Computing Theory

COMP 147 (4 units)

Chapter 3: The Church-Turing Thesis

Section 3.1: Turing Machines

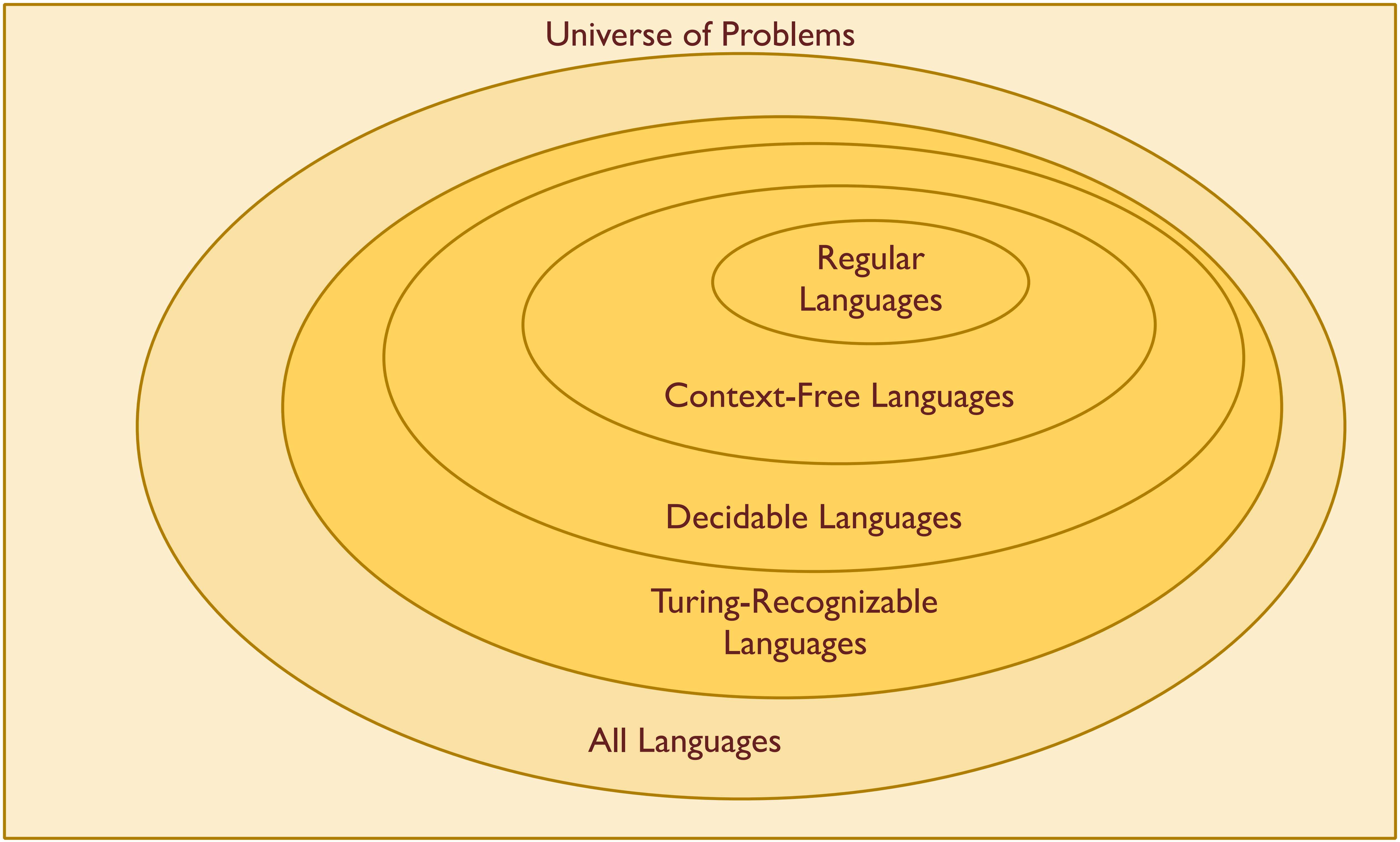
Course Segments

- Automata and Languages
  - How can we define abstract models of computers?
- Computability Theory
  - What can (or cannot) be computed?
- Complexity Theory
  - What makes some problems computationally difficult?

Turing Machines

- Finite State Machines
    - Regular Languages
- Pushdown Automata
    - Context-Free Languages
- Turing Machines
    - New computation model - a model for all computers
    - Decidable Languages
    - Turing Recognizing Languages

The Space of Problems



Turing Machines

- A Turing machine can do anything
  that a real computer can do.

- The Turning machine was proposed by Alan Turing
  in 1936.

image: http://en.wikipedia.org/wiki/File:Alan_Turing_photo.jpg

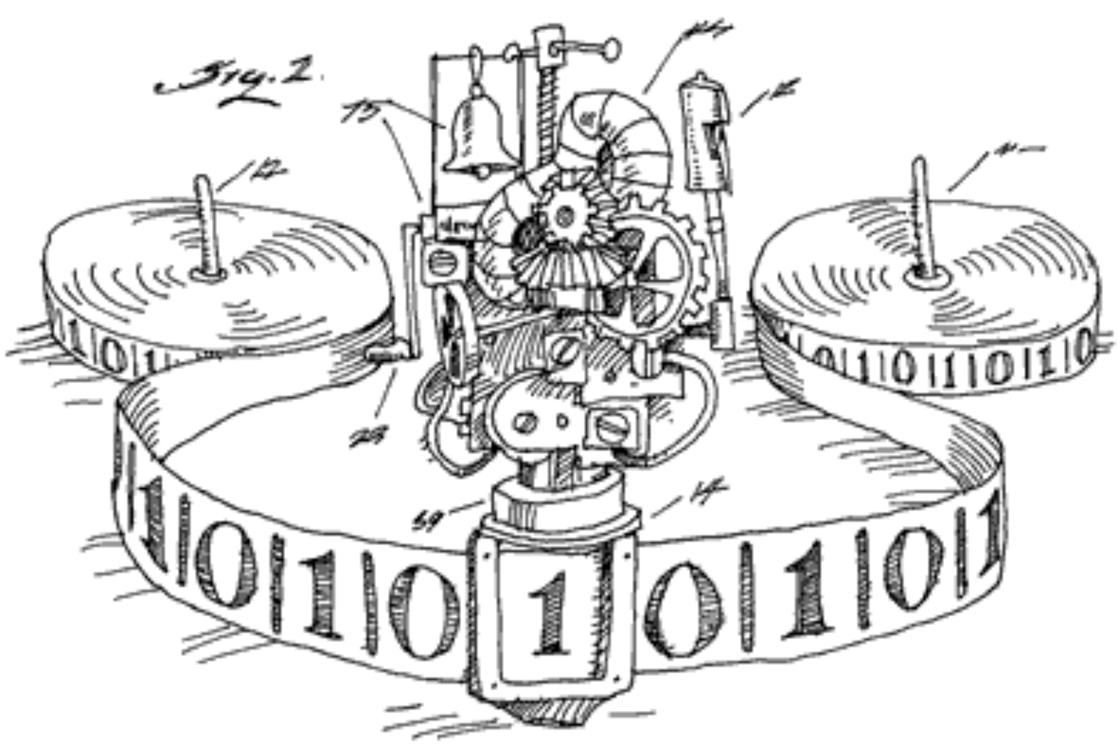
Turing Machines

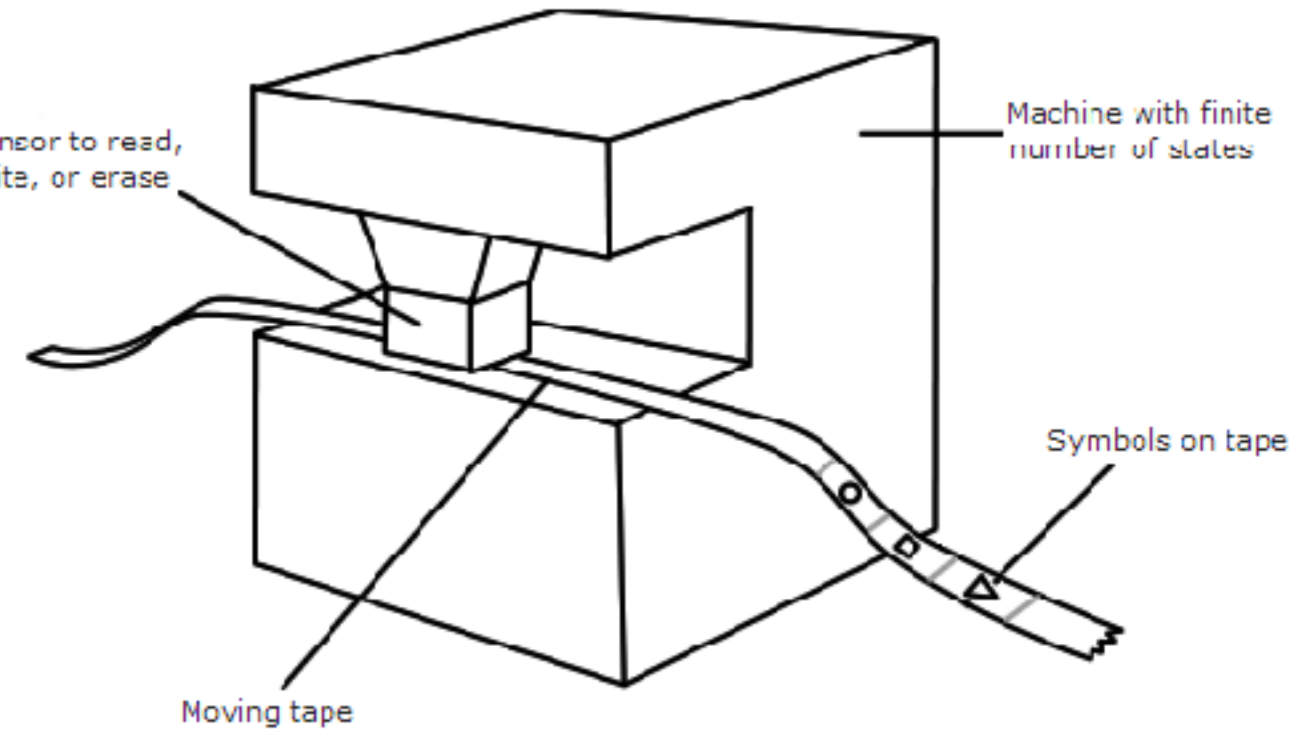
- A Turing machine can do anything that a real computer can do

- A Turing machine cannot solve certain problems

- What implications can we draw from these two statements?

What is a Turing Machine?





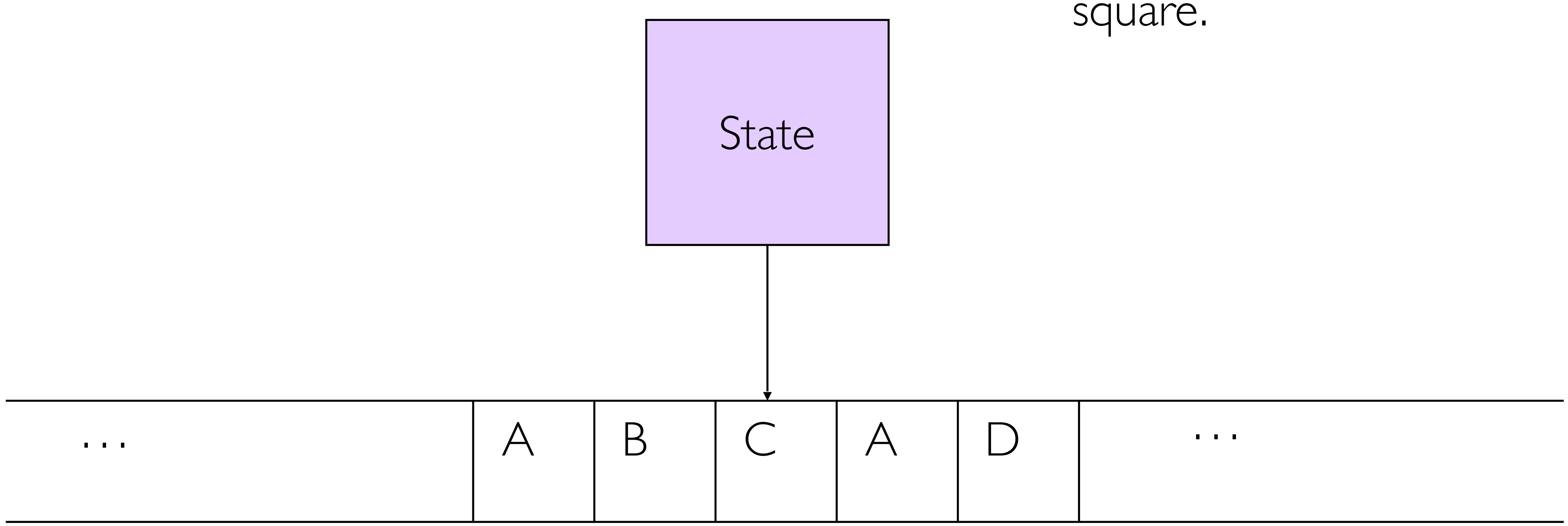
A Turing Machine

A Turing Machine (TM) is similar to a PDA,
with the stack replaced by a tape.
The tape is an infinite sequence of cells,
each of which can store one symbol.
The TM can move left and right along the tape,
reading and/or writing symbols in the cells.

Images: http://www.felienne.com/?p=2974, http://commonsenseatheism.com/?p=13607

Picture of a Turing Machine

Action: based on the state and the tape symbol under the head: change state, rewrite the symbol and move the head one square.

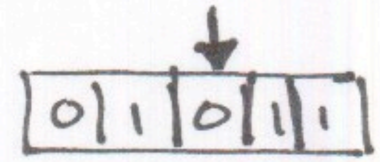


Infinite tape with squares containing tape symbols chosen from a finite alphabet

DATA STRUCTURE

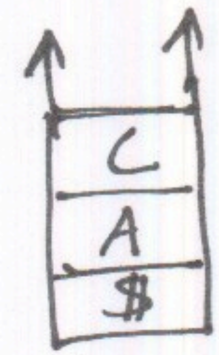
FSM

- THE INPUT STRING



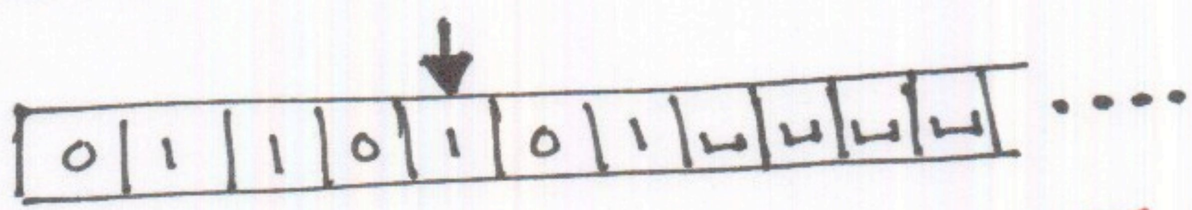
PDA

- THE INPUT STRING
- A STACK



TM

- A "TAPE"



SYMBOLS ~~FOR~~ FROM AN ALPHABET Σ

A SPECIAL BLANK SYMBOL $\sqcup$

INFINITE IN ONE DIRECTION

– BUT FILLED WITH BLANKS.

CURRENT POSITION

Turing-Machine Formalism

- A TM is described by:
  1. A finite set of *states* (Q).
  2. An *input alphabet* (Σ).
  3. A *tape alphabet* (Γ contains blank symbol and Σ).
  4. A *transition function* (δ).
     - $\delta: Q \times \Gamma \rightarrow Q \times \Gamma \times \{L, R\}$
  5. A *start state* (q_0, in Q).
  6. A *blank symbol* (in Γ- Σ, typically).
     - All tape except for the input is blank initially.
  7. 2 *final states*
     1. q_{accept} is the accept state
     2. q_{reject} is the reject state

The Transition Function

- Takes two arguments:
  1. A state, in Q.
  2. A tape symbol in Γ.

- δ(q, Z) is either undefined or a triple of the form (p, Y, D).
  - p is a state.
  - Y is the new tape symbol.
  - D is a *direction*, L or R.
  - if undefined the machine halts

Example: Turing Machine

- This TM scans its input right, looking for a 1.
- If it finds one, it changes it to a 0, goes to final state f, and accepts.
- If it reaches a blank, it changes it to a 1 and moves left.

Example: Turing Machine – (2)

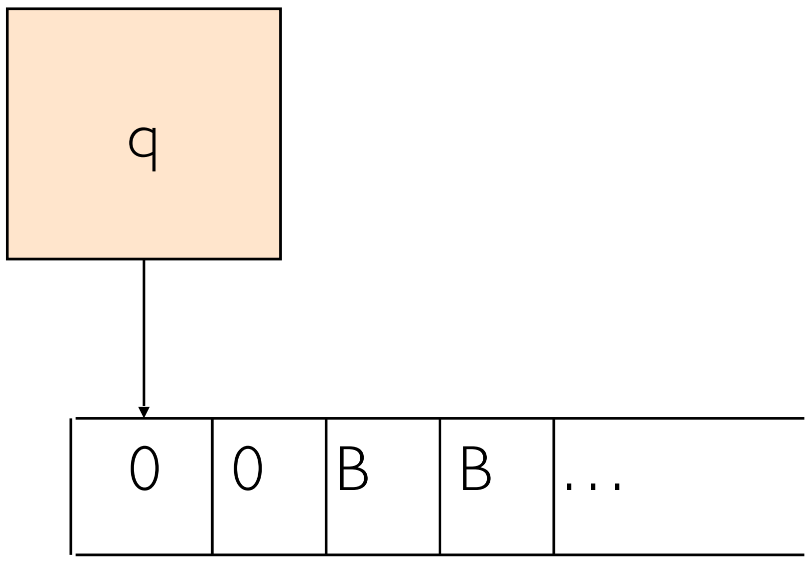
- States = {q (start), f (accept)}.
- Input symbols = {0, 1}.
- Tape symbols = {0, 1, B}.
- $\delta(q, 0) = (q, 0, R)$.
- $\delta(q, 1) = (f, 0, R)$.
- $\delta(q, B) = (q, 1, L)$.

Simulation of TM

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

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

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

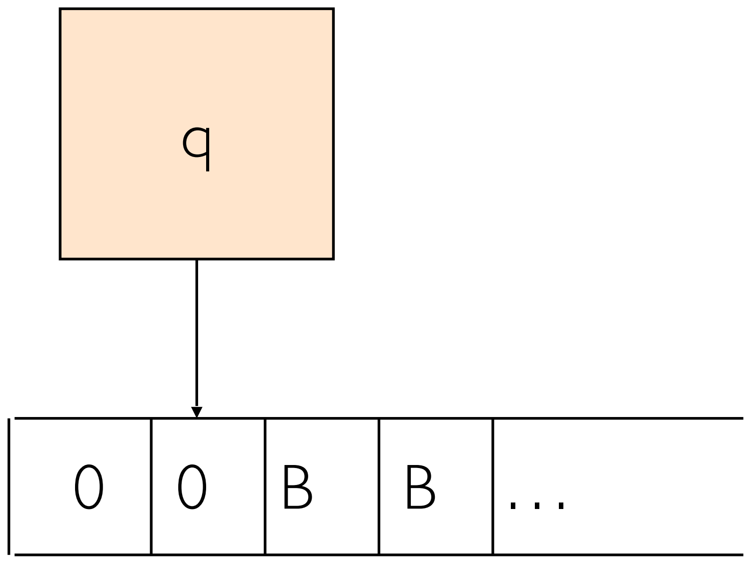


Simulation of TM

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

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

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

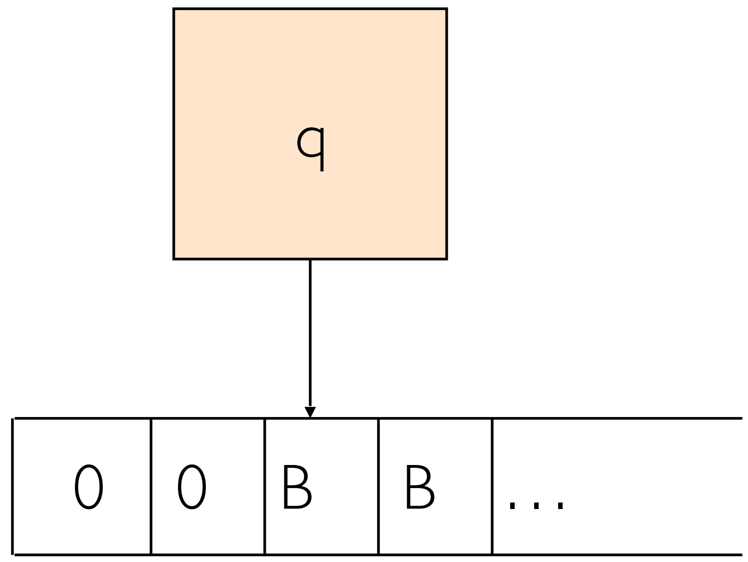


Simulation of TM

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

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

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

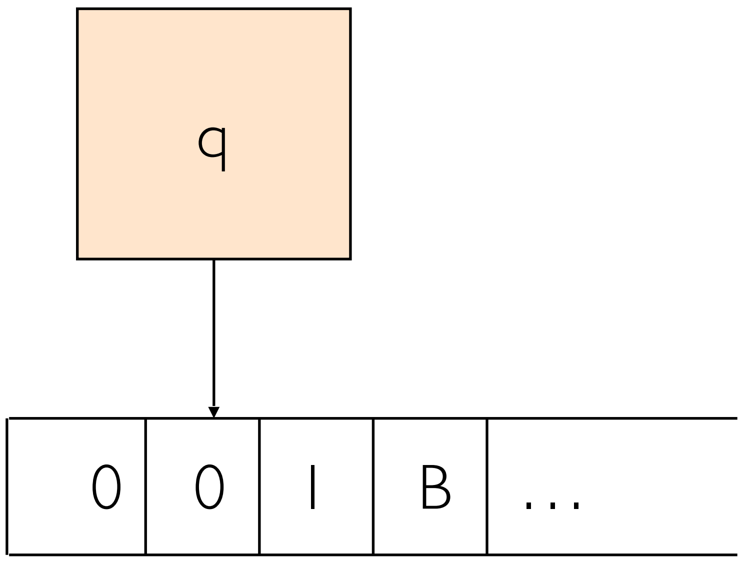


Simulation of TM

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

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

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

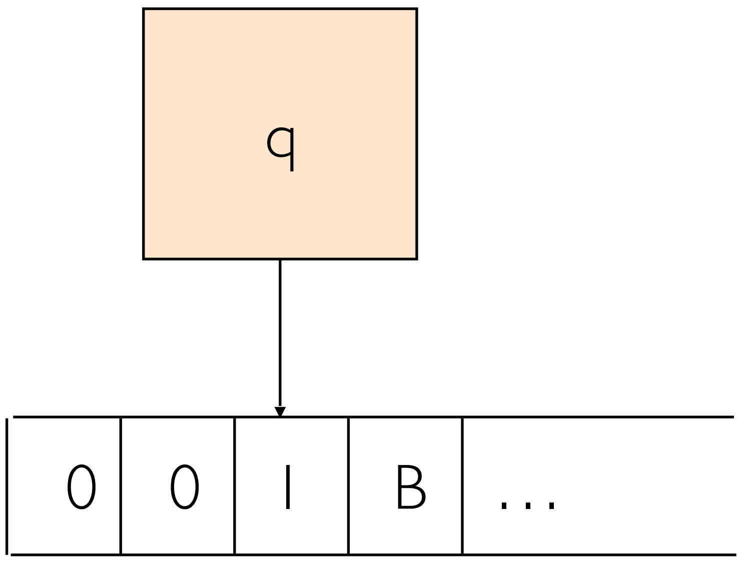


Simulation of TM

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

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

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

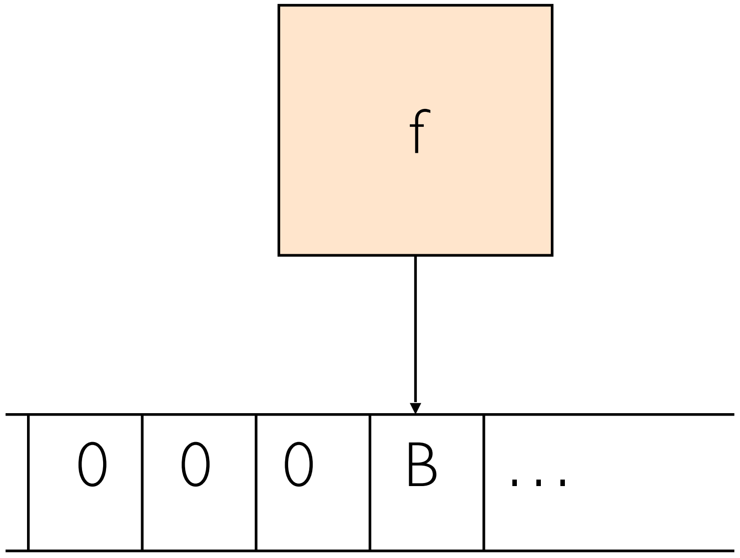


Simulation of TM

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

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

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



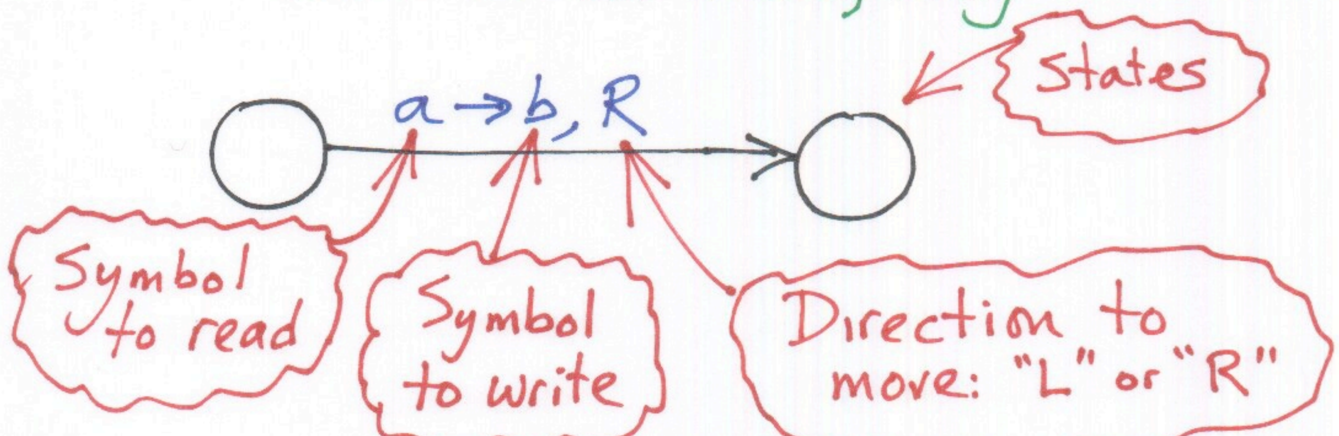
The TM halts and accepts.

Rules of Operation

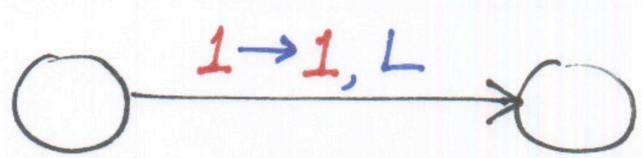
AT EACH STEP OF THE COMPUTATION:

- READ THE CURRENT SYMBOL
- UPDATE (i.e., write) THE SAME CELL.
- MOVE EXACTLY ONE CELL
  EITHER <u>LEFT</u> OR <u>RIGHT</u>.

(If we are at the left end of the tape and trying to move left, then do not move; stay at left end.)



Don't want to update the cell?
Just write the same symbol.



- FINAL STATES

  THE "ACCEPT" STATE
  THE "REJECT" STATE

  Exactly two final states

- COMPUTATION CAN...

  HALT AND "**ACCEPT**"

  (Whenever the machine enters the ACCEPT state, computation immediately HALTS.)

  HALT AND "**REJECT**"

  (Whenever the machine enters the REJECT state, computation immediately HALTS.)

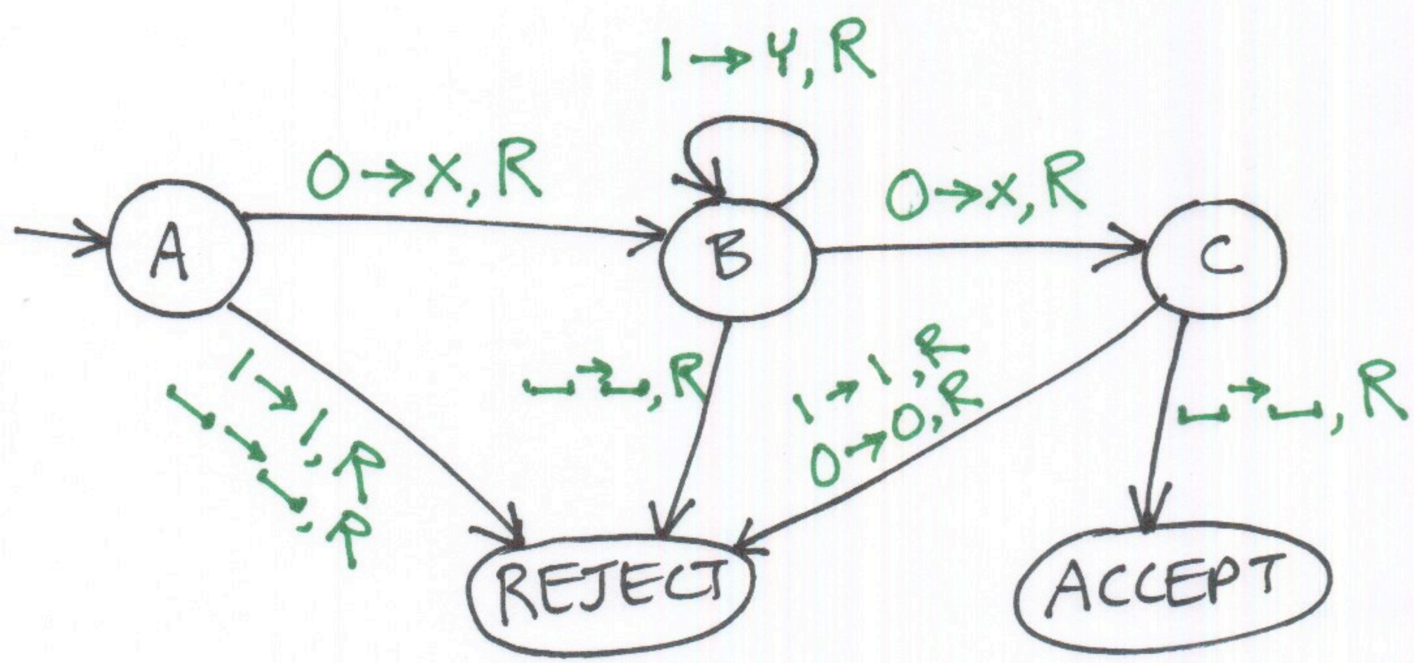
  "**LOOP**"

  (The machine fails to HALT.)

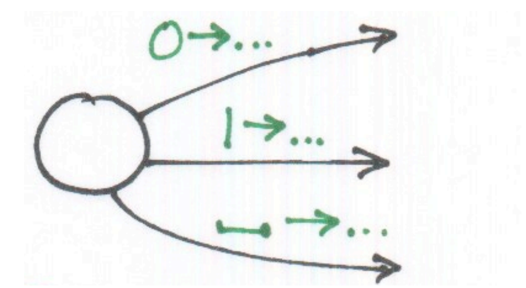
- THE TM IS **DETERMINISTIC**.

AN EXAMPLE

$L = 01^*0$



Is it deterministic?



HENCEFORTH:

IF AN EDGE IS MISSING...
ASSUME IT LEADS TO REJECT.

Turing Machine Example 1

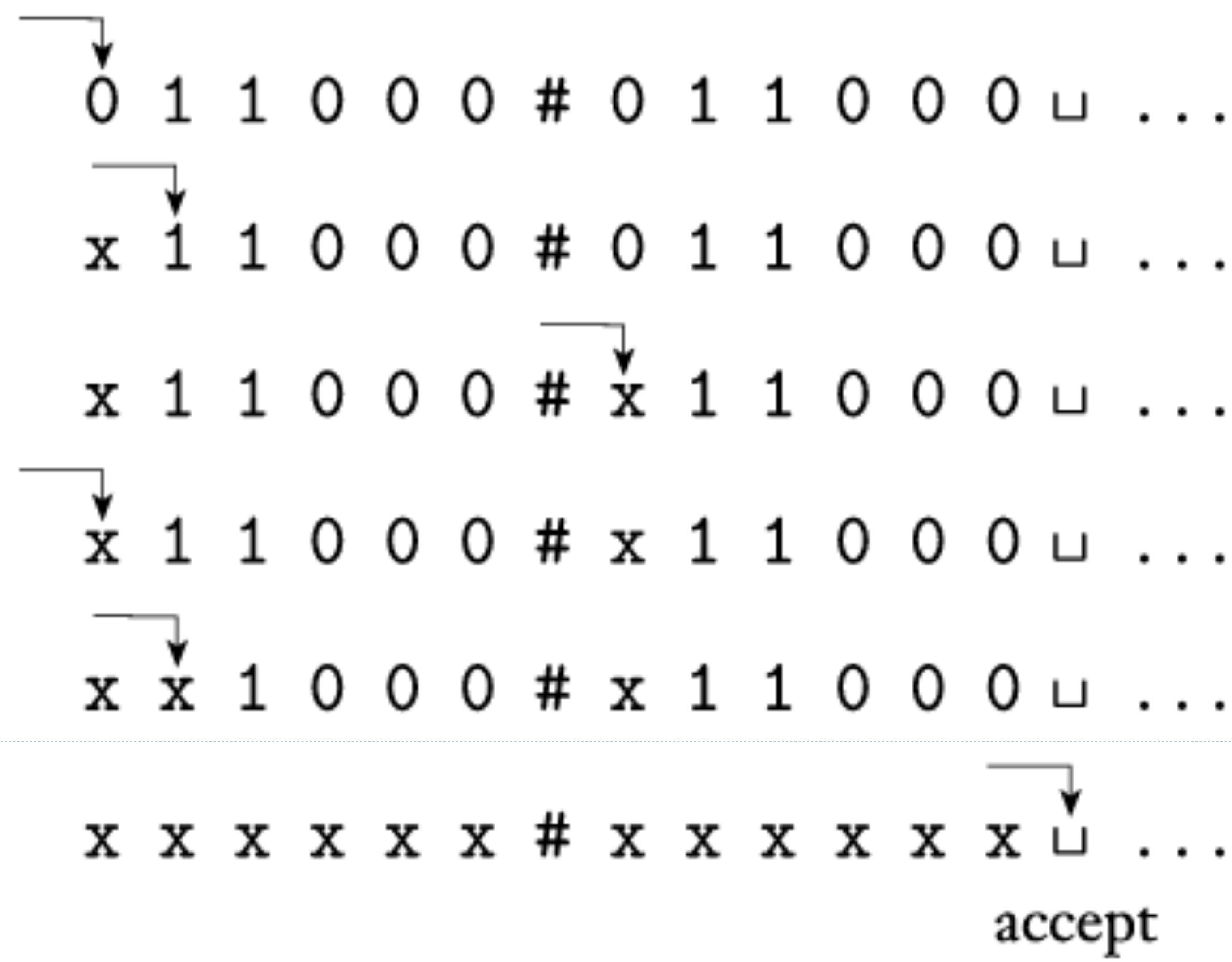
- M_1 recognizes $B = \{\ w\#w \mid w \in \{0, 1\}^*\ \}$

1. Zig-zag across the tape to corresponding positions on either side of the # symbol to check whether these positions contain the same symbol. If they do not, or if no # is found, *reject*. Cross off symbols as they are checked to keep track of which symbols correspond.
2. When all symbols to the left of the # have been crossed off, check for any remaining symbols to the right of the #. If any symbols remain, *reject*; otherwise, *accept*."

Turing Machine Example 1

- M_1 is a TM that recognizes
  $B = \{ w\#w \mid w \in \{0, 1\}^* \}$

Snapshots of M_1's tape



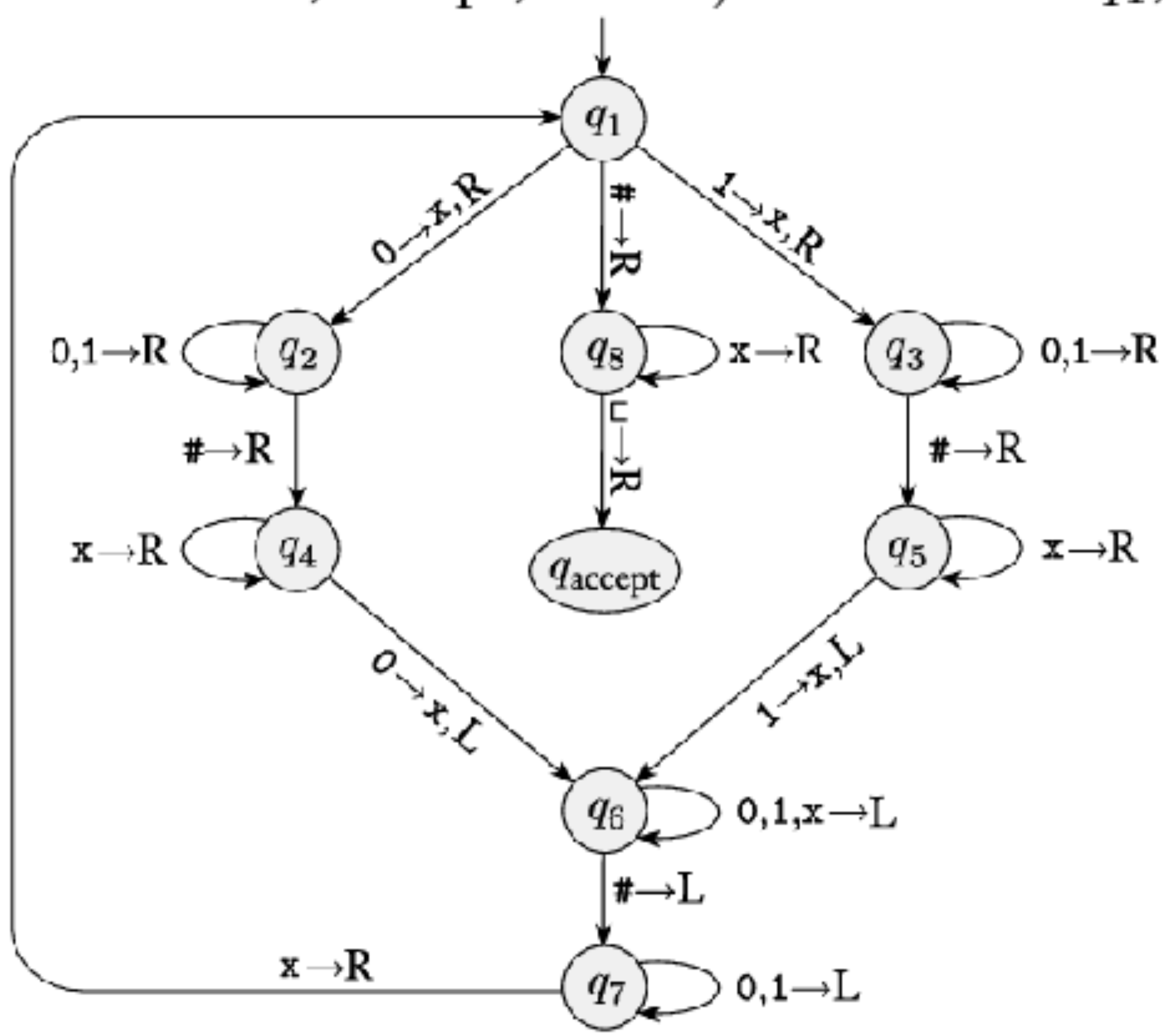
Turing Machine Example 1

- M_1 recognizes $B = \{\ w\#w \mid w \in \{0, 1\}^*\ \}$

$Q = \{q_1, \ldots, q_8, q_{\text{accept}}, q_{\text{reject}}\}$,

$\Sigma = \{0,1,\#\}$, and $\Gamma = \{0,1,\#,x,\sqcup\}$.

The start, accept, and reject states are q_1, q_{accept}, and q_{reject}, respectively.

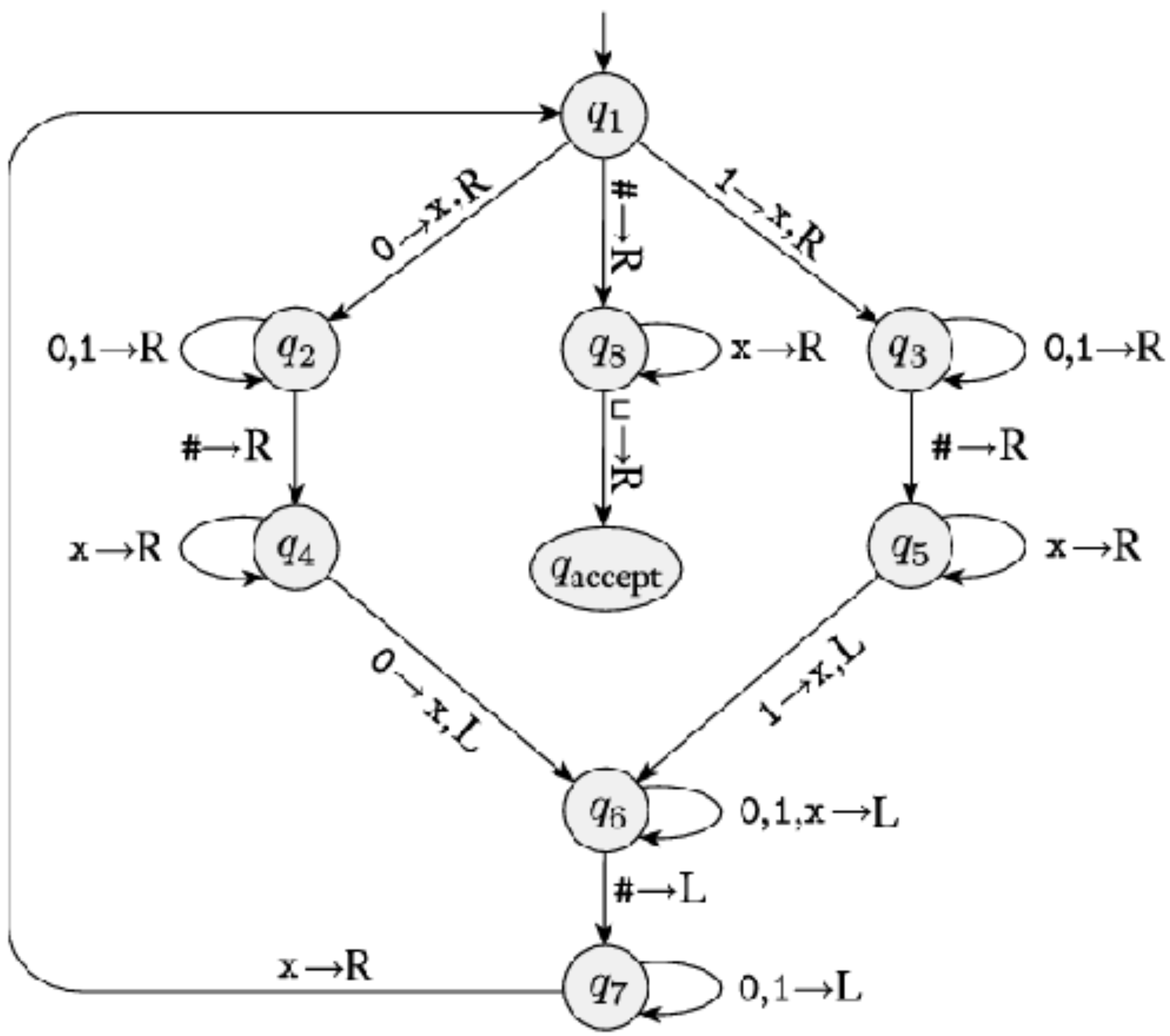


$0 \rightarrow$ x, R:
if current cell has a 0,
write an x and move right

x $\rightarrow$ R:
if current cell has an x,
write an x and move right
(equivalent to no write)

Turing Machine Example 1

- M_1 recognizes $B = \{ w\#w \mid w \in \{0, 1\}^* \}$



q_1: begin checking first symbol before the #

q_2, q_4 : look for 0, after the #

q_3, q_5 : look for 1, after the #

q_6, q_7 : cross-off matched symbol and backup to next symbol before #

q_8 : no symbols left before #, check that no symbols left after #

any omitted transitions go to q_{reject}