However, higher-level, infinite and more powerful automata would be capable of carrying out this task"*

Turing Machines vs Finite Automata

- *Alan Turing conceived the first "infinite" (or unbounded) model of computation: the Turing Machine, in 1936*
- *The Turing Machine can be conceptualised as automaton or control unit having an infinite storage (memory)*

Turing Machines vs Finite Automata

- The memory consists of an infinite number of array of cells
- *"Turing's machine is essentially an abstract model of modern-day computer execution and storage, developed in order to provide a precise mathematical definition of an algorithm or mechanical procedure."*

Turing Machines vs Finite Automata

- A Turing machine has two alphabets:
  - **An input alphabet Σ** - All input strings are written in the input alphabet
  - **A tape alphabet Γ**, where $\Sigma \subseteq \Gamma$ - The tape alphabet contains all symbols that can be written onto the tape

Turing Machines vs Finite Automata

- A Turing Machine has two alphabets:
  - The tape alphabet Γ can contain any number of symbols, but always contains at least one blank symbol.
  - At start, the TM begins with an infinite tape of symbols with the input written at some location
  - The tape head is at the start of the input

Turing Machines vs Finite Automata

Strathmore UNIVERSITY

1. A Turing machine can both write on the tape and read from it
2. The read–write head can move both to the left and to the right
3. The tape is infinite
4. The special states for rejecting and accepting take effect immediately

Turing Machines vs Finite Automata

Strathmore UNIVERSITY

Similarities

- Simple model of computation.
- Input on tape is a finite string with symbols from a finite alphabet.
- Finite number of states.
- State transitions determined by current state and input symbol.

Differences

DFAs

- Can read input symbols from the tape.
- Can only move tape head to the right.
- Tape is finite (a string).
- One step per input symbol.
- Can *recognize* (turn on "YES" or "NO").

TMs

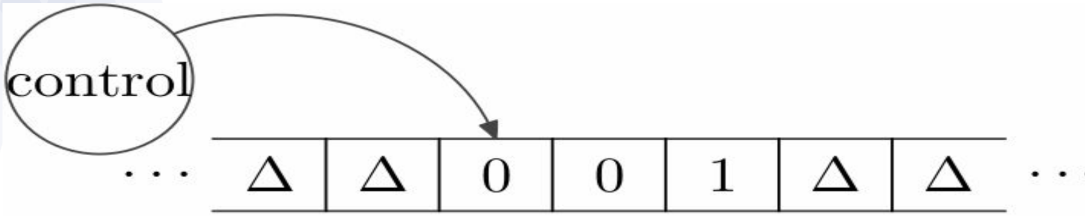
- Can read from or write onto the tape.
- Can move tape head either direction.
- Tape does not end (either direction).
- No limit on number of steps.
- Can also *compute* (with output on tape).

Turing Machine Operation

- A Turing Machine (TM) has three major components:
  - An **infinite tape** divided into cells. Each cell contains one symbol
  - A **head** that accesses one cell at a time, and which can both read from and write on the tape, and can move both left and right
  - A **memory** that is in one of a fixed finite number of states

Turing Machine Operation

- We assume a two-way infinite tape that stretches to infinity in both directions
- Δ denotes an empty or blank cell (Also denoted by ␣ or B)
- The input starts on the tape surrounded by Δ or ␣ with the head at left-most symbol (if input is ε, then tape is empty and head points to empty cell)



Turing Machine Operation

- Once a TM has started, the computation proceeds according to the rules described by the transition function
- If the machine ever tries to move its head to the left off the left-hand end of the tape, the head stays in the same place for that move, even though the transition function may indicate L (Left)
- The *computation continues until it enters either the accept or reject states, at which point it halts*

Turing Machine Operation

- *If neither occurs, the machine goes on forever*
- As a Turing Machine computes, changes occur in the *current state*, the *current tape contents,* and the *current head location*
- A setting of these three items is called a ***configuration of the Turing Machine***

Turing Machine Operation

- The collection of strings that a Turing Machine M accepts is the *language of M , or the language recognized by M , denoted L(M)*
- Therefore, we refer to a language **Turing-recognizable** if some Turing Machine recognizes it

Turing Machine Operation

- When a TM loops, this may entail any simple or complex behavior that never leads to a halting state
- *A Turing Machine M can fail to accept an input by entering the reject state and rejecting, or by looping*
- Turing machines that halt on all inputs (never loop) are called **deciders** since they always make a decision to accept or reject. *A decider that recognizes some language also is said to decide that language*

Turing Machine Operation

- We call a language **Turing-decidable** or **Recursively Enumerable Language** or **decidable** if some Turing Machine decides it

TM's Formal Definition

- A Turing machine is a 7-tuple, $(Q, \Sigma, \Gamma, \delta, q_0, q_{accept}, q_{reject})$, where Q, Σ, Γ are all finite sets and
    1. Q is the set of states,
    2. Σ is the input alphabet not containing the blank symbol $\sqcup$,
    3. Γ is the tape alphabet, where $\sqcup \in \Gamma$ and $\Sigma \subseteq \Gamma$,
    4. $\delta : Q \times \Gamma \rightarrow Q \times \Gamma \times \{L, R, S\}$ is the transition function,
    5. $q_0 \in Q$ is the start state,
    6. $q_{accept} \in Q$ is the accept state, and
    7. $q_{reject} \in Q$ is the reject state where $q_{accept} \neq q_{reject}$

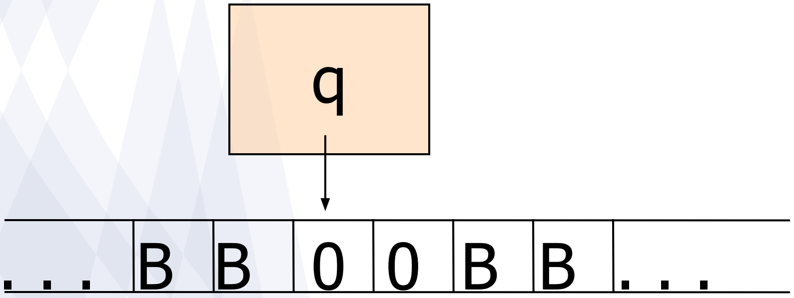
TM Example 1

- A TM to scan input right, turning each 0 into a 1.
    - If it ever finds a 1, it goes to **final** reject state r, and halts.
    - If it reaches a blank, it changes moves left and accepts.
- Its language is 0*

TM Example 1

- States = {q (start), f (accept), r (reject)}
- Input symbols = {0, 1}
- Tape symbols = {0, 1, B}
- δ:
  - $\delta(q, 0) = (q, 1, R)$
  - $\delta(q, 1) = (r, 1, R)$
  - $\delta(q, B) = (f, B, L)$

TM Example 1



$\delta(q, 0) = (q, 1, R)$

$\delta(q, 1) = (r, 1, R)$

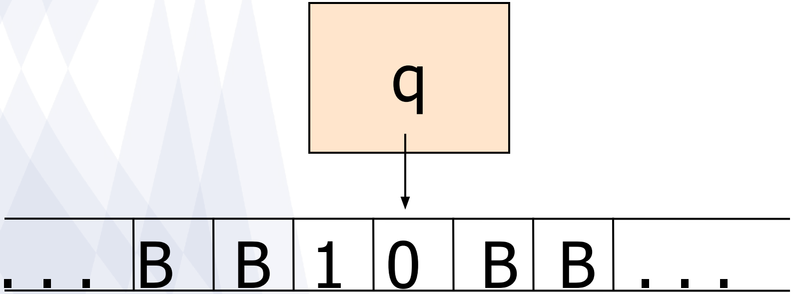
$\delta(q, B) = (f, B, L)$

TM Example 1

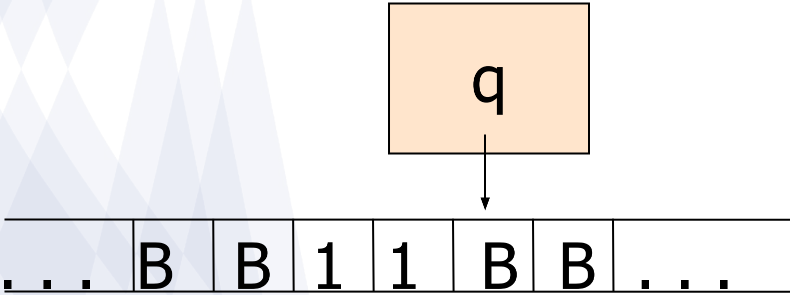
$\delta(q, 0) = (q, 1, R)$

$\delta(q, 1) = (r, 1, R)$

$\delta(q, B) = (f, B, L)$



TM Example 1

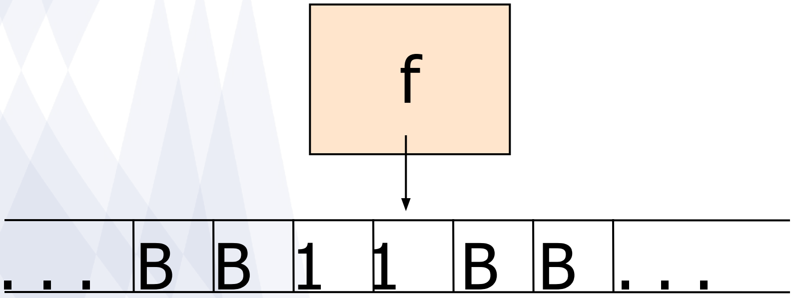


$\delta(q, 0) = (q, 1, R)$

$\delta(q, 1) = (r, 1, R)$

$\delta(q, B) = (f, B, L)$

TM Example 1



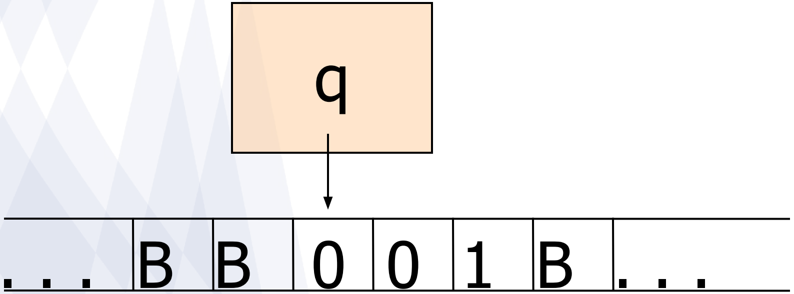
$\delta(q, 0) = (q, 1, R)$

$\delta(q, 1) = (r, 1, R)$

$\delta(q, B) = (f, B, L)$

The TM halts and accepts. (So “00” is in its language.)

TM Example 2

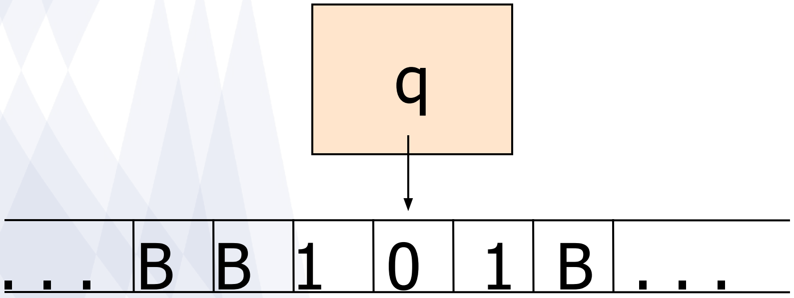


$\delta(q, 0) = (q, 1, R)$

$\delta(q, 1) = (r, 1, R)$

$\delta(q, B) = (f, B, L)$

TM Example 2



$\delta(q, 0) = (q, 1, R)$

$\delta(q, 1) = (r, 1, R)$

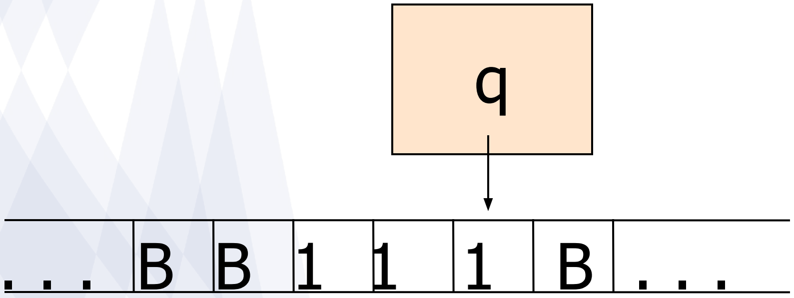
$\delta(q, B) = (f, B, L)$

TM Example 2

$\delta(q, 0) = (q, 1, R)$

$\delta(q, 1) = (r, 1, R)$

$\delta(q, B) = (f, B, L)$

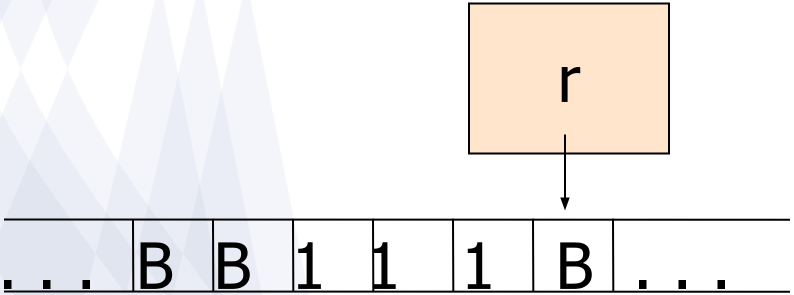


TM Example 2

$\delta(q, 0) = (q, 1, R)$

$\delta(q, 1) = (r, 1, R)$

$\delta(q, B) = (f, B, L)$



The TM halts and rejects. (So “001” is not in its language.)

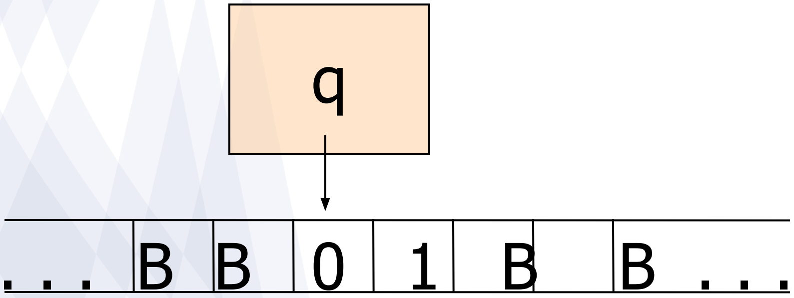
TM Example 3 (Looping)

- Once a TM has entered either the accept state or reject state, it **halts**
- But there is no rule that a TM must halt
- Turing Machines can have infinite loops

TM Example 3 (Looping)

- States = {q (start), f (accept), r (reject)}
- Input symbols = {0, 1}.
- Tape symbols = {0, 1, B}
- δ:
  - δ(q, 0) = (q, 0, R)
  - δ(q, 1) = (q, 1, L)
  - δ(q, B) = (f, B, L)

TM Example 3 (Looping)



$\delta(q, 0) = (q, 0, R)$

$\delta(q, 1) = (q, 1, L)$

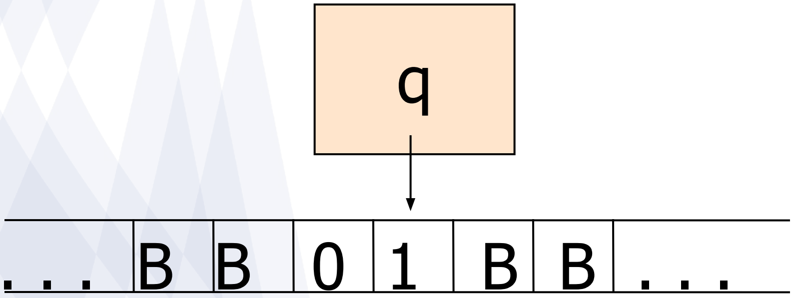
$\delta(q, B) = (f, B, L)$

TM Example 3 (Looping)

$\delta(q, 0) = (q, 0, R)$

$\delta(q, 1) = (q, 1, L)$

$\delta(q, B) = (f, B, L)$

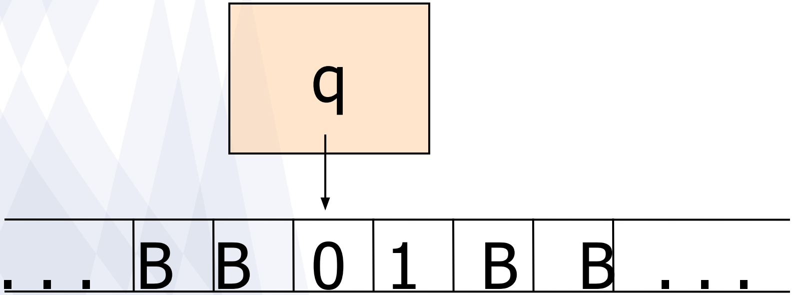


TM Example 3 (Looping)

$\delta(q, 0) = (q, 0, R)$

$\delta(q, 1) = (q, 1, L)$

$\delta(q, B) = (f, B, L)$

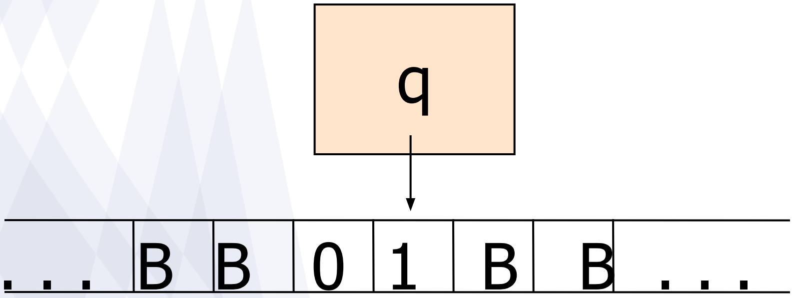


TM Example 3 (Looping)

$\delta(q, 0) = (q, 0, R)$

$\delta(q, 1) = (q, 1, L)$

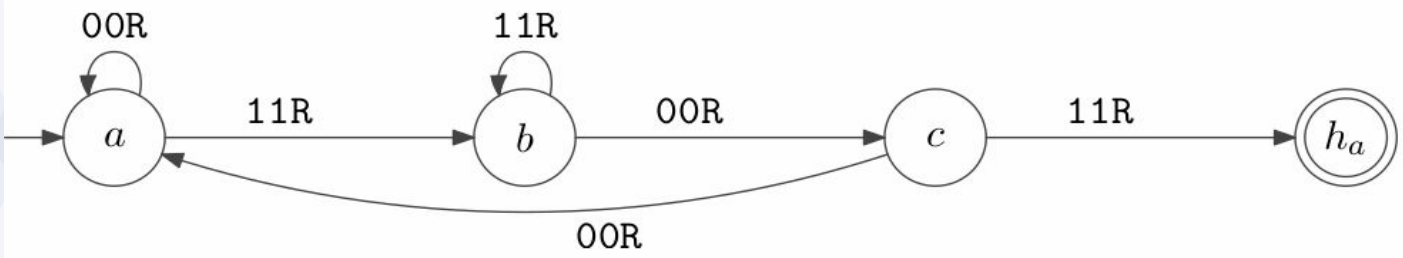
$\delta(q, B) = (f, B, L)$



- The TM never halts in this case
- But notice that it still accepts the language 0*

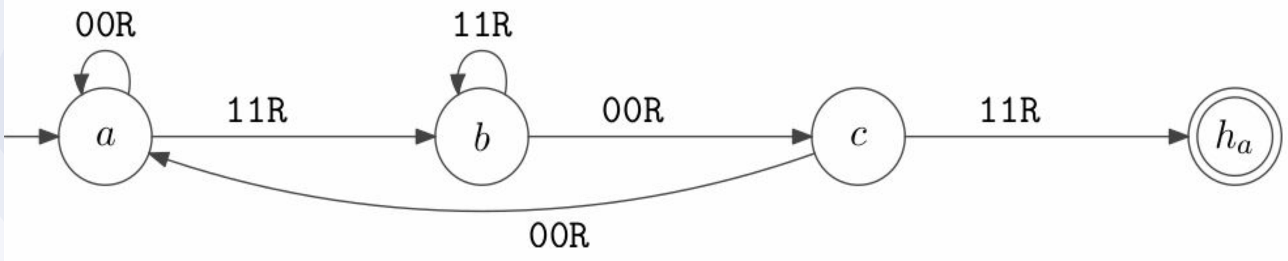
TM Example 4

- Here is a simple TM that mimics an FA for the language of all binary strings that contain the substring 101
- What is the formal description of the machine?

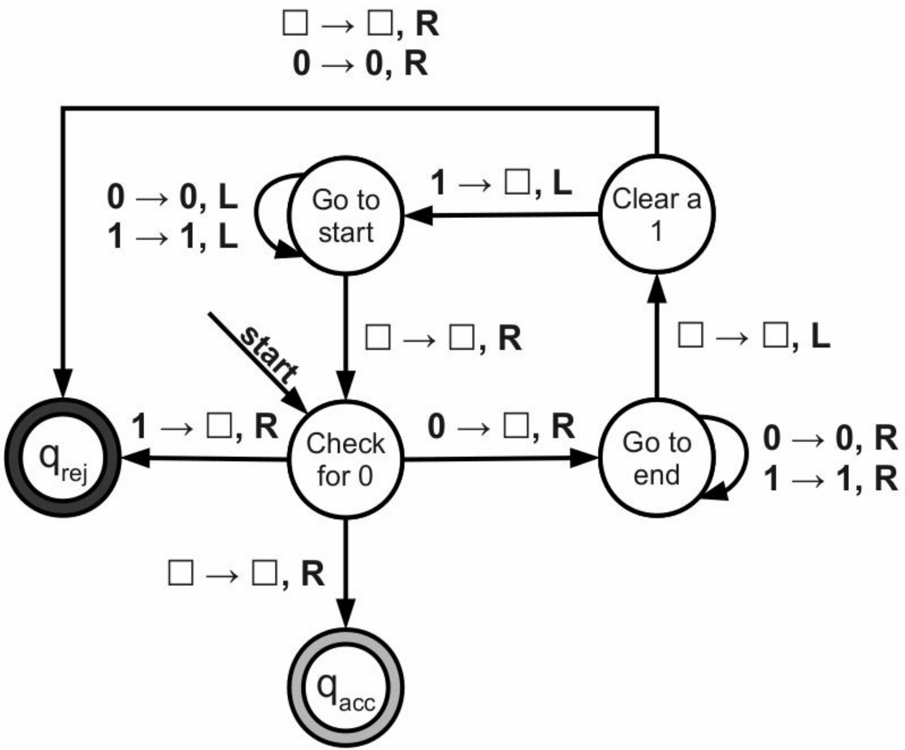


TM Example 4

- As a Turing machine computes, changes occur in the current state, the current tape contents, and the current head location.
- A setting of these three items is called a configuration of the Turing Machine



TM Example 5 (L= {0^N1^N})



TM Example 3- TM for L= $\{0^N 1^N\}$

State	Symbol				
	0	1	X	Y	B
q_0	(q_1, X, R)	-	-	(q_3, Y, R)	-
q_1	$(q_1, 0, R)$	(q_2, Y, L)	-	(q_1, Y, R)	-
q_2	$(q_2, 0, L)$	-	(q_0, X, R)	(q_2, Y, L)	-
q_3	-	-	-	(q_3, Y, R)	(q_4, B, R)
q_4	-	-	-	-	-

Variations of Turing Machines

***(Covered in Assignment 02)*

1. A multitape Turing Machine
2. Nondeterministic Turing Machines
3. Enumerators
4. ?

Related Video Content

1. Turing machine & Halting Problem
2. Lambda Calculus
3. Turing's Enigma problem

Turing Machine Applications

- **(Read on this)*

Who is Alan Turing?
(**Optional** Reading Exercise)

"His death is regarded by most as an act of suicide"

Questions