

# Computability Theory

The Church-Turing Machines  
Decidability  
Reducibility



## Introduction

### In chapter 1 :

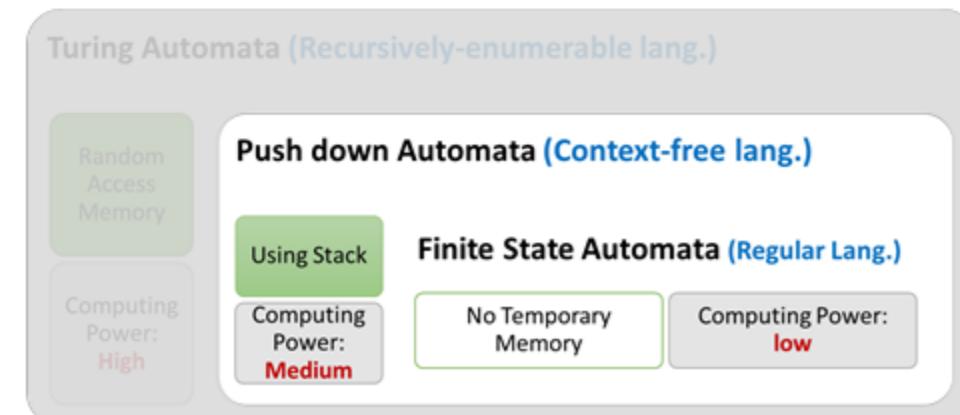
- We learned that
  - we can describe many **languages** ([regular languages](#)) using two equivalent methods:
    - finite automata (DFA, NFA)
      - They have a **small amount of memory**
    - regular expressions (RE)
- We found out
  - There are some **languages** that **cannot be described** using the methods above
    - Non-regular languages
    - Example:  $\{0^n1^n \mid n \geq 0\}$



## Introduction

### In chapter 2 :

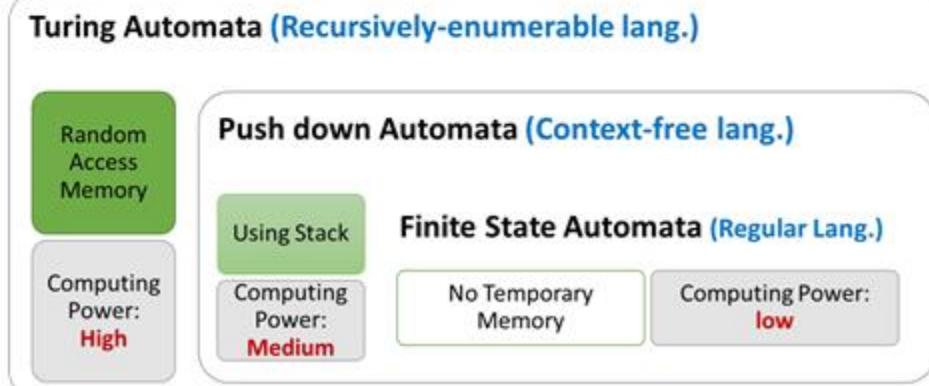
- We learned that
  - There is a **more powerful method** of describing languages:
    - called **context-free grammars (CFG)**
    - Associated languages with CFG: **context-free languages (CFL)**.
    - Associated automata: **Pushdown automata**
      - They have an **unlimited memory** that is usable only in the **last in, first out** manner of a **stack**.
  - We found out
    - There are some **languages** that **cannot be described** using the method above
      - **Non-contextfree languages**
      - Example:  $\{a^n b^n c^n \mid n \geq 0\}$



## Introduction

### In chapter 3 :

- We present a much **more powerful method** of describing languages:
  - called **Turing machine**
  - first proposed by **Alan Turing** in 1936
- Similar to a **finite automaton** but with an **unlimited and unrestricted memory**.
- Some languages that they are **beyond the capabilities** of FA and PDA models can be described using **Turing machine**.



## Chapter 3 : The Church–Turing Thesis

### 3.1 Turing Machines

Formal definition of a Turing machine

Examples of Turing machines

### 3.2 Variants of Turing Machines

Multitape Turing machines

Nondeterministic Turing machines

Enumerators

Equivalence with other models

### 3.3 The Definition of Algorithm

Hilbert's problem

Terminology for describing Turing machines

## Introduction

- In 1936 :
  - Church's Lambda calculus
  - Kleene's(general) recursive functions
  - Turing machines
  - Post's production systems
- All these systems were proved equivalent.
- Church and Turing then proposed equivalent theses.  
**"Church's Thesis"** , "**Turing's Thesis**", or the "**Church-Turing Thesis**".



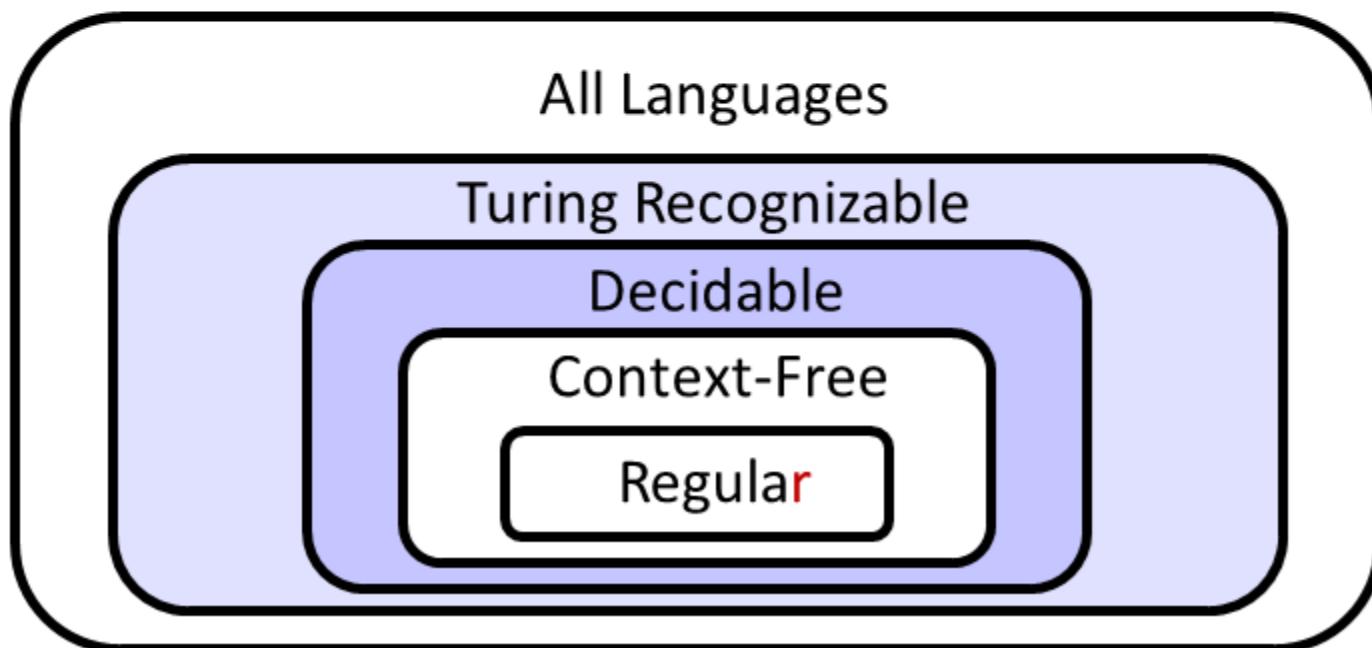
## Introduction

**Church-Turing Thesis:** is a hypothesis about the **nature of computable functions.**

- It states that  
a function on the natural numbers is computable by a human being following an algorithm, ignoring resource limitations, if and only if it is computable by a Turing machine.

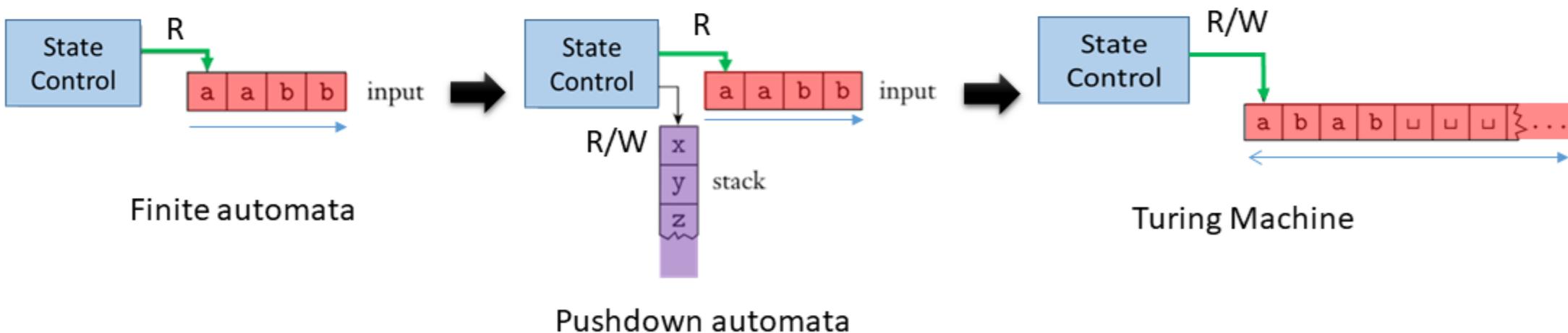
## A Turing machine

- is a **much more accurate model** of a **general purpose computer**.
- can do everything that a real computer can do.
- Even a Turing machine **cannot solve certain problems**.
  - these problems are **beyond** the theoretical **limits of computation**.



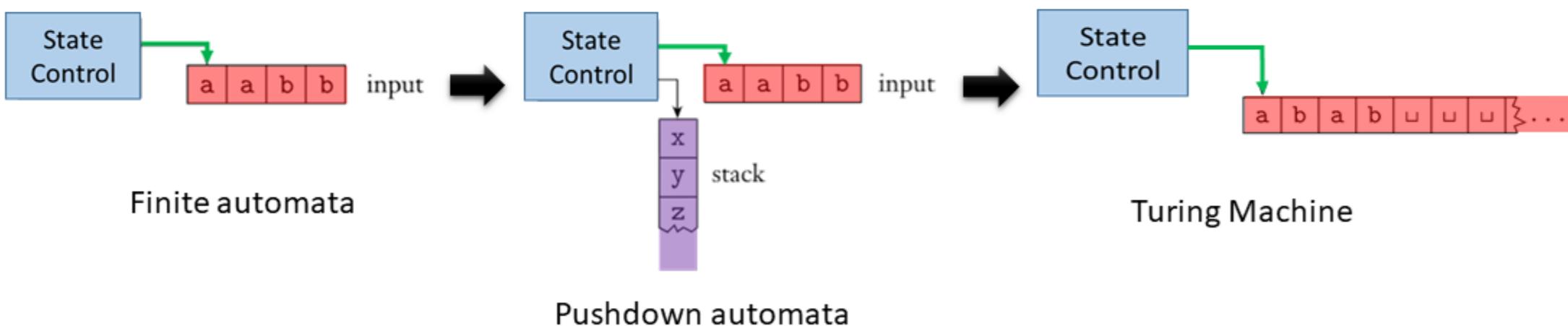
# TURING MACHINES

- A Turing machine model
  - uses an **infinite tape** (in one direction) as its **unlimited memory**.
  - has a **tape head** that can **read** and **write** symbols and **move around** on the tape.
- Initially the **tape** contains **only the input string** and is **blank everywhere else**.



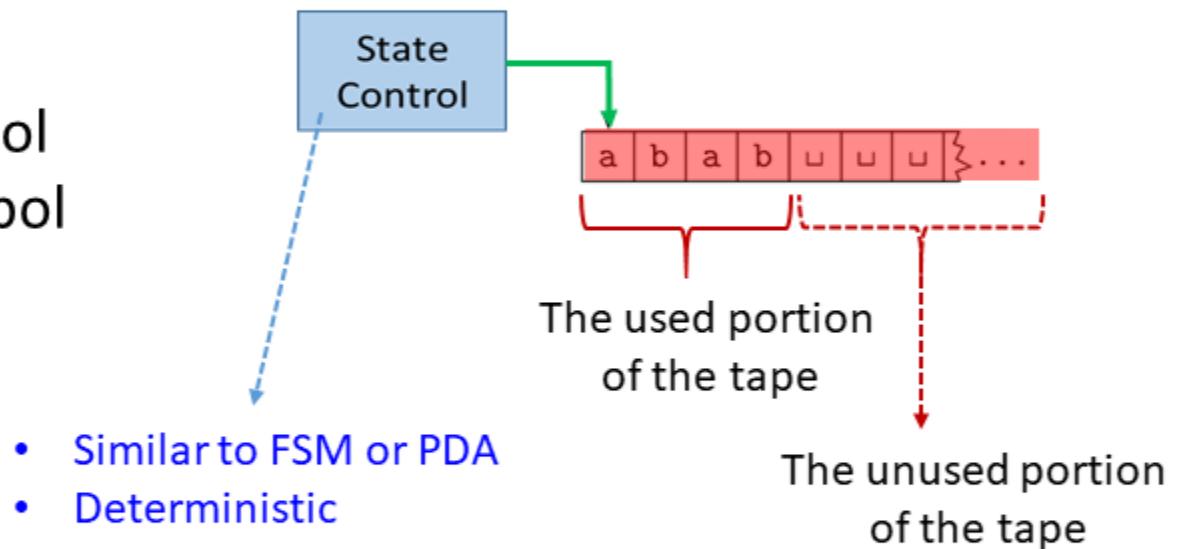
# TURING MACHINES

- **Tape Alphabet**
  - Typical:  $\Sigma = \{0,1\}$
  - Common:  $\Sigma = \{0,1,a,b,x,y,\#, \$\}$
  - The “Blank” symbol is special and  $\sqcup \notin \Sigma$ .



# TURING MACHINES

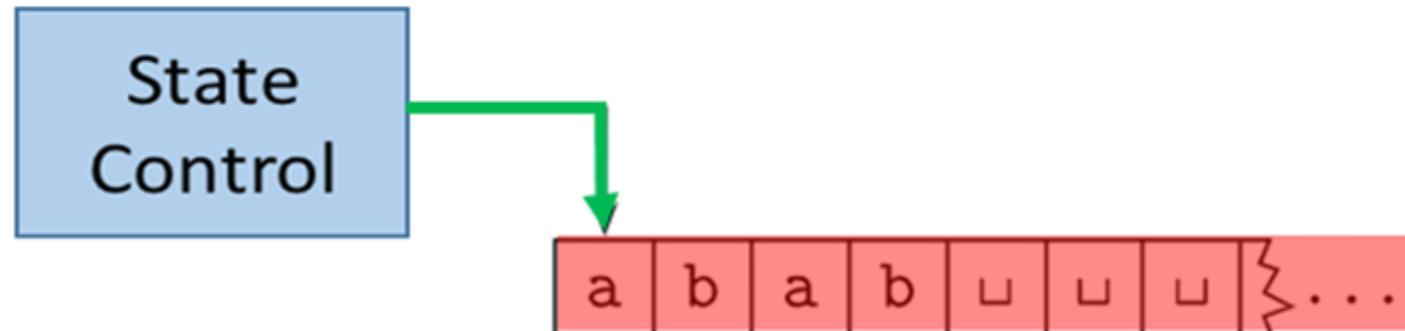
- The Tape head
    - shows the **current position**
    - initially at the **leftmost** square
    - can **move left or right**
    - can **read** the current symbol
    - can **write** the current symbol



# TURING MACHINES

## The differences between finite automata and Turing machines

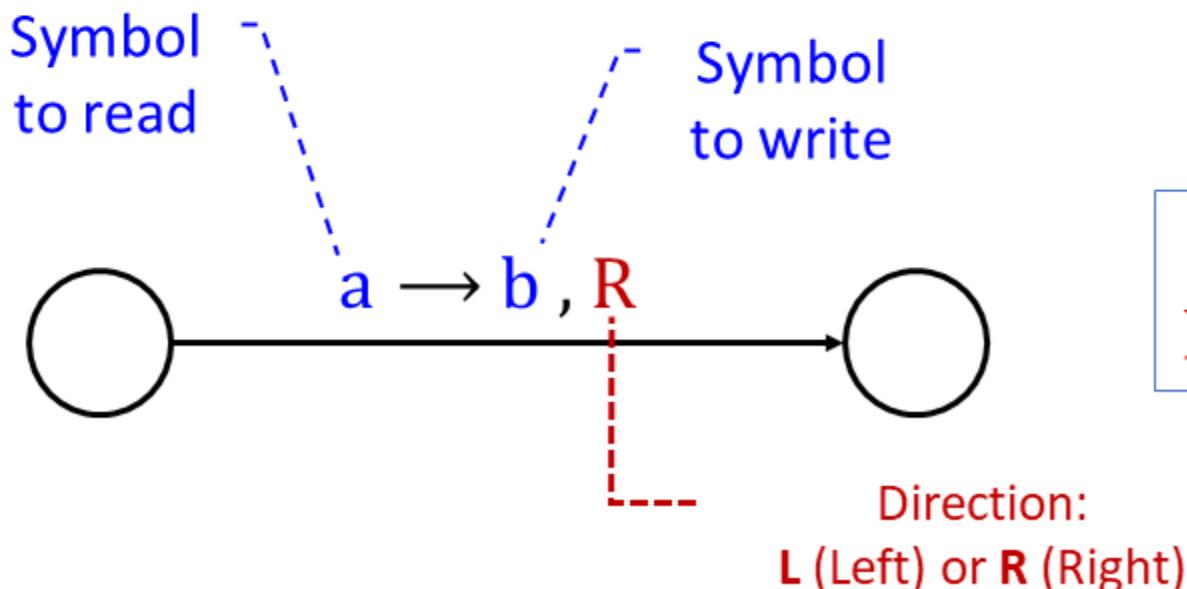
1. A Turing machine can both **write** on the tape and **read** from it.
2. The **read–write head** can **move** both to the **left** and to the **right**.
3. The **tape** is **infinite**.
4. The special states for **rejecting** and **accepting** take effect immediately.



## Rules of Operation

At each step of the computation:

1. Read the current symbol
2. Update (write) the same cell.
3. Move exactly one cell either left or right.
  - At the left end of the tape, there is no move left and the head will stay at left end.



$0 \rightarrow 0, R$

No update and move to right

## State Control

- is similar to FSM.
- It has
  - A **unique Initial state**
  - 1. Exactly two final states
    - 1. The “**accept**” state
    - 2. The “**reject**” state

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.

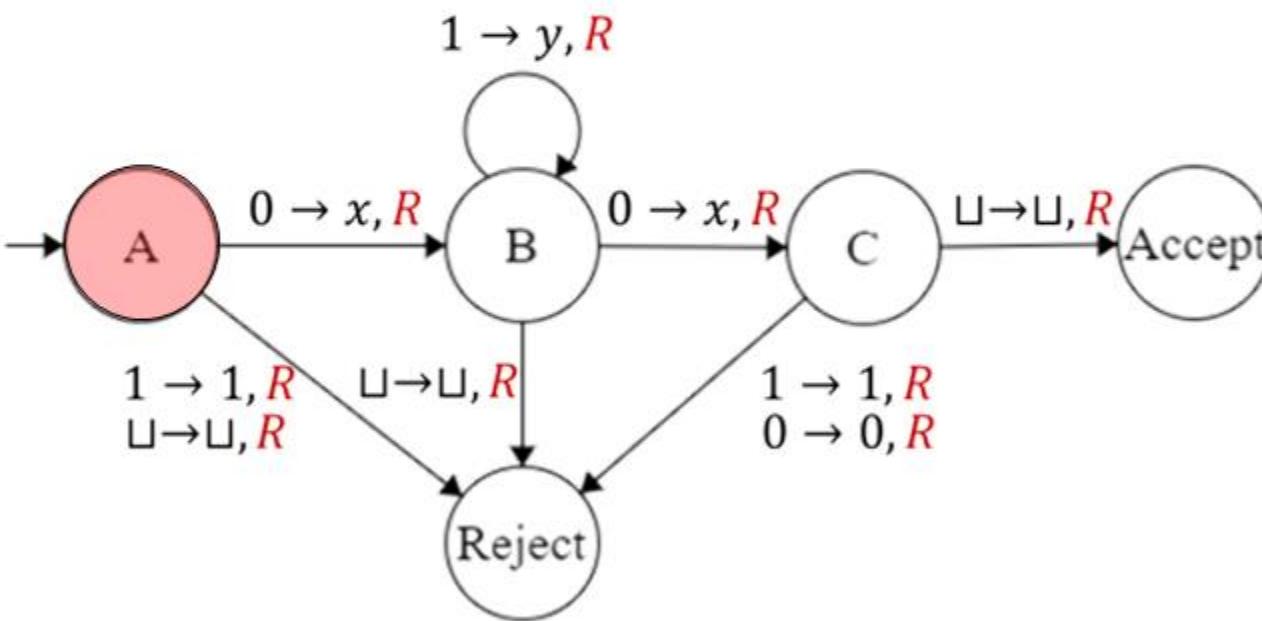
## State Control

- is similar to FSM.
- It has
  - A **unique Initial state**
  - Exactly **two final states**
    1. The “**accept**” state
    2. The “**reject**” state

## Example 1

Show that the following Turing machine M1 can be used for testing membership in the language  $A = \{w = 01^*0 | w \in \{0,1\}^*\}$ .

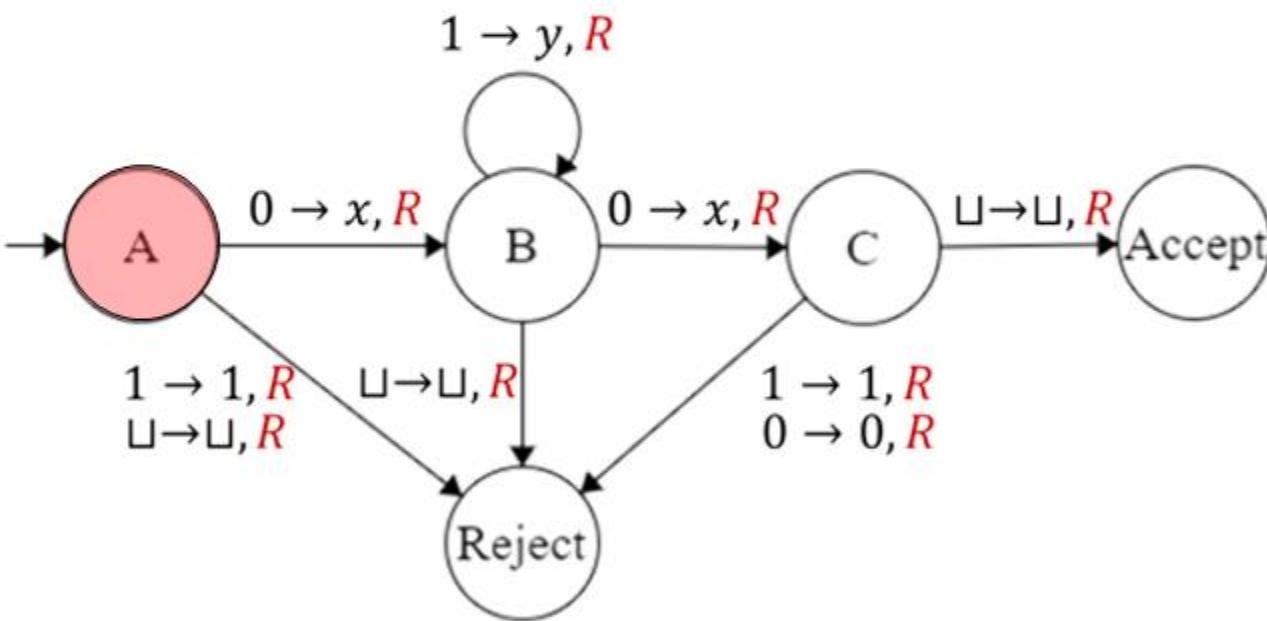
if  $w = 01110$



If an edge is missing, assume it leads to **Reject**.

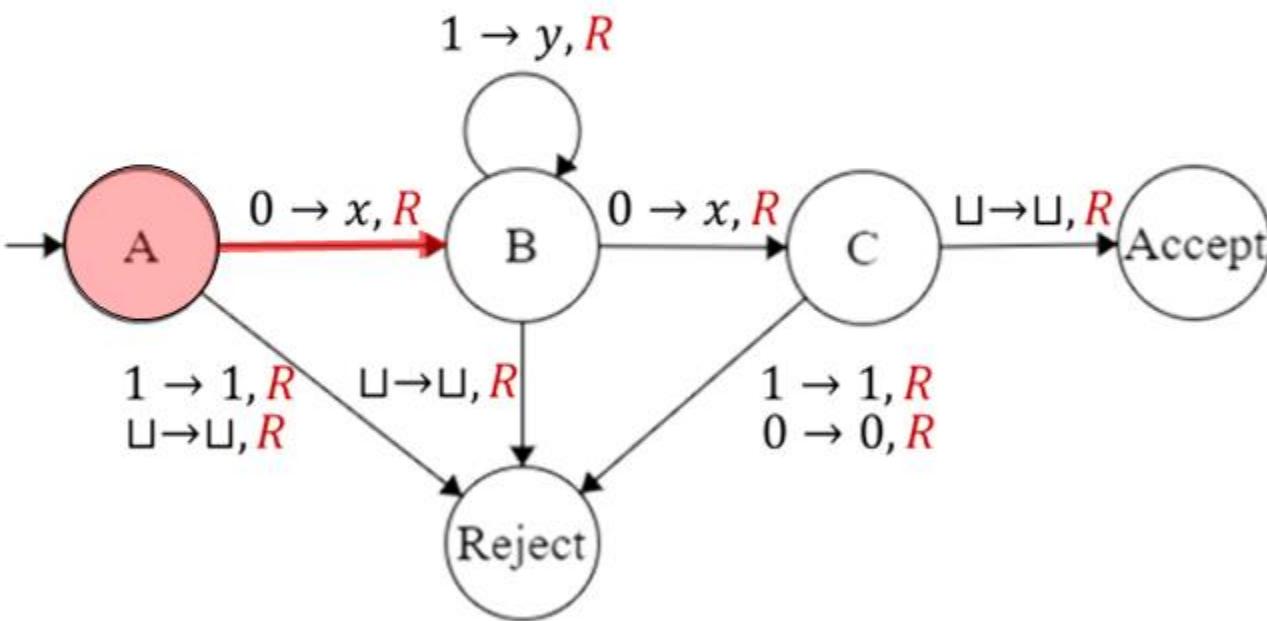
## Example 1

Show that the following Turing machine M1 can be used for testing membership in the language  $A = \{w = 01^*0 \mid w \in \{0,1\}^*\}$ .



## Example 1

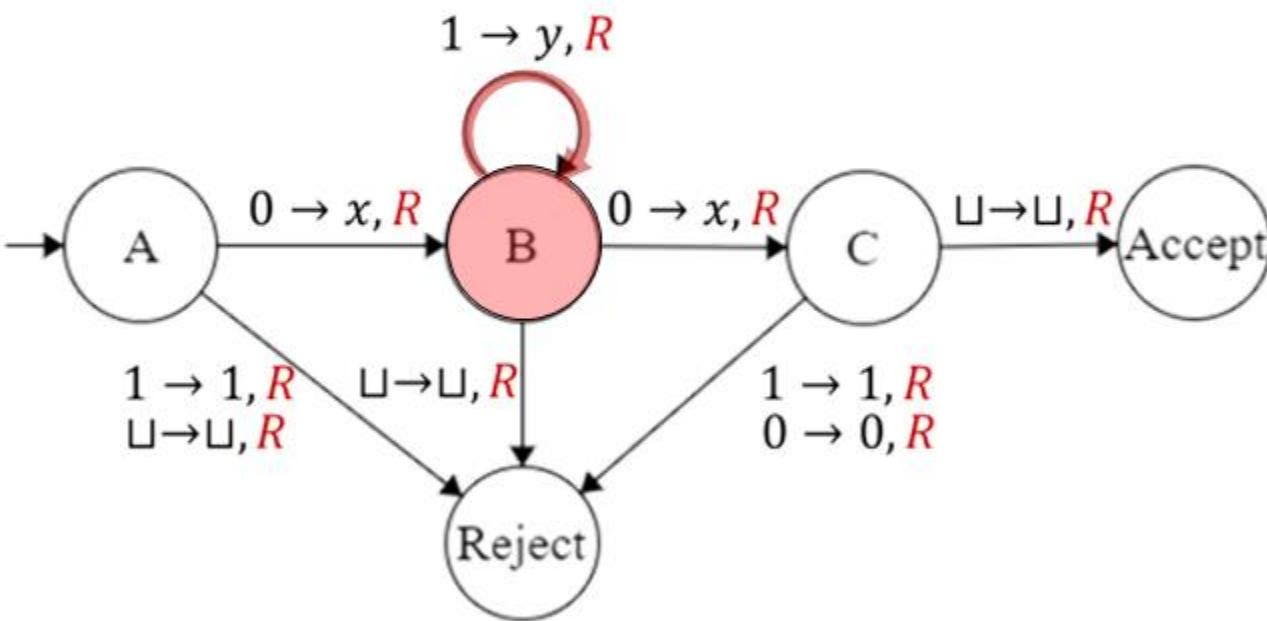
Show that the following Turing machine M1 can be used for testing membership in the language  $A = \{w = 01^*0 \mid w \in \{0,1\}^*\}$ .



$$0 \rightarrow x, R$$

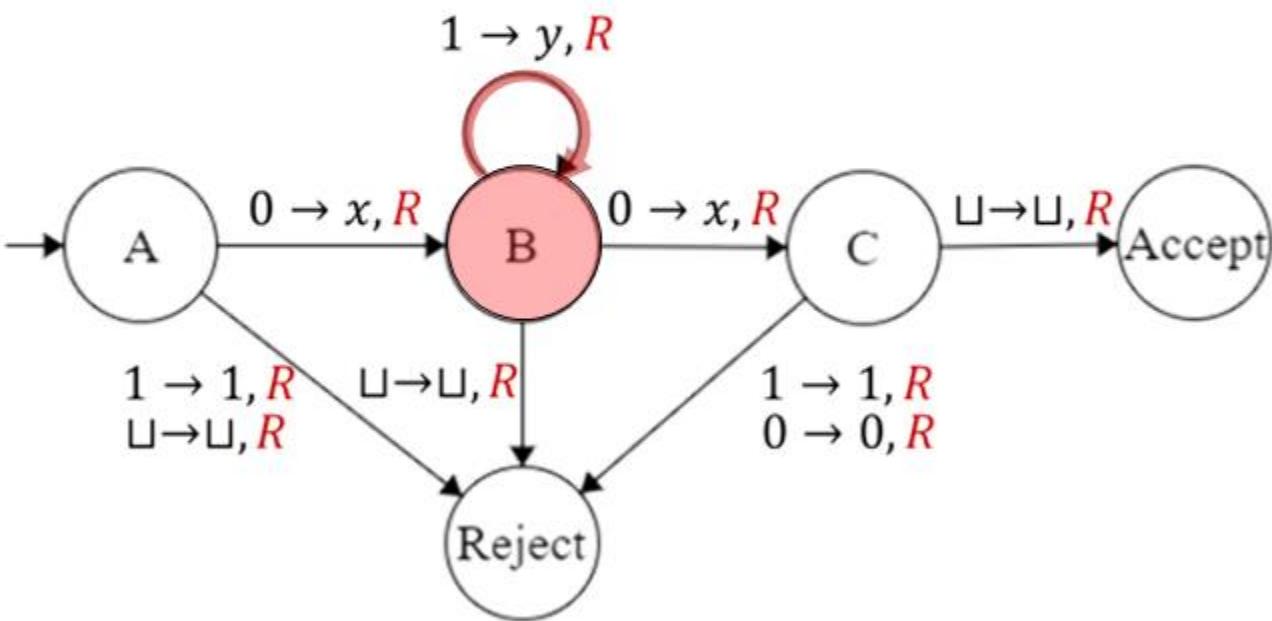
## Example 1

Show that the following Turing machine M1 can be used for testing membership in the language  $A = \{w = 01^*0 \mid w \in \{0,1\}^*\}$ .



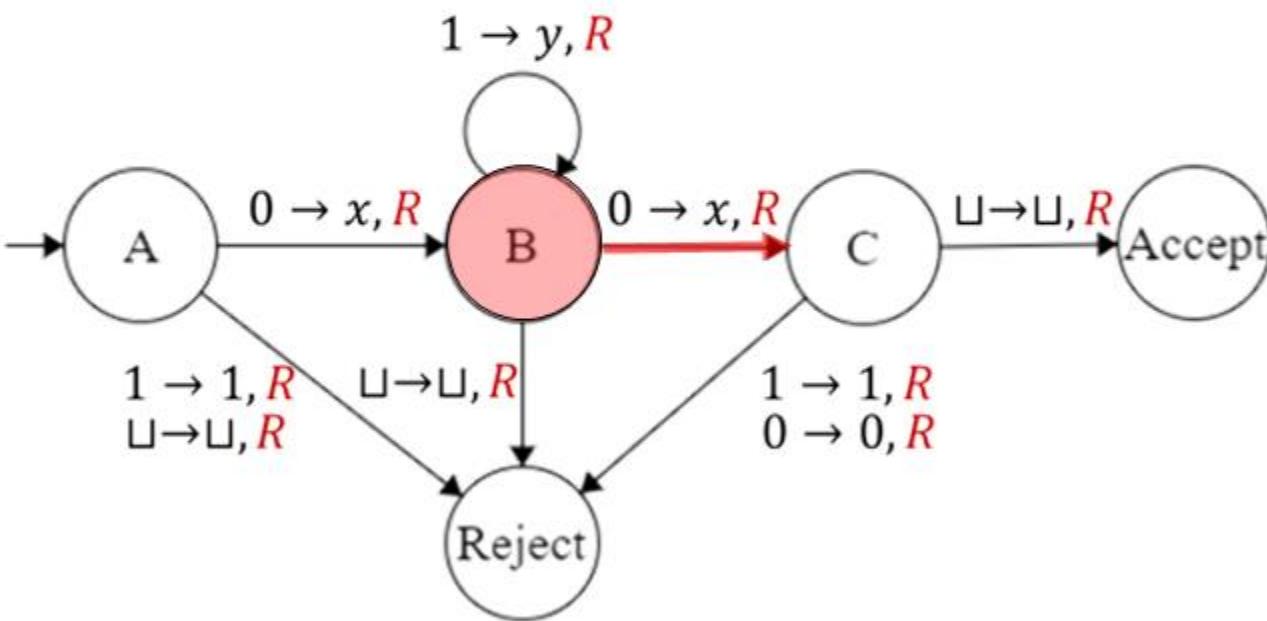
## Example 1

Show that the following Turing machine M1 can be used for testing membership in the language  $A = \{w = 01^*0 \mid w \in \{0,1\}^*\}$ .



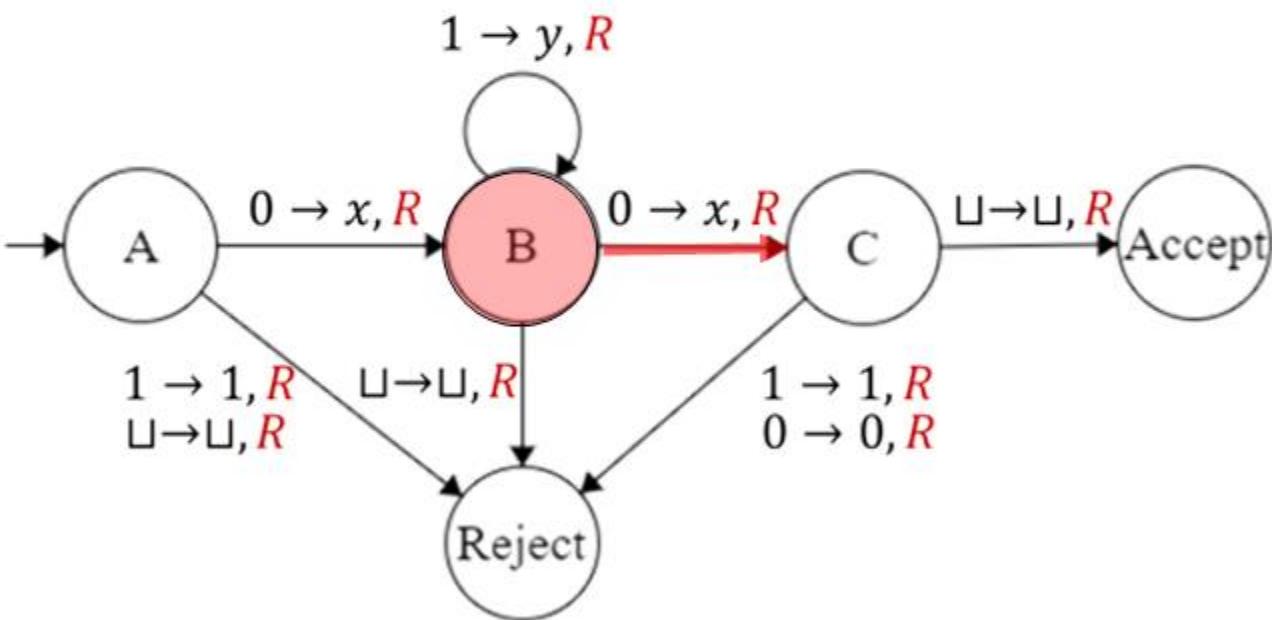
## Example 1

Show that the following Turing machine M1 can be used for testing membership in the language  $A = \{w = 01^*0 \mid w \in \{0,1\}^*\}$ .



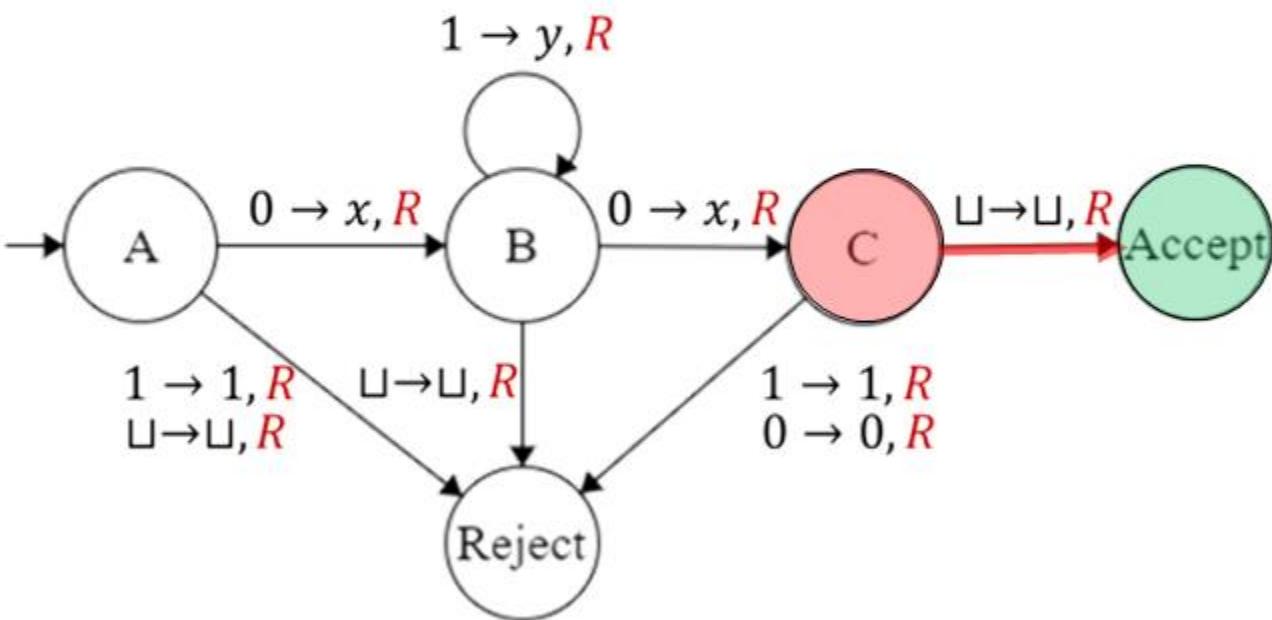
## Example 1

Show that the following Turing machine M1 can be used for testing membership in the language  $A = \{w = 01^*0 \mid w \in \{0,1\}^*\}$ .



## Example 1

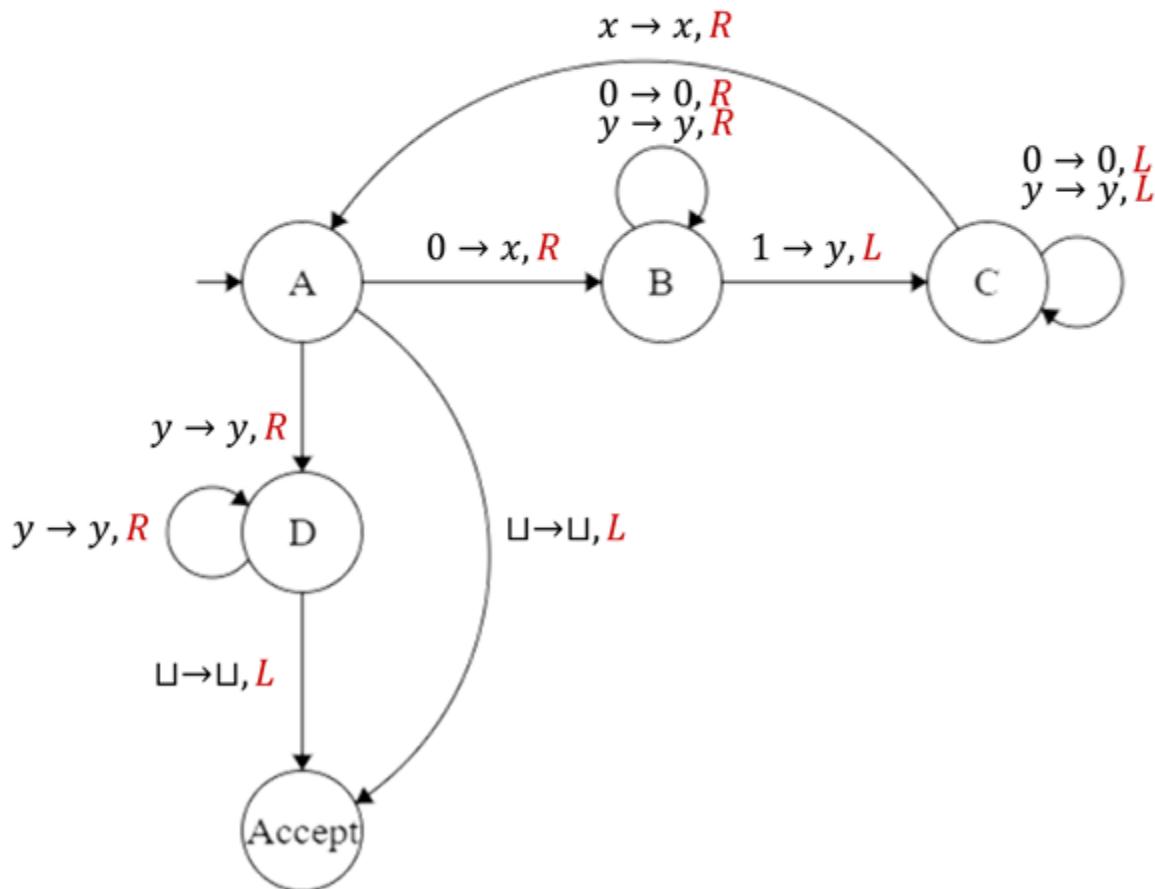
Show that the following Turing machine M1 can be used for testing membership in the language  $A = \{w = 01^*0 \mid w \in \{0,1\}^*\}$ .



## Chapter 3

### Example 2

The following Turing machine M2 has been designed for testing membership in the language  $B = \{w = 0^n 1^n | w \in \{0,1\}^*, n \geq 0\}$ .



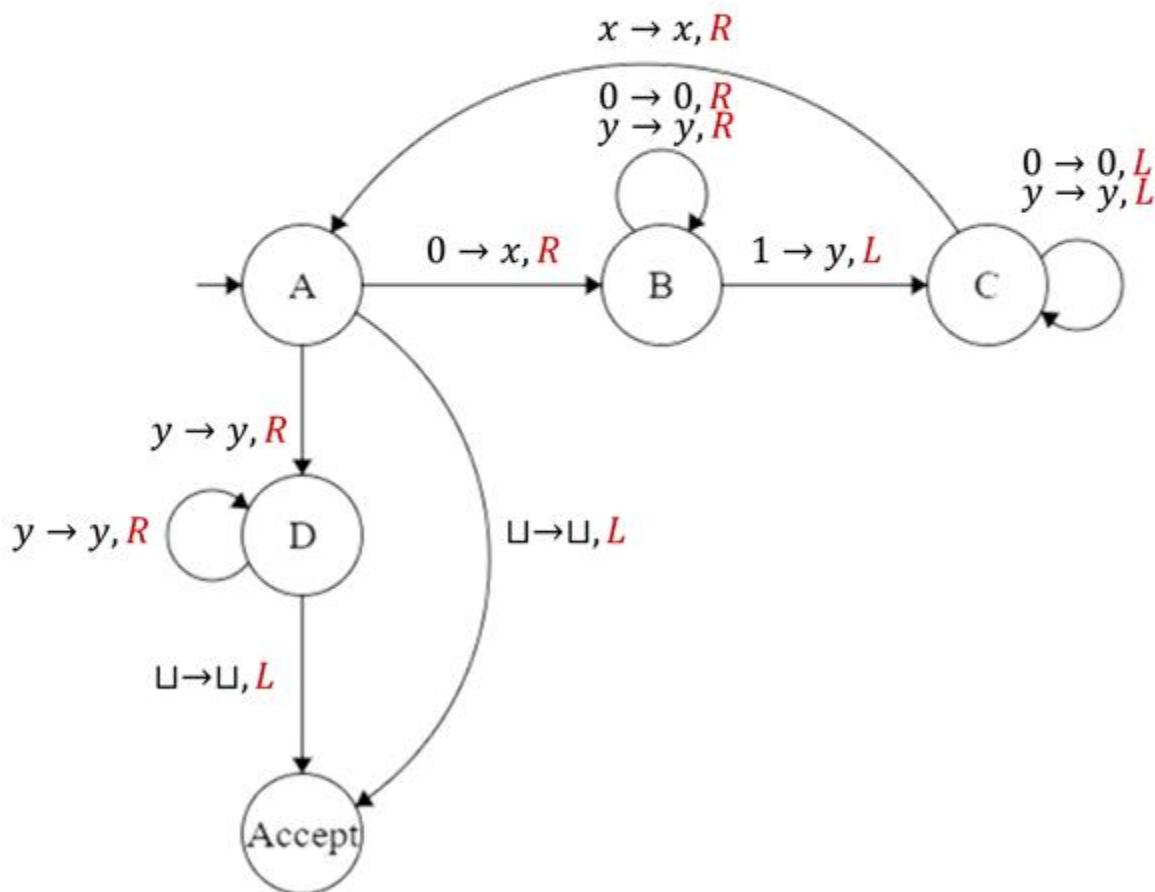
#### Algorithm:

- Change 0 to x
- Move right to first 1
  - If none: Reject
- Change 1 into y
- Move left to leftmost 0
- Repeat until no more 0s
- Make sure no more 1 remain.

- Does it work?
- Does it contain bugs?

## Example 2

The following Turing machine M2 has been designed for testing membership in the language  $B = \{w = 0^n 1^n | w \in \{0,1\}^*, n \geq 0\}$ .



### Computation history

0 0 0 0 1 1 1 1  
X 0 0 0 1 1 1 1  
X 0 0 0 Y 1 1 1 1  
0 X 0 0 Y 1 1 1 1

↓\*

X X X X Y Y Y Y  
X X X X Y Y Y Y

### Example 3

- Construct a Turing machine M1 for testing membership in the language

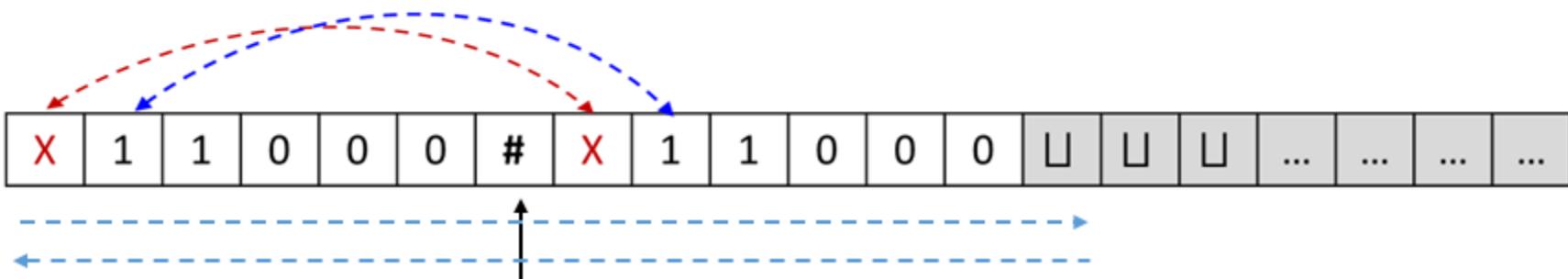
$$B = \{w\#w \mid w \in \{0,1\}^*\}.$$

- The TM M1 should **accept** if its input is a member of B and **reject** otherwise.

## Example 3

$$B = \{w\#w \mid w \in \{0,1\}^*\}.$$

- The input is too long for you to remember it all,
- But you are allowed to move **back and forth** over the input and make marks on it.
- **Strategy:**
  - zig-zag to the corresponding places on the two sides of the # and determine whether they match.
  - Place marks on the tape to keep track of which places correspond.



## TURING MACHINES : Example

- In summary,  $M_1$ 's algorithm is as follows.  $B = \{w\#w \mid w \in \{0,1\}^*\}$ .

$M_1$  = “On input string  $w$ :

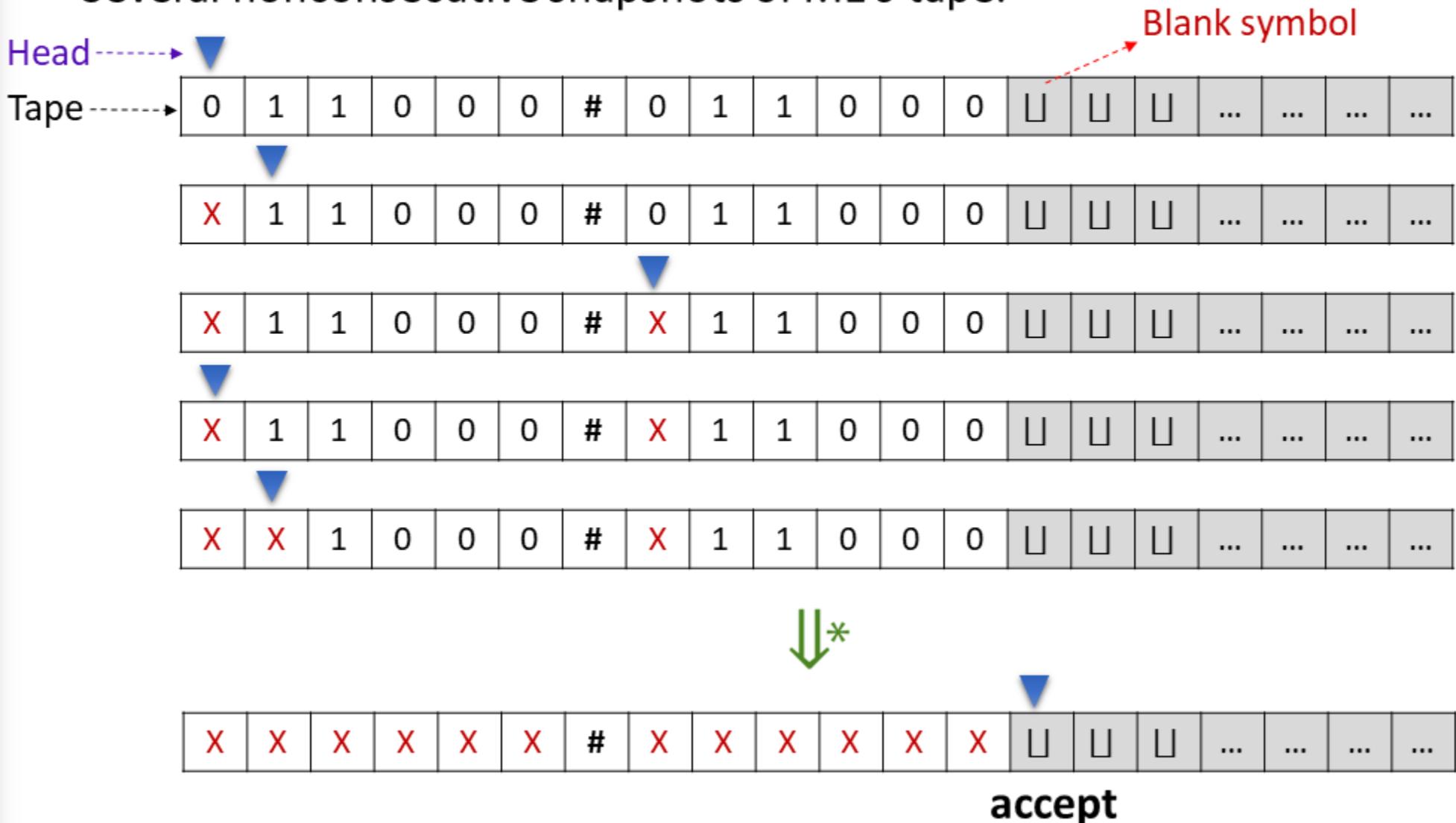
1. Zig-zag across the tape to corresponding positions on either side of the # symbol to check whether these positions contain the same symbol. If they do not, or if no # is found, *reject*. Cross off symbols as they are checked to keep track of which symbols correspond.
2. When all symbols to the left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

0	1	1	0	0	0	#	0	1	1	0	0	0	U	U	U	...	...	...	...
---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	-----	-----	-----	-----

### Chapter 3

## TURING MACHINES : Example

- Several nonconsecutive snapshots of M1's tape.



## FORMAL DEFINITION OF A TURING MACHINE

A *Turing machine* is a 7-tuple,  $(Q, \Sigma, \Gamma, \delta, q_0, q_{\text{accept}}, q_{\text{reject}})$ , where  $Q, \Sigma, \Gamma$  are all finite sets and

## FORMAL DEFINITION OF A TURING MACHINE

A **Turing machine** is a 7-tuple,  $(Q, \Sigma, \Gamma, \delta, q_0, q_{\text{accept}}, q_{\text{reject}})$ , where  $Q, \Sigma, \Gamma$  are all finite sets and

1.  $Q$  is the set of states,
2.  $\Sigma$  is the input alphabet not containing the **blank symbol**  $\sqcup$ ,
3.  $\Gamma$  is the tape alphabet, where  $\sqcup \in \Gamma$  and  $\Sigma \subseteq \Gamma$ ,

## FORMAL DEFINITION OF A TURING MACHINE

A **Turing machine** is a 7-tuple,  $(Q, \Sigma, \Gamma, \delta, q_0, q_{\text{accept}}, q_{\text{reject}})$ , where  $Q, \Sigma, \Gamma$  are all finite sets and

1.  $Q$  is the set of states,
2.  $\Sigma$  is the input alphabet not containing the **blank symbol**  $\sqcup$ ,
3.  $\Gamma$  is the tape alphabet, where  $\sqcup \in \Gamma$  and  $\Sigma \subseteq \Gamma$ ,
4.  $\delta: Q \times \Gamma \longrightarrow Q \times \Gamma \times \{\text{L}, \text{R}\}$  is the transition function,
5.  $q_0 \in Q$  is the start state,
6.  $q_{\text{accept}} \in Q$  is the accept state, and 
$$\delta(q, a) = (r, b, \text{L})$$
7.  $q_{\text{reject}} \in Q$  is the reject state, where  $q_{\text{reject}} \neq q_{\text{accept}}$ .

## COMPUTATION IN TURING MACHINE

A Turing Machine  $M = (Q, \Sigma, \Gamma, \delta, q_0, q_{accept}, q_{reject})$  computes as follows:

- **Initial Configuration:** Initially, M receives its input  $w = w_1 w_2 \dots w_n \in \Sigma^*$  on the leftmost  $n$  squares of the tape, and the rest of the tape is blank.
- **Start :** The head starts on the **leftmost square** of the tape.
  - the **first blank** appearing on the tape marks the **end of the input**.
- **Computation process:** Once M has started,
  - the **computation proceeds according to the rules** described by the **transition function**.

## COMPUTATION IN TURING MACHINE (Cont.)

- If  $M$  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 indicates L.
- **Halt:** The **computation continues until**
  - it enters either the **accept or reject states**, at which point it halts.
  - If neither occurs, **M goes on forever. (LOOP)**

## CONFIGURATION OF THE TURING MACHINE

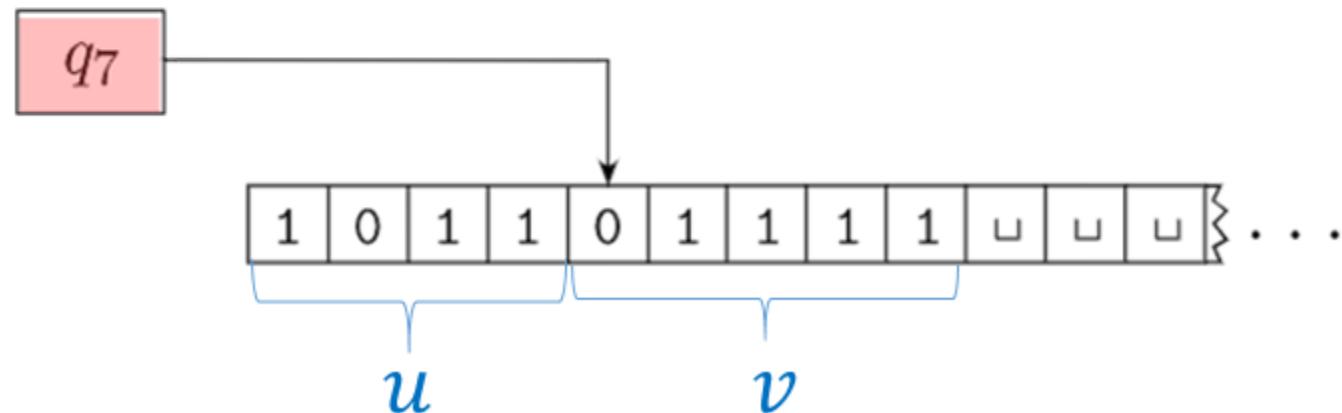
- As a Turing machine computes, changes occur in
  - the **current state**,
  - the **current tape contents**, and
  - the **current head location**.
- A **setting of these three items** is called a **configuration** of the Turing machine.

## CONFIGURATION OF THE TURING MACHINE

- For a state  $q$  and two strings  $u$  and  $v$  over the tape alphabet  $\Gamma$ ,
  - we write  $uqv$  for the configuration where
    - $q$  : the current state is,
    - $uv$  : the current tape contents
    - the current head location : is the first symbol of  $v$ .

- Example:  $1011\mathbf{q}_7\mathbf{0}1111$

$u$                $v$



## CONFIGURATION OF THE TURING MACHINE

- Suppose that we have
  - Tape symbols  $a, b$ , and  $c$  in  $\Gamma$ ,
  - Strings  $u$  and  $v$  in  $\Gamma^*$
  - states  $q_i$  and  $q_j$ .
  - two configurations  $C_1 = uaq_i bv$  and  $C_2 = uq_j acv$
- Say that
- if (for a leftward move)
  - Say that
  - if (for a rightward move)

$uaq_i bv$  yields  $uq_j acv$

$\delta(q_i, b) = (q_j, c, L)$ .

$uaq_i bv$  yields  $uacq_j v$

$\delta(q_i, b) = (q_j, c, R)$ .

$C_1 \rightarrow C_2$

## CONFIGURATION OF THE TURING MACHINE

- **Special cases:** when the **head** is at **one of the ends** of the configuration.
  - **the left-hand end:**
    - if the transition is **left-moving** :  $q_i b \nu$  yields  $q_j c \nu$ 
      - prevent the machine from going off the left-hand end of the tape
    - if the transition is **right-moving** :  $q_i b \nu$  yields  $c q_j \nu$ .

## CONFIGURATION OF THE TURING MACHINE

- **Special cases:** when the **head** is at **one of the ends** of the configuration.
  - **the right-hand end:**
    - if the transition is **left-moving** :
      - we can handle this case as before
    - if the transition is **right-moving** :  $uaq_i$  is equivalent to  $uaq_i \sqcup$ .
      - we can handle this case as before with the head no longer at the right-hand end

## CONFIGURATION OF THE TURING MACHINE

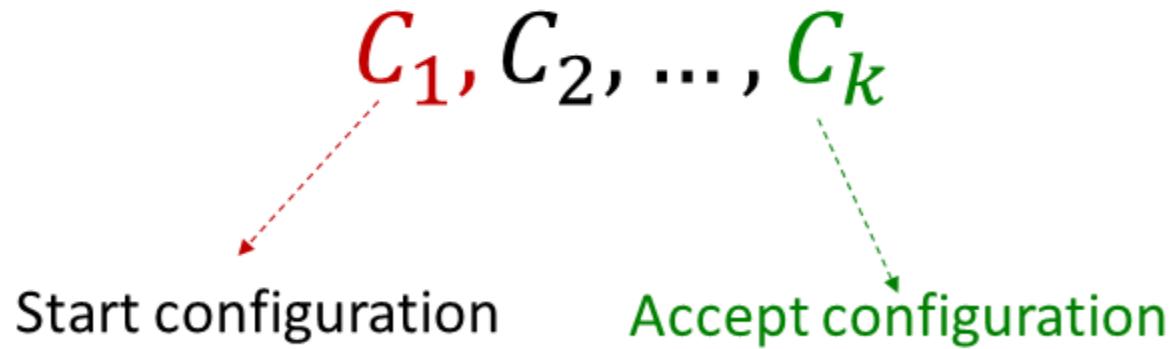
Different configurations of TM  $M$  on string  $w$

- **Start configuration** ( $q_0 w$ ):
  - indicates that the machine is in the **start state**  $q_0$  with its **head** at the **leftmost position** on the tape configuration.
- **Accepting configuration:**
  - The state of the configuration is  $q_{accept}$
- **Rejecting configuration:**
  - state of the configuration is  $q_{reject}$
- **Halting configurations:**
  - Accepting and rejecting configurations

## CONFIGURATION OF THE TURING MACHINE

A Turing machine  $M$  accepts input  $w$  if a sequence of configurations  $C_1, C_2, \dots, C_k$  exists, where

1.  $C_1$  is the start configuration of  $M$  on input  $w$ ,
2. each  $C_i$  yields  $C_{i+1}$ , and
3.  $C_k$  is an accepting configuration.

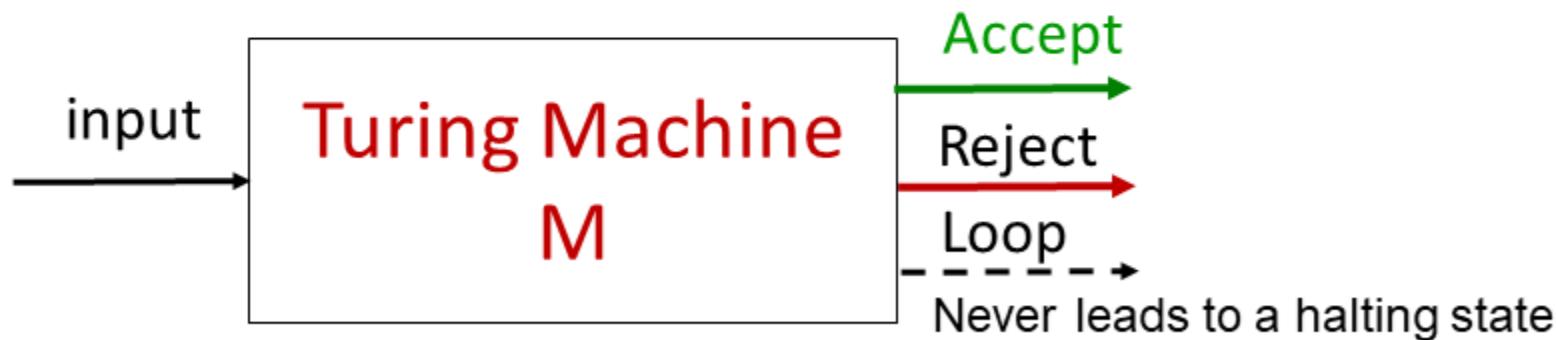


## Definition: Turing-recognizable

- The collection of strings that TM  $M$  accepts is the language of  $M$ , or the language recognized by  $M$ , denoted  $L(M)$ .

## Definition: Turing-recognizable

- The collection of strings that TM  $M$  accepts is the language of  $M$ , or the language recognized by  $M$ , denoted  $L(M)$ .
- Call a language **Turing-recognizable** if some Turing machine recognizes it.
  - Also called : **recursively enumerable language**

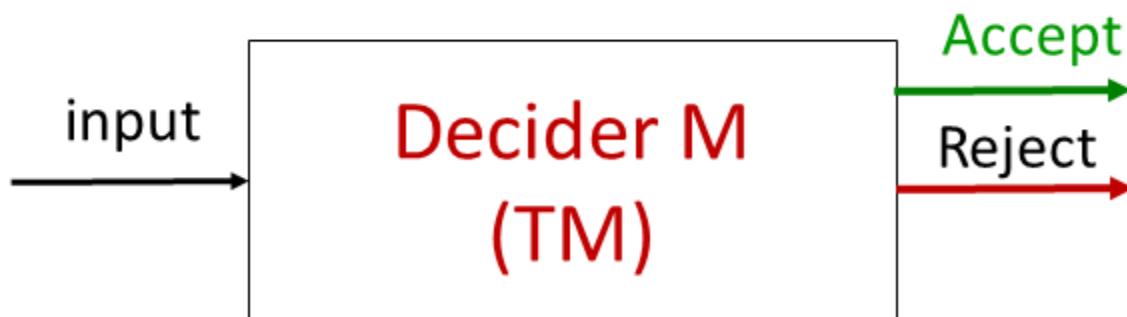


## Definition: Decider

- we **prefer** Turing machines that **halt on all inputs**;
  - such machines never loop.

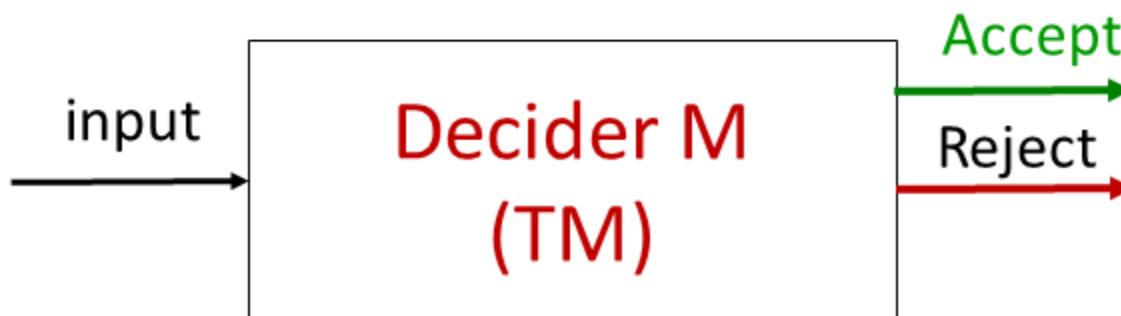
## Definition: Decider

- we prefer Turing machines that halt on all inputs;
  - such machines never loop.
- These machines are called **deciders** because they always make a decision to **accept or reject**.



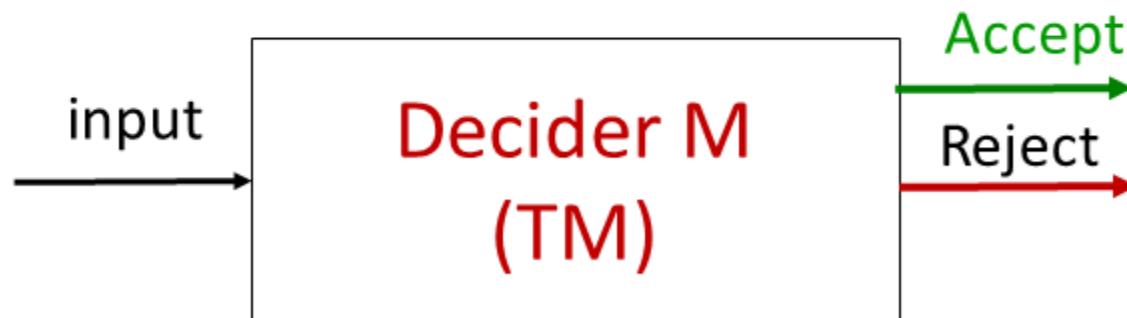
## Definition: Decider

- we prefer Turing machines that halt on all inputs;
  - such machines never loop.
- These machines are called **deciders** because they always make a decision to **accept or reject**.
- A **decider that recognizes** some language also is said
  - to decide that language.



## Definition: Turing-decidable

- Call a language **Turing-decidable** or simply **decidable**
  - if **some** Turing machine decides it.
  - Also called : **recursive language**
- Every **decidable language** is **Turing-recognizable**.



## Example 4

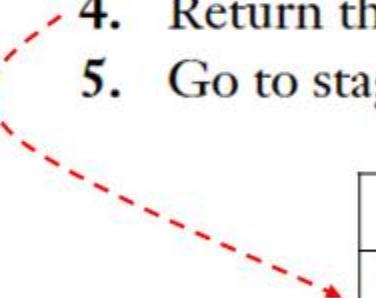
- The following Turing machine ( $TM$ )  $M2$  decides  $A = \{0^{2^n} \mid n \geq 0\}$ ,
  - the language consisting of all strings of 0s whose length is a power of 2.

## Example 4

- The following Turing machine ( $TM$ )  $M_2$  decides  $A = \{0^{2^n} \mid n \geq 0\}$ ,
  - the language consisting of all strings of 0s whose length is a power of 2.

$M_2$  = “On input string  $w$ :

1. Sweep left to right across the tape, crossing off every other 0.
2. If in stage 1 the tape contained a single 0, *accept*.
3. If in stage 1 the tape contained more than a single 0 and the number of 0s was odd, *reject*.
4. Return the head to the left-hand end of the tape.
5. Go to stage 1.”



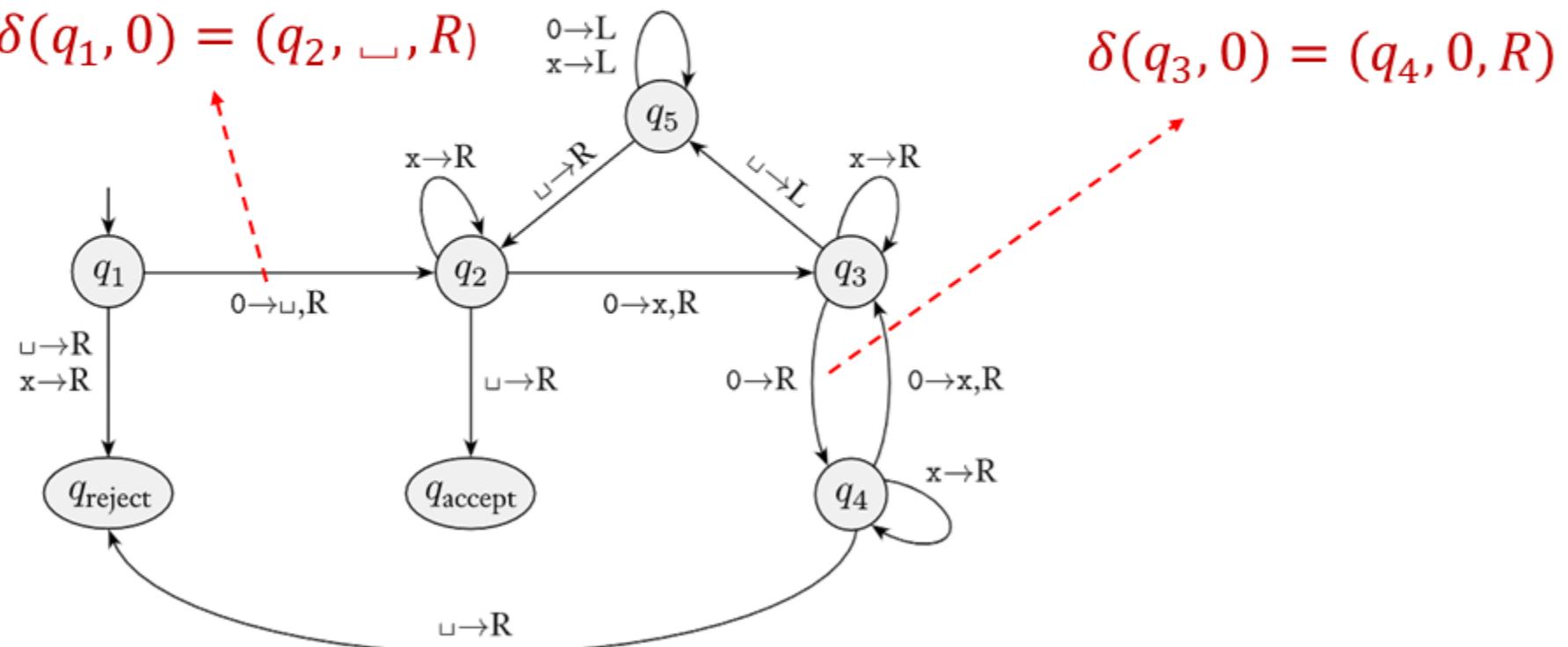
0	0	0	0	0	0	0	0	◻	◻	◻	...
◻	X	0	X	0	X	0	X	◻	◻	◻	...

(cuts the number of 0s in half.)

### Chapter 3

#### Example 4

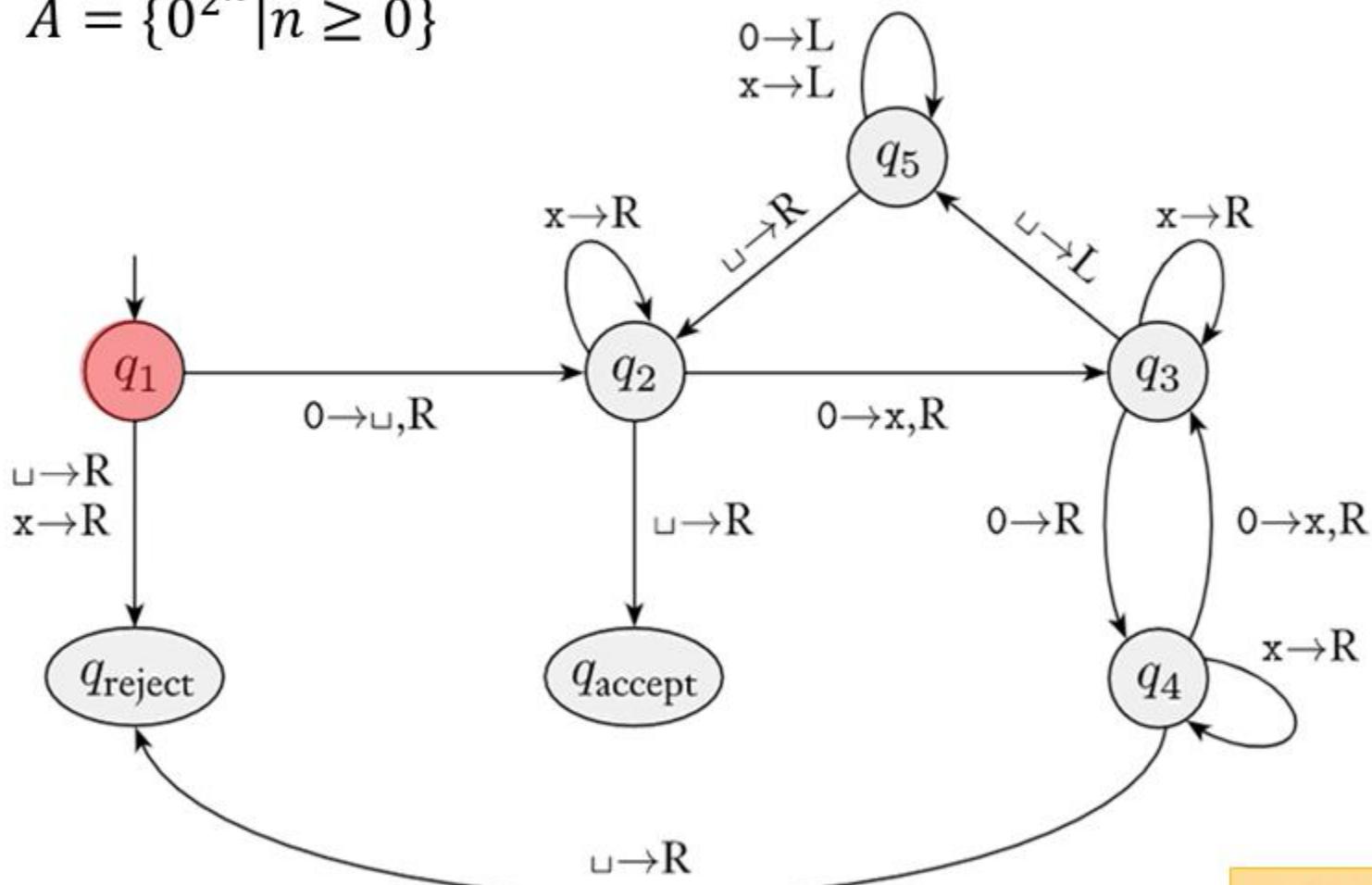
- $A = \{0^{2^n} | n \geq 0\}$ ,



## Chapter 3

### Example 4

- $A = \{0^{2n} | n \geq 0\}$



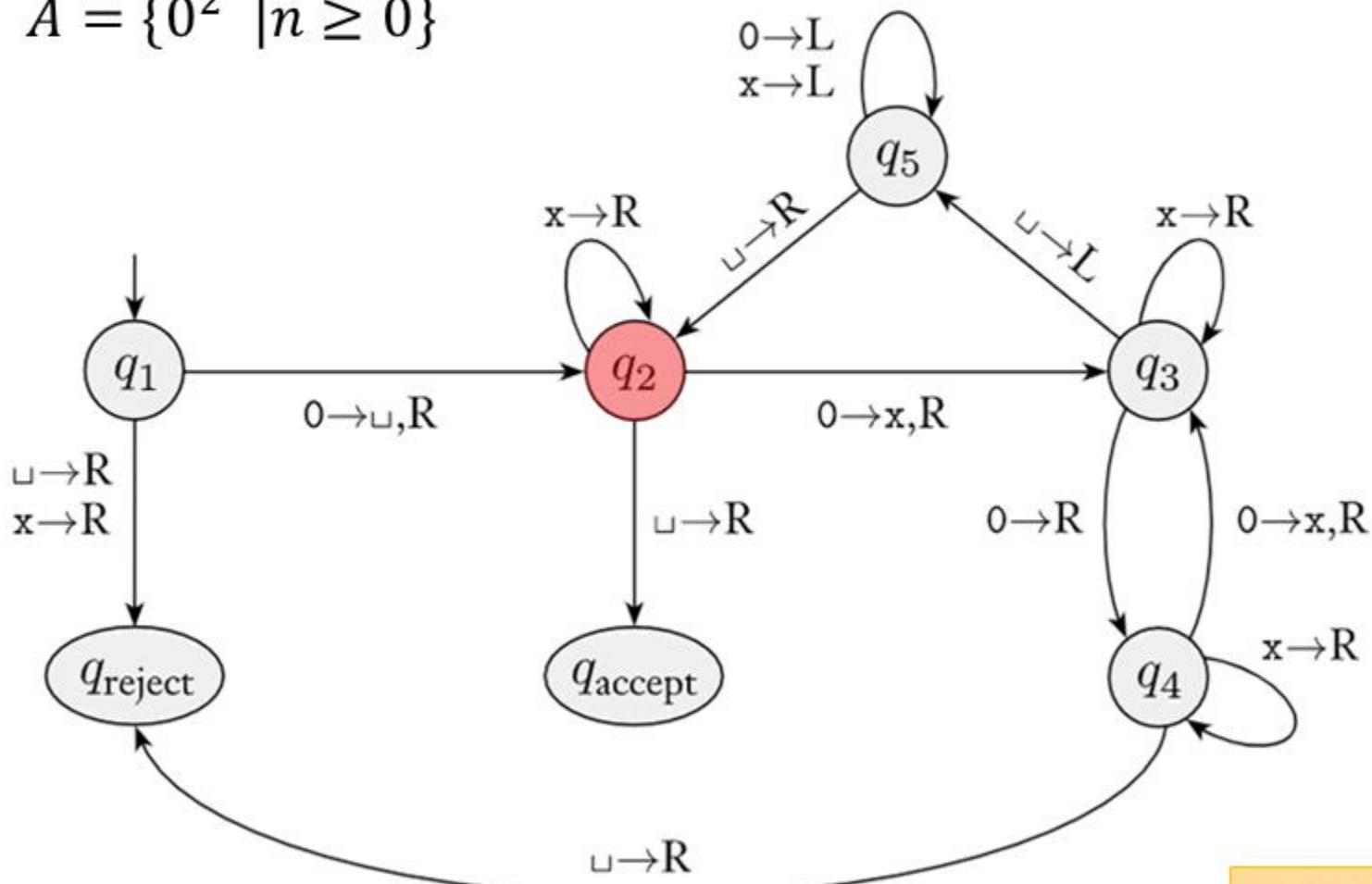
a sample run of  
this machine on  
input 0000

q10000
◻q2000
◻xq300
◻x0q40
◻x0xq3◻
◻x0q5x◻
◻xq50x◻
◻q5x0x◻
q5◻x0x◻
◻q2x0x◻
◻xq20x◻
◻xxq3x◻
◻xxxq3◻
◻xxq5x◻
◻xq5xx◻
◻q5xxx◻
◻q2xxx◻
◻xq2xx◻
◻xxq2x◻
◻xxxq2◻
◻xxx◻qaccept

## Chapter 3

### Example 4

- $A = \{0^{2n} | n \geq 0\}$



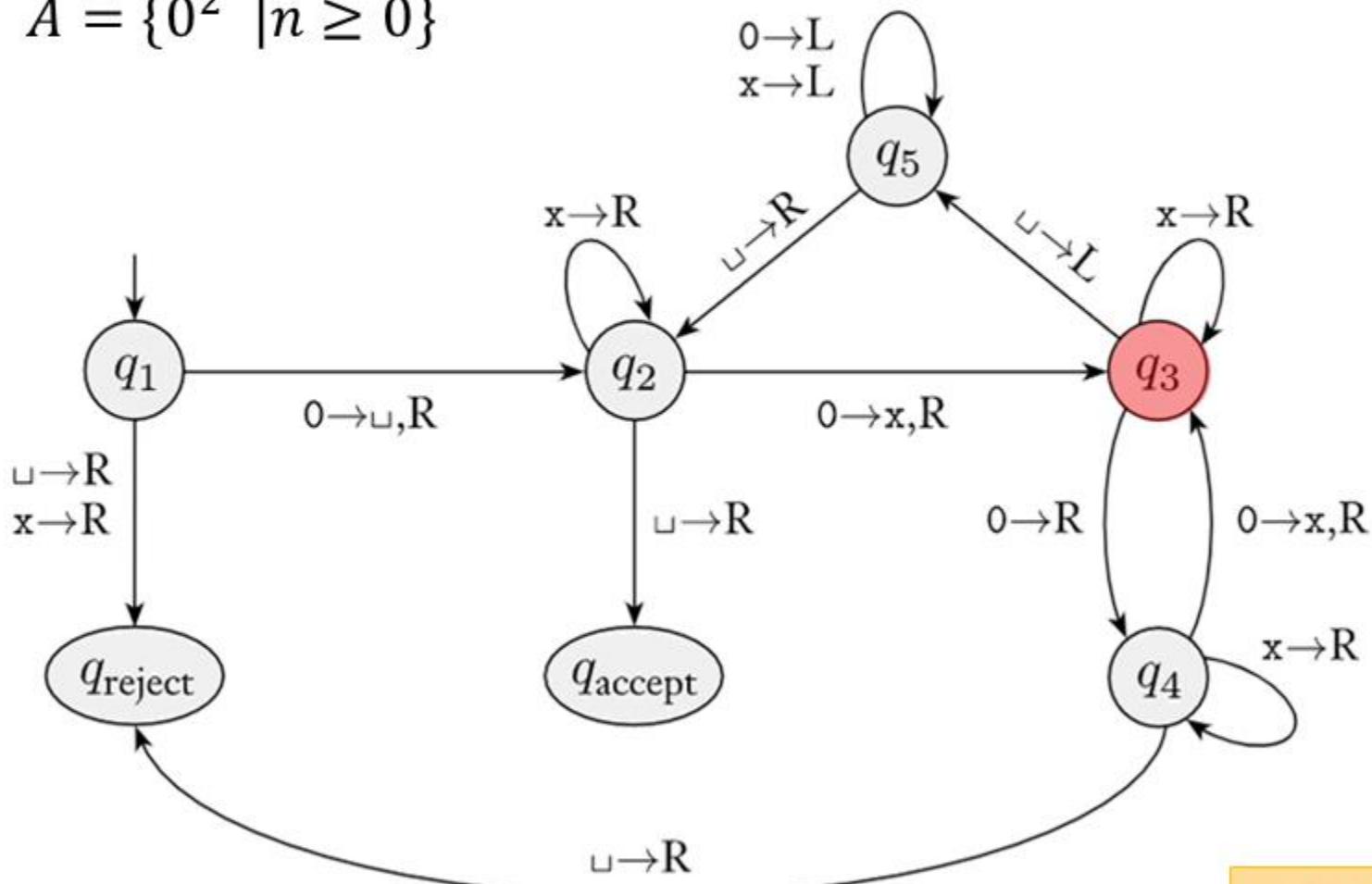
a sample run of  
this machine on  
input 0000

$q_1 0000$
$\sqcup q_2 000$
$\sqcup x q_3 00$
$\sqcup x 0 q_4 0$
$\sqcup x 0 x q_3 \sqcup$
$\sqcup x 0 q_5 x \sqcup$
$\sqcup x q_5 0 x \sqcup$
$\sqcup q_5 x 0 x \sqcup$
$q_5 \sqcup x 0 x \sqcup$
$\sqcup q_2 x 0 x \sqcup$
$\sqcup x q_2 0 x \sqcup$
$\sqcup x x q_3 x \sqcup$
$\sqcup x x x q_3 \sqcup$
$\sqcup x x q_5 x \sqcup$
$\sqcup x q_5 x x \sqcup$
$\sqcup q_5 x x x \sqcup$
$q_5 \sqcup x x x \sqcup$
$\sqcup q_2 x x x \sqcup$
$\sqcup x q_2 x x \sqcup$
$\sqcup x x q_2 x \sqcup$
$\sqcup x x x q_2 \sqcup$
$\sqcup x x x \sqcup q_{accept}$

## Chapter 3

### Example 4

- $A = \{0^{2n} | n \geq 0\}$



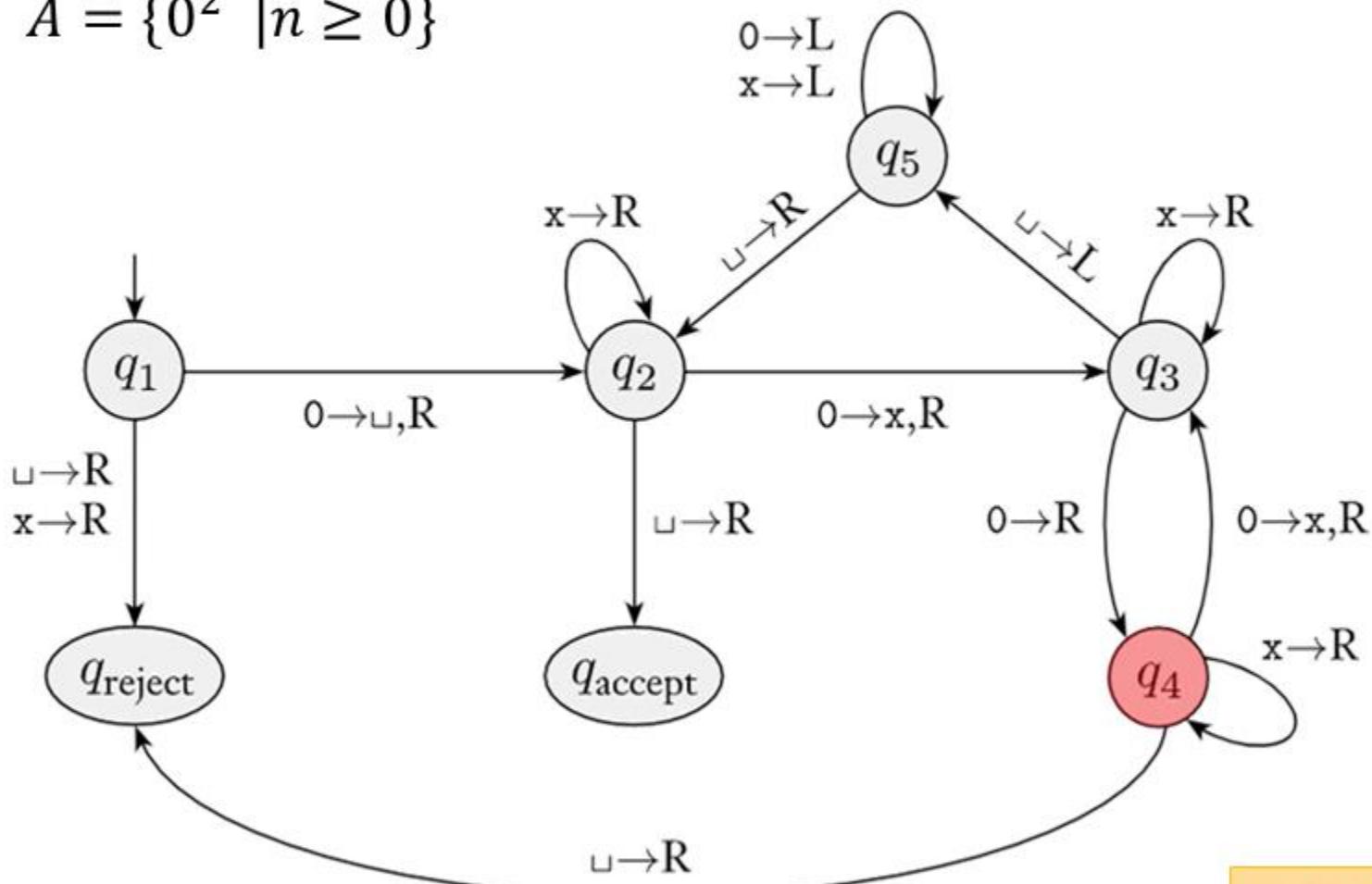
a sample run of  
this machine on  
input 0000

$q_1 0000$
$\sqcup q_2 000$
$\sqcup x q_3 00$
$\sqcup x 0 q_4 0$
$\sqcup x 0 x q_3 \sqcup$
$\sqcup x 0 q_5 x \sqcup$
$\sqcup x q_5 0 x \sqcup$
$\sqcup q_5 x 0 x \sqcup$
$q_5 \sqcup x 0 x \sqcup$
$\sqcup q_2 x 0 x \sqcup$
$\sqcup x q_2 0 x \sqcup$
$\sqcup x x q_3 x \sqcup$
$\sqcup x x x q_3 \sqcup$
$\sqcup x x q_5 x \sqcup$
$\sqcup x q_5 x x \sqcup$
$\sqcup q_5 x x x \sqcup$
$q_5 \sqcup x x x \sqcup$
$\sqcup q_2 x x x \sqcup$
$\sqcup x q_2 x x \sqcup$
$\sqcup x x q_2 x \sqcup$
$\sqcup x x x q_2 \sqcup$
$\sqcup x x x \sqcup q_{accept}$

## Chapter 3

### Example 4

- $A = \{0^{2n} | n \geq 0\}$



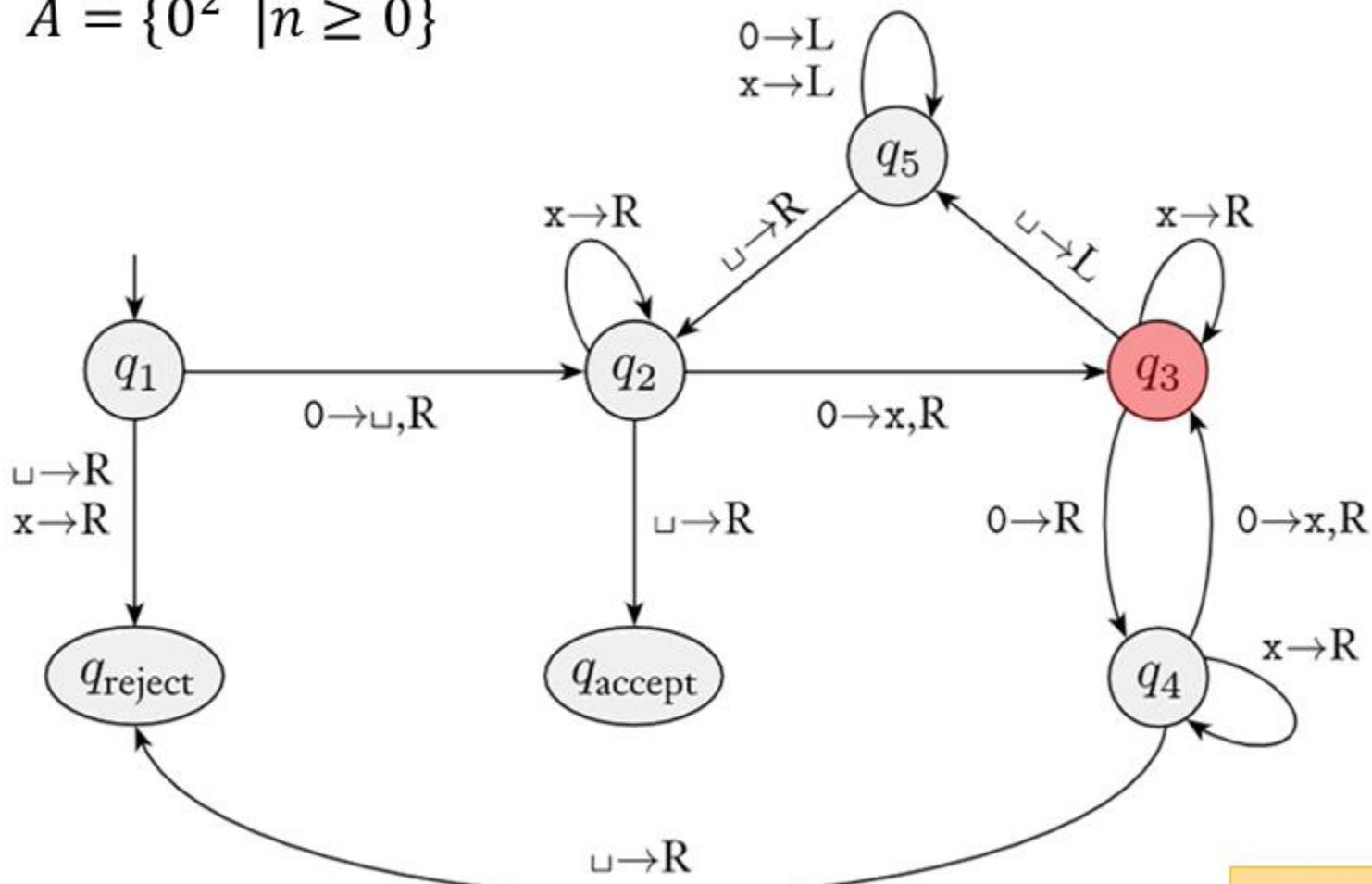
a sample run of  
this machine on  
input 0000

$q_1 0000$
$\sqcup q_2 000$
$\sqcup x q_3 00$
$\sqcup x 0 q_4 0$
$\sqcup x 0 x q_3 \sqcup$
$\sqcup x 0 q_5 x \sqcup$
$\sqcup x q_5 0 x \sqcup$
$\sqcup q_5 x 0 x \sqcup$
$q_5 \sqcup x 0 x \sqcup$
$\sqcup q_2 x 0 x \sqcup$
$\sqcup x q_2 0 x \sqcup$
$\sqcup x x q_3 x \sqcup$
$\sqcup x x x q_3 \sqcup$
$\sqcup x x q_5 x \sqcup$
$\sqcup x q_5 x x \sqcup$
$q_5 \sqcup x x x \sqcup$
$\sqcup q_2 x x x \sqcup$
$\sqcup x q_2 x x \sqcup$
$\sqcup x x q_2 x \sqcup$
$\sqcup x x x q_2 \sqcup$
$\sqcup x x x \sqcup q_{accept}$

## Chapter 3

### Example 4

- $A = \{0^{2n} | n \geq 0\}$



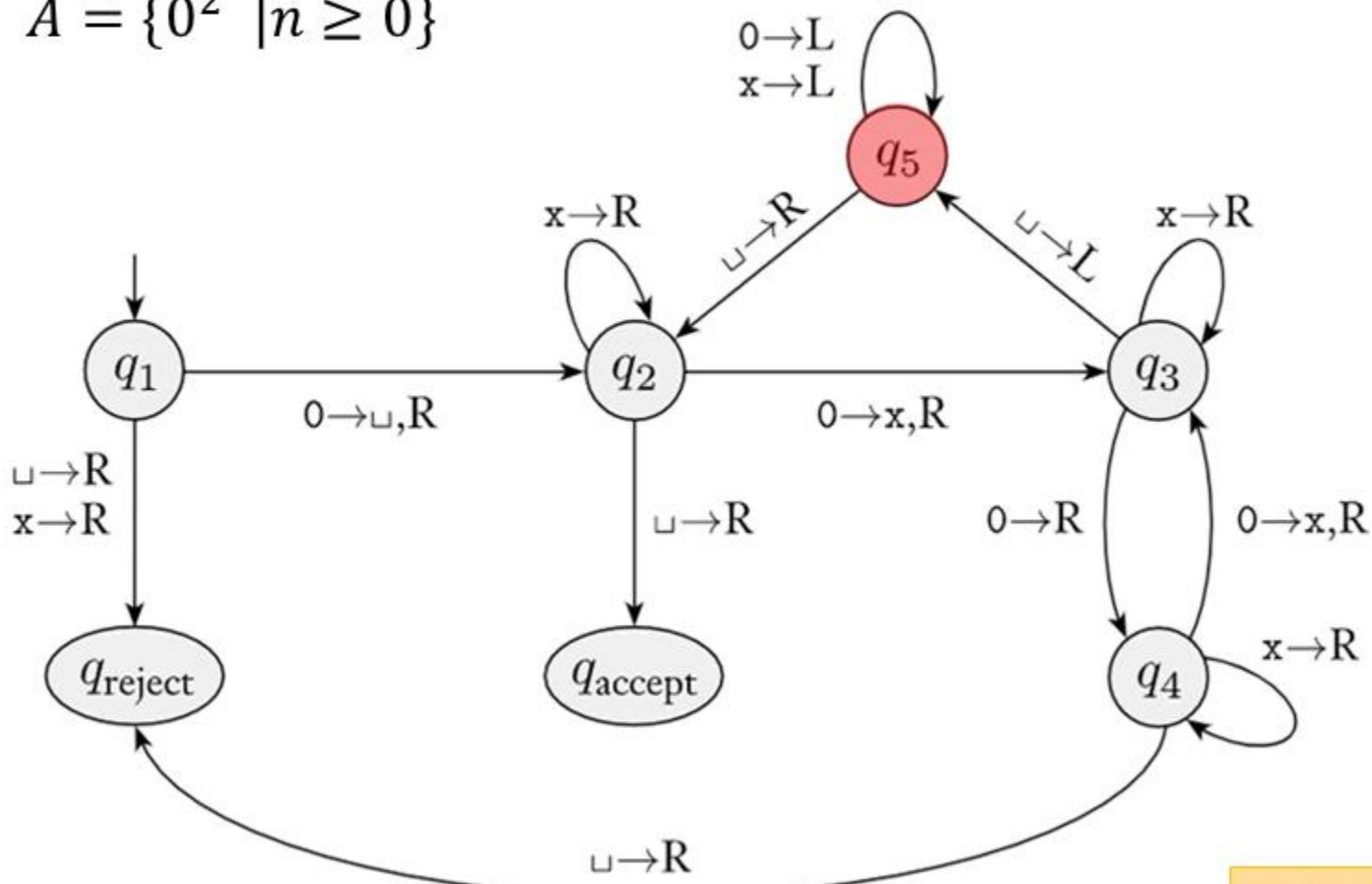
a sample run of  
this machine on  
input 0000

$q_1 0000$
$\sqcup q_2 000$
$\sqcup x q_3 00$
$\sqcup x 0 q_4 0$
$\sqcup x 0 x q_3 \sqcup$
$\sqcup x 0 q_5 x \sqcup$
$\sqcup x q_5 0 x \sqcup$
$\sqcup q_5 x 0 x \sqcup$
$q_5 \sqcup x 0 x \sqcup$
$\sqcup q_2 x 0 x \sqcup$
$\sqcup x q_2 0 x \sqcup$
$\sqcup x x q_3 x \sqcup$
$\sqcup x x x q_3 \sqcup$
$\sqcup x x q_5 x \sqcup$
$\sqcup x q_5 x x \sqcup$
$q_5 \sqcup x x x \sqcup$
$\sqcup q_2 x x x \sqcup$
$\sqcup x q_2 x x \sqcup$
$\sqcup x x q_2 x \sqcup$
$\sqcup x x x q_2 \sqcup$
$\sqcup x x x \sqcup q_{accept}$

## Chapter 3

### Example 4

- $A = \{0^{2n} | n \geq 0\}$



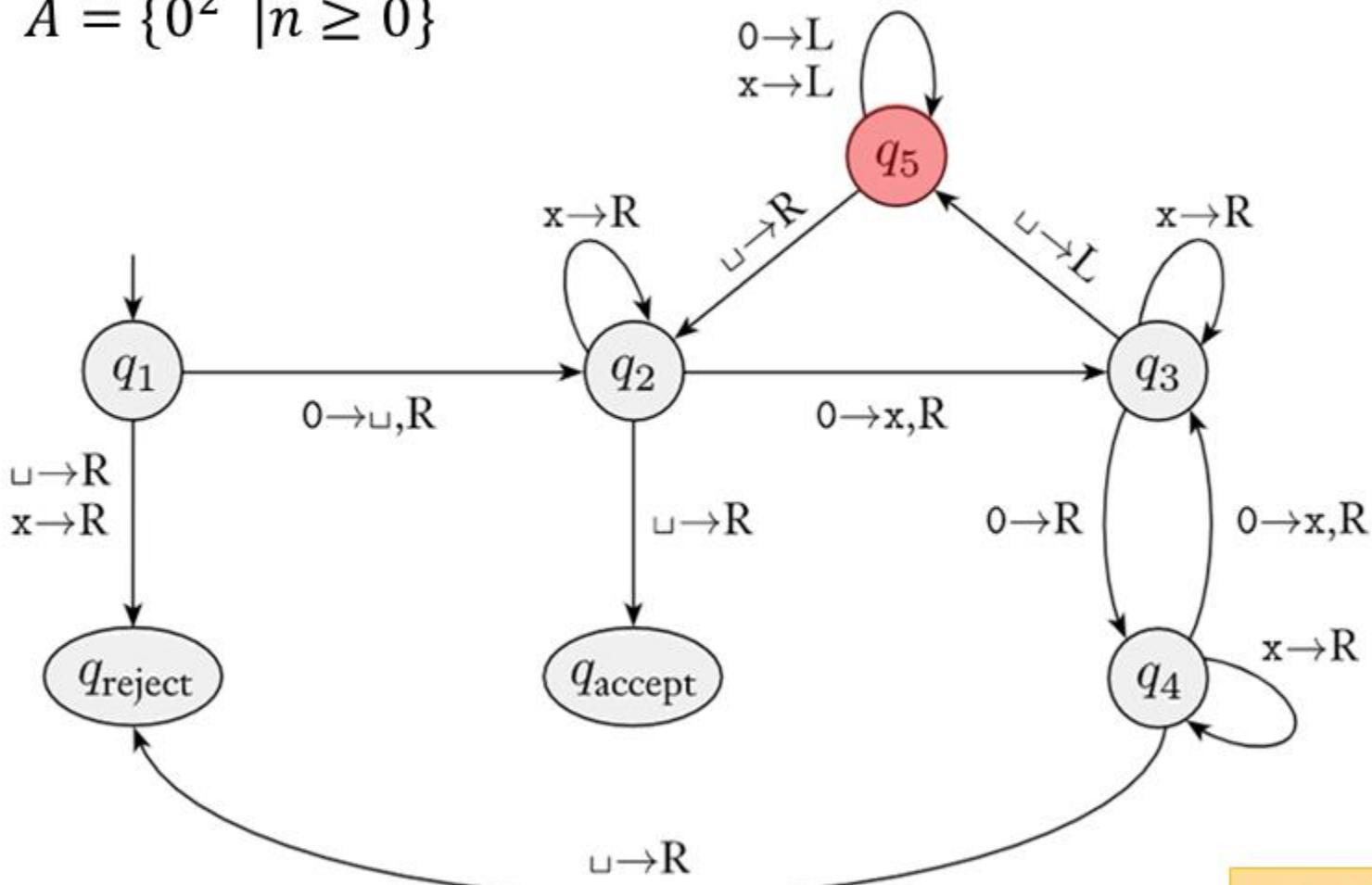
a sample run of  
this machine on  
input 0000

$q_1 0000$
$\sqcup q_2 000$
$\sqcup x q_3 00$
$\sqcup x 0 q_4 0$
$\sqcup x 0 x q_3 \sqcup$
$\sqcup x 0 q_5 x \sqcup$
$\sqcup x q_5 0 x \sqcup$
$\sqcup q_5 x 0 x \sqcup$
$q_5 \sqcup x 0 x \sqcup$
$\sqcup q_2 x 0 x \sqcup$
$\sqcup x q_2 0 x \sqcup$
$\sqcup x x q_3 x \sqcup$
$\sqcup x x x q_3 \sqcup$
$\sqcup x x q_5 x \sqcup$
$\sqcup x q_5 x x \sqcup$
$q_5 \sqcup x x x \sqcup$
$\sqcup q_2 x x x \sqcup$
$\sqcup x q_2 x x \sqcup$
$\sqcup x x q_2 x \sqcup$
$\sqcup x x x q_2 \sqcup$
$\sqcup x x x \sqcup q_{accept}$

## Chapter 3

### Example 4

- $A = \{0^{2n} | n \geq 0\}$



$\square \rightarrow R$

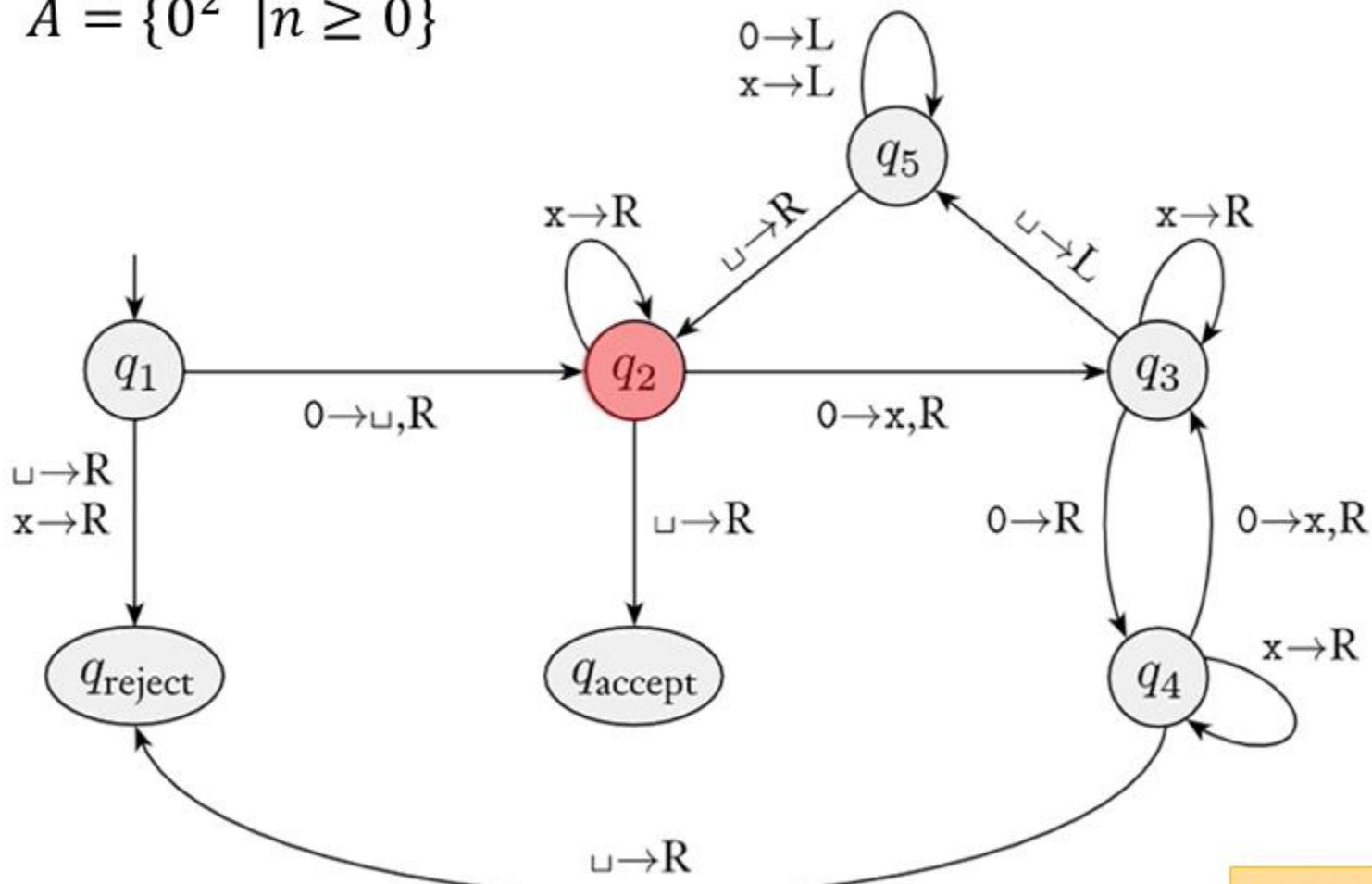
a sample run of  
this machine on  
input 0000

$q_1 0000$
$\sqcup q_2 000$
$\sqcup x q_3 00$
$\sqcup x 0 q_4 0$
$\sqcup x 0 x q_3 \sqcup$
$\sqcup x 0 q_5 x \sqcup$
$\sqcup x q_5 0 x \sqcup$
$\sqcup q_5 x 0 x \sqcup$
$q_5 \sqcup x 0 x \sqcup$
$\sqcup q_2 x 0 x \sqcup$
$\sqcup x q_2 0 x \sqcup$
$\sqcup x x q_3 x \sqcup$
$\sqcup x x x q_3 \sqcup$
$\sqcup x x q_5 x \sqcup$
$\sqcup x q_5 x x \sqcup$
$\sqcup q_5 x x x \sqcup$
$q_5 \sqcup x x x \sqcup$
$\sqcup q_2 x x x \sqcup$
$\sqcup x q_2 x x \sqcup$
$\sqcup x x q_2 x \sqcup$
$\sqcup x x x q_2 \sqcup$
$\sqcup x x x \sqcup q_{\text{accept}}$

## Chapter 3

### Example 4

- $A = \{0^{2n} | n \geq 0\}$



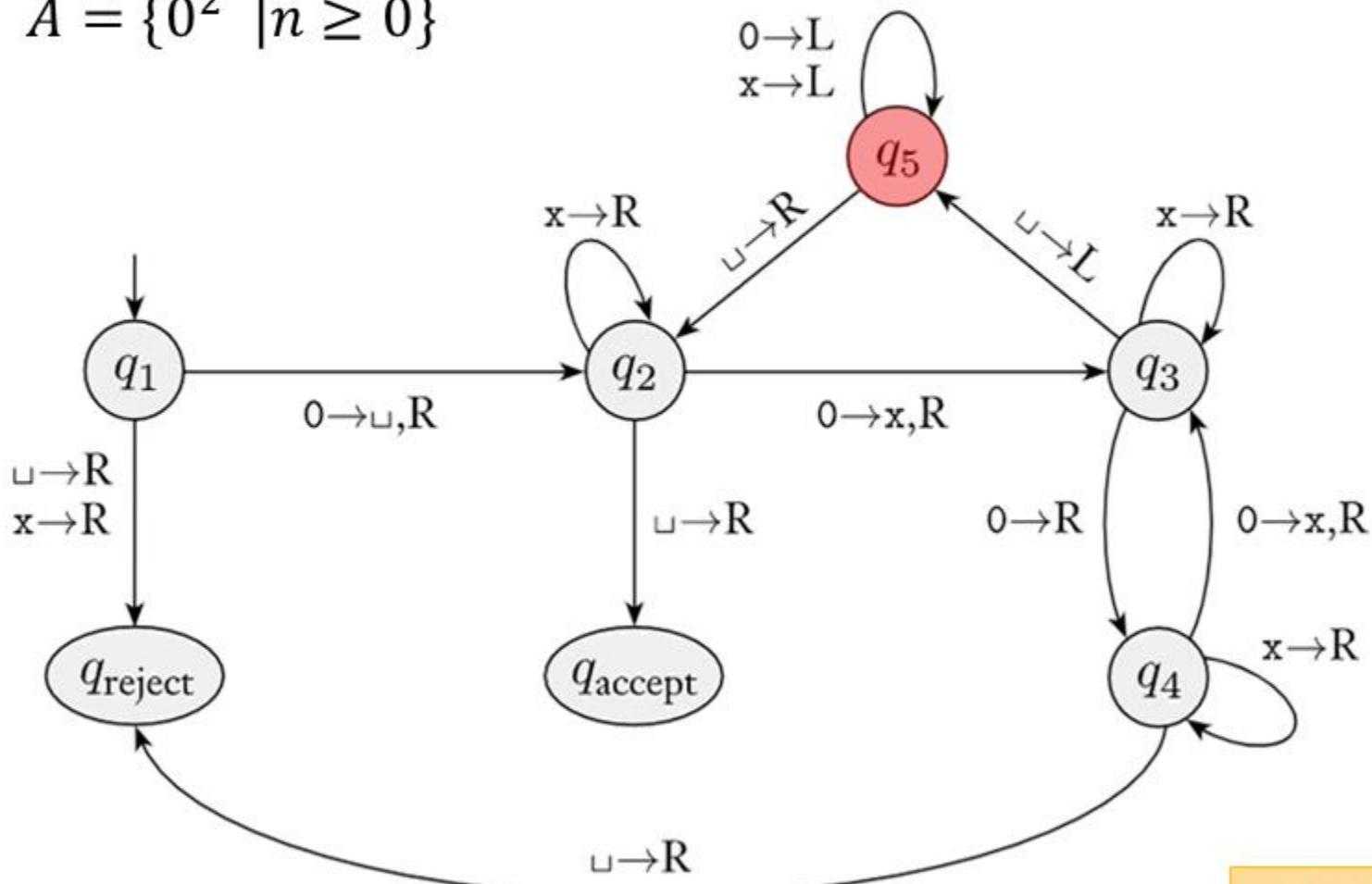
a sample run of  
this machine on  
input 0000

q<sub>1</sub>0000  
◻q<sub>2</sub>000  
◻xq<sub>3</sub>00  
◻x0q<sub>4</sub>0  
◻x0xq<sub>3</sub>◻  
◻x0q<sub>5</sub>x◻  
◻xq<sub>5</sub>0x◻  
◻q<sub>5</sub>x0x◻  
q<sub>5</sub>◻x0x◻  
◻q<sub>2</sub>x0x◻  
◻xq<sub>2</sub>0x◻  
◻xxq<sub>3</sub>x◻  
◻xxxq<sub>3</sub>◻  
◻xxq<sub>5</sub>x◻  
◻xq<sub>5</sub>xx◻  
◻q<sub>5</sub>xxx◻  
q<sub>5</sub>◻xxx◻  
◻q<sub>2</sub>xxx◻  
◻xq<sub>2</sub>xx◻  
◻xxxq<sub>2</sub>◻  
◻xxx◻q<sub>accept</sub>

## Chapter 3

### Example 4

- $A = \{0^{2n} | n \geq 0\}$

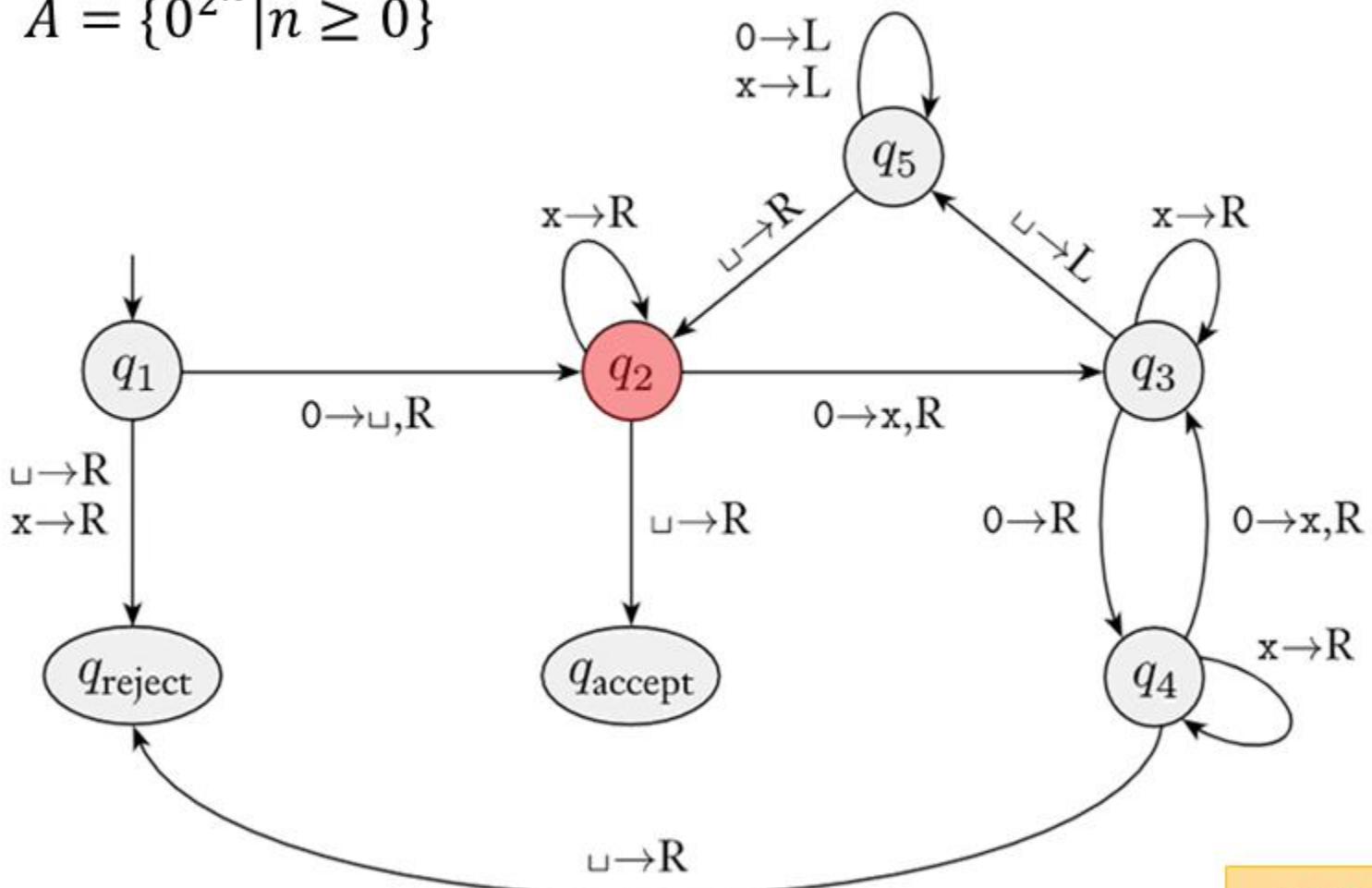


a sample run of  
this machine on  
input 0000

$q_1 0000$   
 $\sqcup q_2 000$   
 $\sqcup x q_3 00$   
 $\sqcup x 0 q_4 0$   
 $\sqcup x 0 x q_3 \sqcup$   
 $\sqcup x 0 q_5 x \sqcup$   
 $\sqcup x q_5 0 x \sqcup$   
 $\sqcup q_5 x 0 x \sqcup$   
 $q_5 \sqcup x 0 x \sqcup$   
 $\sqcup q_2 x 0 x \sqcup$   
 $\sqcup x q_2 0 x \sqcup$   
 $\sqcup x x q_3 x \sqcup$   
 $\sqcup x x x q_3 \sqcup$   
 $\sqcup x x q_5 x \sqcup$   
 $\sqcup x q_5 x x \sqcup$   
 $\sqcup q_5 x x x \sqcup$   
 $q_5 \sqcup x x x \sqcup$   
 $\sqcup q_2 x x x \sqcup$   
 $\sqcup x q_2 x x \sqcup$   
 $\sqcup x x q_2 x \sqcup$   
 $\sqcup x x x q_2 \sqcup$   
 $\sqcup x x x \sqcup q_{accept}$

## Example 4

- $A = \{0^{2^n} \mid n \geq 0\}$



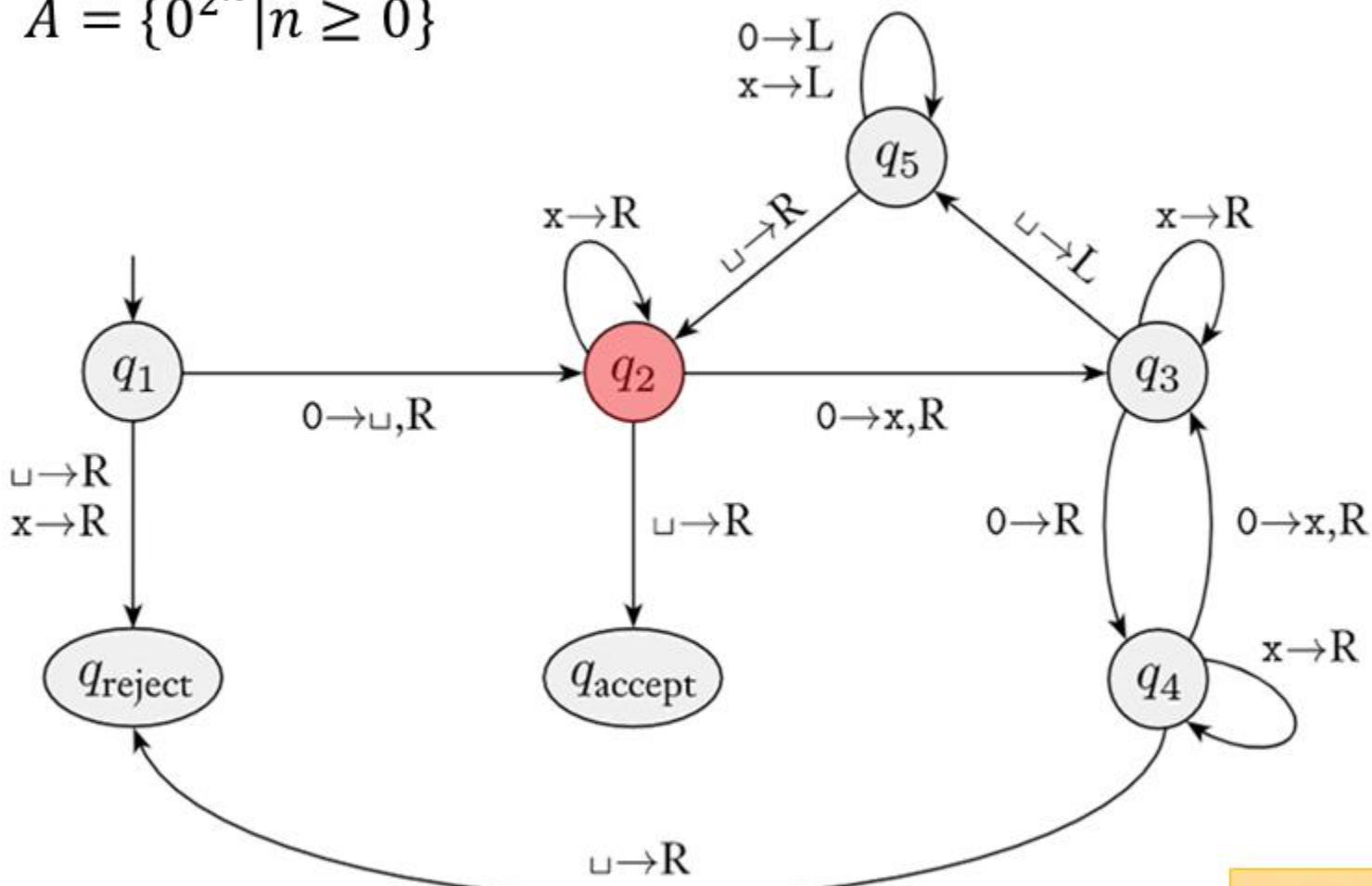
a sample run of  
this machine on  
input 0000

$q_1$ 0000  
 $\sqcup q_2$ 000  
 $\sqcup xq_3$ 00  
 $\sqcup x0q_4$ 0  
 $\sqcup x0xq_3$  $\sqcup$   
 $\sqcup x0q_5$ x $\sqcup$   
 $\sqcup xq_5$ 0x $\sqcup$   
 $\sqcup q_5$ x0x $\sqcup$   
 $q_5$  $\sqcup x0x$  $\sqcup$   
 $\sqcup q_2$ x0x $\sqcup$   
 $\sqcup xq_2$ 0x $\sqcup$   
 $\sqcup xxq_3$ x $\sqcup$   
 $\sqcup xxxxq_3$  $\sqcup$   
 $\sqcup xxq_5$ x $\sqcup$   
 $\sqcup xq_5$ xx $\sqcup$   
 $\sqcup q_5$ xxx $\sqcup$   
 $q_5$  $\sqcup xxx$  $\sqcup$   
 $\sqcup q_2$ xxx $\sqcup$   
 $\sqcup xq_2$ xx $\sqcup$   
 $\sqcup xxq_2$ x $\sqcup$   
 $\sqcup xxxxq_2$  $\sqcup$   
 $\sqcup xxxx$  $\sqcup q_{acc}$

## Chapter 3

### Example 4

- $A = \{0^{2n} | n \geq 0\}$



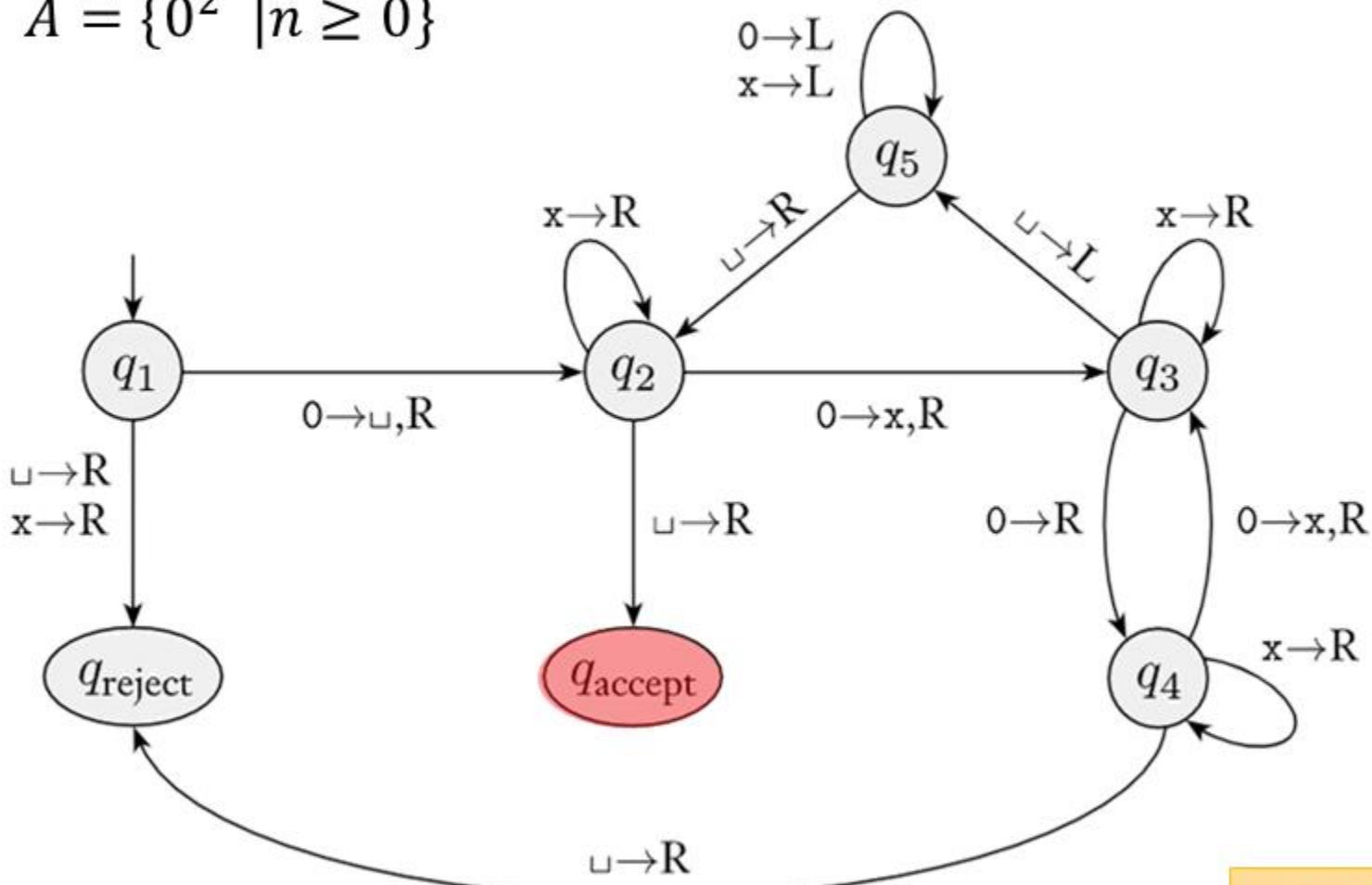
a sample run of  
this machine on  
input 0000

q<sub>1</sub>0000  
  q<sub>2</sub>000  
  xq<sub>3</sub>00  
  x0q<sub>4</sub>0  
  x0xq<sub>3</sub>  
  x0q<sub>5</sub>x  
  xq<sub>5</sub>0x  
  q<sub>5</sub>x0x  
  q<sub>2</sub>x0x  
  xq<sub>2</sub>0x  
  xxq<sub>3</sub>x  
  xxxq<sub>3</sub>  
  xxq<sub>5</sub>x  
  xq<sub>5</sub>xx  
  q<sub>5</sub>xxx  
  q<sub>2</sub>xxx  
  xq<sub>2</sub>xx  
  xxq<sub>2</sub>x  
  xxxxq<sub>2</sub>  
  xxxxq<sub>accept</sub>

## Chapter 3

### Example 4

- $A = \{0^{2n} | n \geq 0\}$



a sample run of  
this machine on  
input 0000

$q_1 0000$   
 $\sqcup q_2 000$   
 $\sqcup x q_3 00$   
 $\sqcup x 0 q_4 0$   
 $\sqcup x 0 x q_3 \sqcup$   
 $\sqcup x 0 q_5 x \sqcup$   
 $\sqcup x q_5 0 x \sqcup$   
 $\sqcup q_5 x 0 x \sqcup$   
 $q_5 \sqcup x 0 x \sqcup$   
 $\sqcup q_2 x 0 x \sqcup$   
 $\sqcup x q_2 0 x \sqcup$   
 $\sqcup x x q_3 x \sqcup$   
 $\sqcup x x x q_3 \sqcup$   
 $\sqcup x x q_5 x \sqcup$   
 $\sqcup x q_5 x x \sqcup$   
 $\sqcup q_5 x x x \sqcup$   
 $\sqcup q_2 x x x \sqcup$   
 $\sqcup x q_2 x x \sqcup$   
 $\sqcup x x q_2 x \sqcup$   
 $\sqcup x x x q_2 \sqcup$   
 $\sqcup x x x \sqcup q_{accept}$

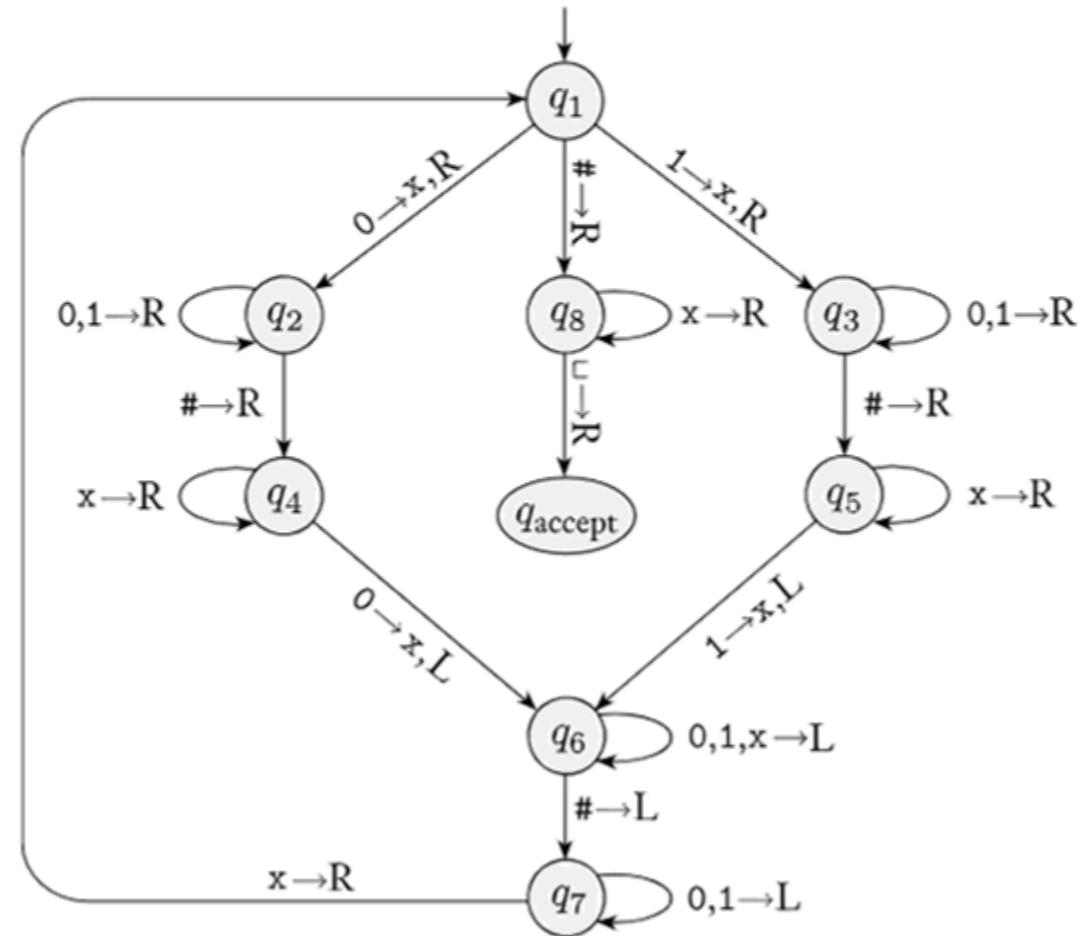
## Example 5

Formal definition of Turing machine ( $TM$ )  $M_1$  that decides  
$$B = \{w\#w \mid w \in \{0,1\}^*\}.$$

### Chapter 3

### Example 5

Formal definition of Turing machine ( $TM$ )  $M_1$  that decides  
 $B = \{w\#w \mid w \in \{0,1\}^*\}$ .

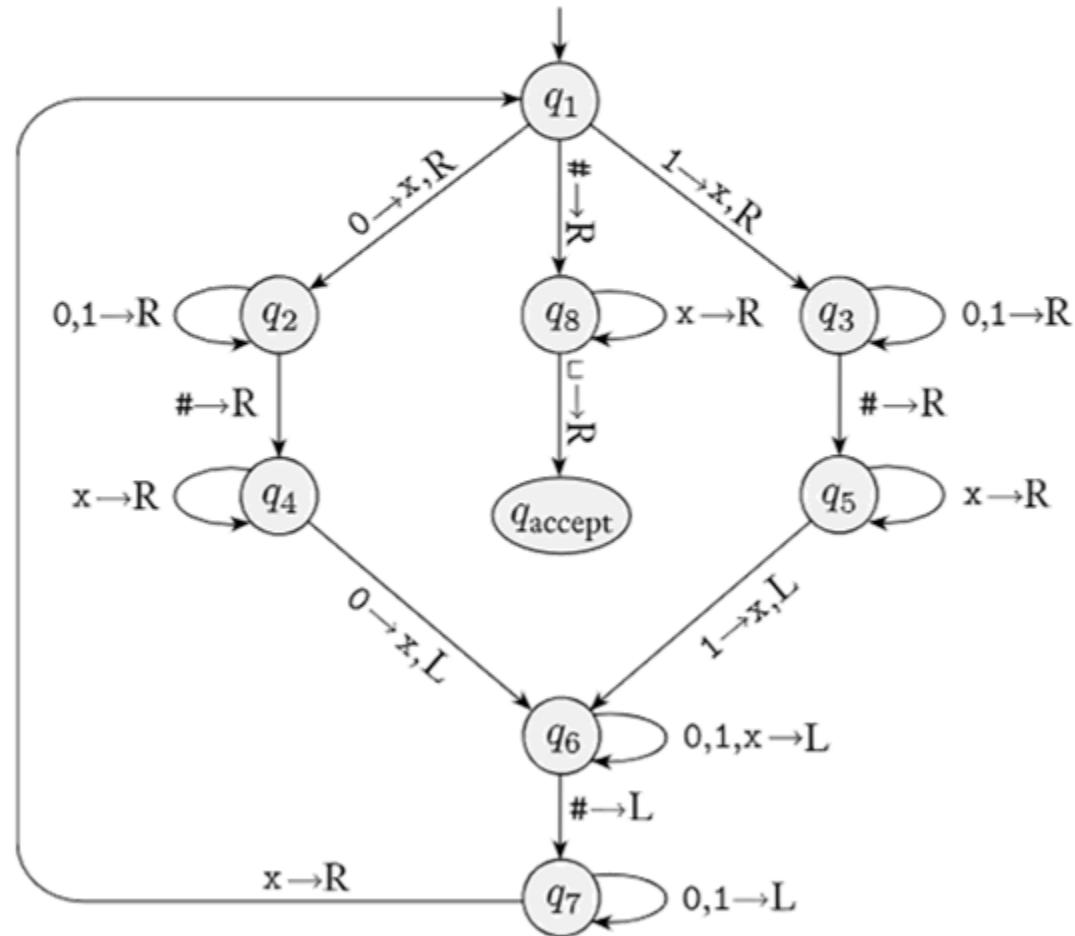


### Chapter 3

#### Example 5

Show the formal definition of Turing machine ( $TM$ )  $M_1$  that decides  
 $B = \{w\#w \mid w \in \{0,1\}^*\}$ .

- $M_1 = (Q, \Sigma, \Gamma, \delta, q_1, q_{accept}, q_{reject})$
- $Q = \{q_1, \dots, q_8, q_{accept}, q_{reject}\}$
- $\Sigma = \{0,1, \#\}$ , and  $\Gamma = \{0,1, \#, x, \_\}$ .
- The **transition function** can be described with a state diagram



### Chapter 3

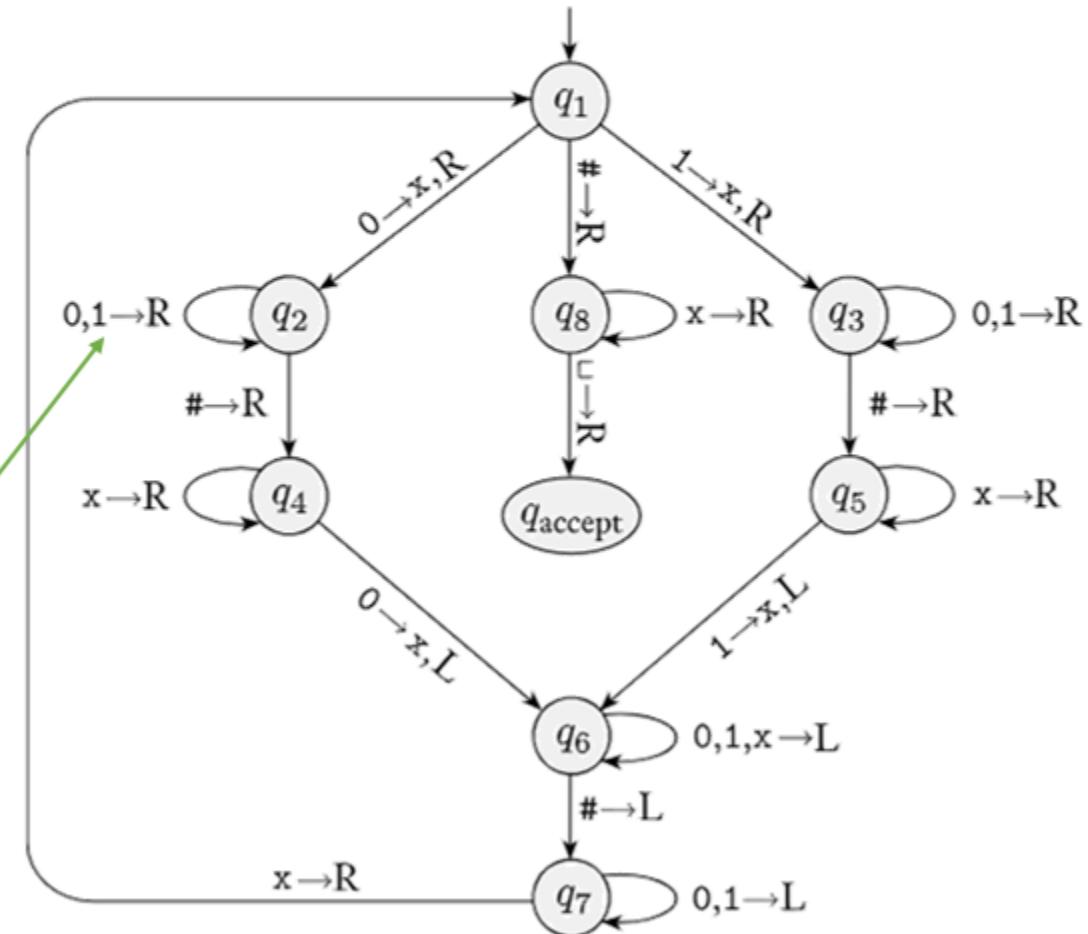
#### Example 5

Show the formal definition of Turing machine ( $TM$ )  $M_1$  that decides  
 $B = \{w\#w \mid w \in \{0,1\}^*\}$ .

- $M_1 = (Q, \Sigma, \Gamma, \delta, q_1, q_{accept}, q_{reject})$
- $Q = \{q_1, \dots, q_8, q_{accept}, q_{reject}\}$
- $\Sigma = \{0,1, \#\}$ , and  $\Gamma = \{0,1, \#, x, \_\}$ .
- The **transition function** can be described with a state diagram

The machine stays in  $q_2$  and moves to the right when it reads a 0 or a 1 in state  $q_2$ .

- To simplify the figure, we didn't show the reject state.



## Example 6

The TM  $M_3$  that decides the language  $C = \{a^i b^j c^k \mid i \times j = k \text{ and } i, j, k \geq 1\}$ .

- This machine is doing some elementary arithmetic.

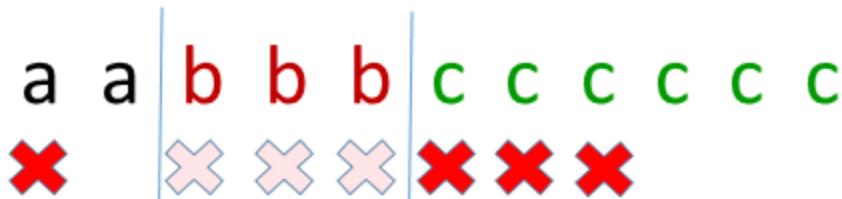
## Example 6

The TM  $M_3$  that decides the language  $C = \{a^i b^j c^k \mid i \times j = k \text{ and } i, j, k \geq 1\}$ .

- This machine is doing some elementary arithmetic.

$M_3$  = “On input string  $w$ :

1. Scan the input from left to right to determine whether it is a member of  $a^+b^+c^+$  and *reject* if it isn’t.
2. Return the head to the left-hand end of the tape.
3. Cross off an  $a$  and scan to the right until a  $b$  occurs. Shuttle between the  $b$ ’s and the  $c$ ’s, crossing off one of each until all  $b$ ’s are gone. If all  $c$ ’s have been crossed off and some  $b$ ’s remain, *reject*.
4. Restore the crossed off  $b$ ’s and repeat stage 3 if there is another  $a$  to cross off. If all  $a$ ’s have been crossed off, determine whether all  $c$ ’s also have been crossed off. If yes, *accept*; otherwise, *reject*.”



First iteration of Stage 3

## Example 8

- Here, a TM  $M_4$  is solving what is called **the element distinctness problem**.
- It is given a list of strings over  $\{0,1\}$  separated by  $\#$ s and its job is to accept if all the strings are different. The language is

$$E = \{\#x_1\#x_2\#\cdots\#x_l \mid \text{each } x_i \in \{0,1\}^* \text{ and } x_i \neq x_j \text{ for each } i \neq j\}$$

## Example 8

$$E = \{\#x_1\#x_2\#\cdots\#x_l | \text{each } x_i \in \{0,1\}^* \text{ and } x_i \neq x_j \text{ for each } i \neq j\}$$

$M_4$  = “On input  $w$ :

1. Place a mark on top of the leftmost tape symbol. If that symbol was a blank, *accept*. If that symbol was a #, continue with the next stage. Otherwise, *reject*.

Placing the 1<sup>st</sup> mark

#101#11#1#1011

#101#11#1#1011

#101#11#1#1011

#101#11#1#1011

#101#11#1#1011

#101#11#1#1011

#101#11#1#1011

## Example 8

$$E = \{\#x_1\#x_2\#\cdots\#x_l \mid \text{each } x_i \in \{0,1\}^* \text{ and } x_i \neq x_j \text{ for each } i \neq j\}$$

$M_4$  = “On input  $w$ :

1. Place a mark on top of the leftmost tape symbol. If that symbol was a blank, *accept*. If that symbol was a  $\#$ , continue with the next stage. Otherwise, *reject*.
2. Scan right to the next  $\#$  and place a second mark on top of it. If no  $\#$  is encountered before a blank symbol, only  $x_1$  was present, so *accept*.

Placing the 2<sup>nd</sup> mark

#101#11#1#1011

#101#11#1#1011

#101#11#1#1011

#101#11#1#1011

#101#11#1#1011

#101#11#1#1011

#101#11#1#1011

## Example 8

$$E = \{\#x_1\#x_2\#\cdots\#x_l | \text{each } x_i \in \{0,1\}^* \text{ and } x_i \neq x_j \text{ for each } i \neq j\}$$

$M_4$  = “On input  $w$ :

1. Place a mark on top of the leftmost tape symbol. If that symbol was a blank, *accept*. If that symbol was a #, continue with the next stage. Otherwise, *reject*.
2. Scan right to the next # and place a second mark on top of it. If no # is encountered before a blank symbol, only  $x_1$  was present, so *accept*.
3. By zig-zagging, compare the two strings to the right of the marked #s. If they are equal, *reject*.

## Comparing two strings

#101#11#1#1011

#101#11#1#1011

#101#11#1#1011

#101#11#1#1011

#101#11#1#1011

#101#11#1#1011

#101#11#1#1011

## Example 8

$$E = \{\#x_1\#x_2\#\cdots\#x_l | \text{each } x_i \in \{0,1\}^* \text{ and } x_i \neq x_j \text{ for each } i \neq j\}$$

$M_4$  = “On input  $w$ :

1. Place a mark on top of the leftmost tape symbol. If that symbol was a blank, *accept*. If that symbol was a  $\#$ , continue with the next stage. Otherwise, *reject*.
2. Scan right to the next  $\#$  and place a second mark on top of it. If no  $\#$  is encountered before a blank symbol, only  $x_1$  was present, so *accept*.
3. By zig-zagging, compare the two strings to the right of the marked  $\#$ s. If they are equal, *reject*.
4. Move the rightmost of the two marks to the next  $\#$  symbol to the right. If no  $\#$  symbol is encountered before a blank symbol, move the leftmost mark to the next  $\#$  to its right and the rightmost mark to the  $\#$  after that. This time, if no  $\#$  is available for the rightmost mark, all the strings have been compared, so *accept*.

**Updating the marked  $\#$ s**

#101#11#1#1011  
#101#11#1#1011  
#101#11#1#1011  
#101#11#1#1011  
#101#11#1#1011  
#101#11#1#1011  
#101#11#1#1011

## Example 8

$$E = \{\#x_1\#x_2\#\cdots\#x_l | \text{each } x_i \in \{0,1\}^* \text{ and } x_i \neq x_j \text{ for each } i \neq j\}$$

$M_4$  = “On input  $w$ :

1. Place a mark on top of the leftmost tape symbol. If that symbol was a blank, *accept*. If that symbol was a #, continue with the next stage. Otherwise, *reject*.
2. Scan right to the next # and place a second mark on top of it. If no # is encountered before a blank symbol, only  $x_1$  was present, so *accept*.
3. By zig-zagging, compare the two strings to the right of the marked #s. If they are equal, *reject*.
4. Move the rightmost of the two marks to the next # symbol to the right. If no # symbol is encountered before a blank symbol, move the leftmost mark to the next # to its right and the rightmost mark to the # after that. This time, if no # is available for the rightmost mark, all the strings have been compared, so *accept*.
5. Go to stage 3.”

#101#11#1#1011

#101#11#1#1011

#101#11#1#1011

#101#11#1#1011

#101#11#1#1011

#101#11#1#1011

#101#11#1#1011

## Example 8

$$E = \{\#x_1\#x_2\#\cdots\#x_l | \text{each } x_i \in \{0,1\}^* \text{ and } x_i \neq x_j \text{ for each } i \neq j\}$$

$M_4$  = “On input  $w$ :

1. Place a mark on top of the leftmost tape symbol. If that symbol was a blank, *accept*. If that symbol was a #, continue with the next stage. Otherwise, *reject*.
2. Scan right to the next # and place a second mark on top of it. If no # is encountered before a blank symbol, only  $x_1$  was present, so *accept*.
3. By zig-zagging, compare the two strings to the right of the marked #s. If they are equal, *reject*.
4. Move the rightmost of the two marks to the next # symbol to the right. If no # symbol is encountered before a blank symbol, move the leftmost mark to the next # to its right and the rightmost mark to the # after that. This time, if no # is available for the rightmost mark, all the strings have been compared, so *accept*.
5. Go to stage 3.”

```
#101#11#1#1011
#101#11#1#1011
#101#11#1#1011
#101#11#1#1011
#101#11#1#1011
#101#11#1#1011
```

## Example 8

$$E = \{\#x_1\#x_2\#\cdots\#x_l | \text{each } x_i \in \{0,1\}^* \text{ and } x_i \neq x_j \text{ for each } i \neq j\}$$

$M_4$  = “On input  $w$ :

1. Place a mark on top of the leftmost tape symbol. If that symbol was a blank, *accept*. If that symbol was a #, continue with the next stage. Otherwise, *reject*.
  2. Scan right to the next # and place a second mark on top of it. If no # is encountered before a blank symbol, only  $x_1$  was present, so *accept*.
  3. By zig-zagging, compare the two strings to the right of the marked #s. If they are equal, *reject*.
  4. Move the rightmost of the two marks to the next # symbol to the right. If no # symbol is encountered before a blank symbol, move the leftmost mark to the next # to its right and the rightmost mark to the # after that. This time, if no # is available for the rightmost mark, all the strings have been compared, so *accept*.
  5. Go to stage 3.”

#101#11#1#1011

#101#11#1#1011

#101#11#1#1011

#101 #11#1 #1011

#101 #11#1 #1011

#101 #11#1 #1011

#101#11#1#1011

## Example 8

$$E = \{\#x_1\#x_2\#\cdots\#x_l | \text{each } x_i \in \{0,1\}^* \text{ and } x_i \neq x_j \text{ for each } i \neq j\}$$

$M_4$  = “On input  $w$ :

1. Place a mark on top of the leftmost tape symbol. If that symbol was a blank, *accept*. If that symbol was a #, continue with the next stage. Otherwise, *reject*.
2. Scan right to the next # and place a second mark on top of it. If no # is encountered before a blank symbol, only  $x_1$  was present, so *accept*.
3. By zig-zagging, compare the two strings to the right of the marked #s. If they are equal, *reject*.
4. Move the rightmost of the two marks to the next # symbol to the right. If no # symbol is encountered before a blank symbol, move the leftmost mark to the next # to its right and the rightmost mark to the # after that. This time, if no # is available for the rightmost mark, all the strings have been compared, so *accept*.
5. Go to stage 3.”

#101#11#1#1011  
#101#11#1#1011  
#101#11#1#1011  
#101#11#1#1011  
#101#11#1#1011  
#101#11#1#1011  
#101#11#1#1011

## Example 8

$$E = \{\#x_1\#x_2\#\cdots\#x_l | \text{each } x_i \in \{0,1\}^* \text{ and } x_i \neq x_j \text{ for each } i \neq j\}$$

$M_4$  = “On input  $w$ :

1. Place a mark on top of the leftmost tape symbol. If that symbol was a blank, *accept*. If that symbol was a #, continue with the next stage. Otherwise, *reject*.
2. Scan right to the next # and place a second mark on top of it. If no # is encountered before a blank symbol, only  $x_1$  was present, so *accept*.
3. By zig-zagging, compare the two strings to the right of the marked #s. If they are equal, *reject*.
4. Move the rightmost of the two marks to the next # symbol to the right. If no # symbol is encountered before a blank symbol, move the leftmost mark to the next # to its right and the rightmost mark to the # after that. This time, if no # is available for the rightmost mark, all the strings have been compared, so *accept*.
5. Go to stage 3.”

#101#11#1#1011

#101#11#1#1011

#101#11#1#1011

#101#11#1#1011

#101#11#1#1011

#101#11#1#1011

#101#11#1#1011

## Example 8

$$E = \{\#x_1\#x_2\#\cdots\#x_l | \text{each } x_i \in \{0,1\}^* \text{ and } x_i \neq x_j \text{ for each } i \neq j\}$$

$M_4$  = “On input  $w$ :

1. Place a mark on top of the leftmost tape symbol. If that symbol was a blank, *accept*. If that symbol was a #, continue with the next stage. Otherwise, *reject*.
2. Scan right to the next # and place a second mark on top of it. If no # is encountered before a blank symbol, only  $x_1$  was present, so *accept*.
3. By zig-zagging, compare the two strings to the right of the marked #s. If they are equal, *reject*.
4. Move the rightmost of the two marks to the next # symbol to the right. If no # symbol is encountered before a blank symbol, move the leftmost mark to the next # to its right and the rightmost mark to the # after that. This time, if no # is available for the rightmost mark, all the strings have been compared, so *accept*.
5. Go to stage 3.”

#101#11#1#1011

#101#11#1#1011

#101#11#1#1011

#101#11#1#1011

#101#11#1#1011

#101#11#1#1011

#101#11#1#1011

## Example 8

$$E = \{\#x_1\#x_2\#\cdots\#x_l | \text{each } x_i \in \{0,1\}^* \text{ and } x_i \neq x_j \text{ for each } i \neq j\}$$

$M_4$  = “On input  $w$ :

1. Place a mark on top of the leftmost tape symbol. If that symbol was a blank, *accept*. If that symbol was a #, continue with the next stage. Otherwise, *reject*.
2. Scan right to the next # and place a second mark on top of it. If no # is encountered before a blank symbol, only  $x_1$  was present, so *accept*.
3. By zig-zagging, compare the two strings to the right of the marked #s. If they are equal, *reject*.
4. Move the rightmost of the two marks to the next # symbol to the right. If no # symbol is encountered before a blank symbol, move the leftmost mark to the next # to its right and the rightmost mark to the # after that. This time, if no # is available for the rightmost mark, all the strings have been compared, so *accept*.
5. Go to stage 3.”

#101#11#1#1011

#101#11#1#1011

#101#11#1#1011

#101#11#1#1011

#101#11#1#1011

#101#11#1#1011

#101#11#1#1011

## Example 8

$$E = \{\#x_1\#x_2\#\cdots\#x_l | \text{each } x_i \in \{0,1\}^* \text{ and } x_i \neq x_j \text{ for each } i \neq j\}$$

$M_4$  = “On input  $w$ :

1. Place a mark on top of the leftmost tape symbol. If that symbol was a blank, *accept*. If that symbol was a #, continue with the next stage. Otherwise, *reject*.
2. Scan right to the next # and place a second mark on top of it. If no # is encountered before a blank symbol, only  $x_1$  was present, so *accept*.
3. By zig-zagging, compare the two strings to the right of the marked #s. If they are equal, *reject*.
4. Move the rightmost of the two marks to the next # symbol to the right. If no # symbol is encountered before a blank symbol, move the leftmost mark to the next # to its right and the rightmost mark to the # after that. This time, if no # is available for the rightmost mark, all the strings have been compared, so *accept*.
5. Go to stage 3.”

#101#11#1#1011

#101#11#1#1011

#101#11#1#1011

#101#11#1#1011

#101#11#1#1011

#101#11#1#1011

#101#11#1#1011

ACCEPT

## 3.2 VARIANTS OF TURING MACHINES

## VARIANTS OF TURING MACHINES

- Alternative definitions of Turing machines abound,
  - including versions with **multiple tapes** or with **nondeterminism**.
  - Called **variants** of the Turing machine model.

## VARIANTS OF TURING MACHINES

- Alternative definitions of Turing machines abound,
  - including versions with **multiple tapes** or with **nondeterminism**.
  - Called **variants** of the Turing machine model.
- The **original** model and its reasonable **variants**
  - have the **same power**
  - recognize the **same class of languages**.

## VARIANTS OF TURING MACHINES

- Example: Suppose that we had allowed the Turing machine the ability to stay put.
  - The **transition function** would then have the form

$$\delta: Q \times \Gamma \rightarrow Q \times \Gamma \times \{L, R, S\}.$$

- Any additional power?
- Any additional languages can be recognized with this feature?

## VARIANTS OF TURING MACHINES

- Suppose that we had allowed the Turing machine the ability to **stay put**.
  - The transition function would then have the form

$$\delta: Q \times \Gamma \rightarrow Q \times \Gamma \times \{L, R, S\}.$$

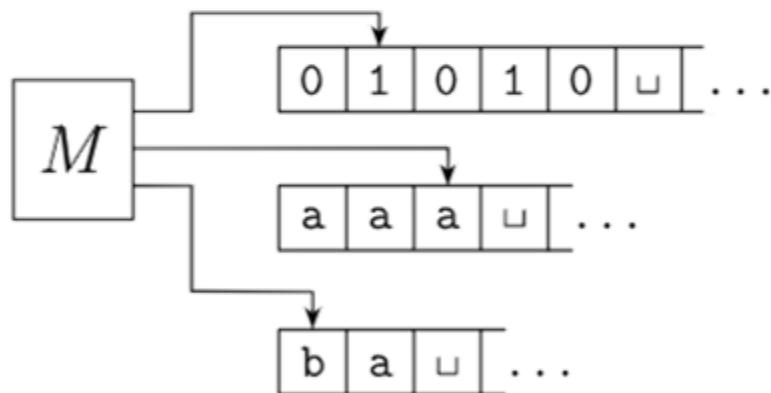
- Any additional power?
- Any additional languages can be recognized with this feature?
- **Of course not.**
  - because we can **convert** any **TM with the “stay put” feature** to **one that does not have it**.
    - by replacing each **stay put** transition **with two transitions**:
      - **move to the right** and
      - **back to the left**.

## VARIANTS OF TURING MACHINES

- To show that two **models are equivalent**, we simply need to show that
  - one can simulate the other.

## MULTITAPE TURING MACHINES

- A **multi tape Turing machine** is like an ordinary Turing machine with **several tapes**.
- Each tape has its **own head** for reading and writing.
- Initially the **input appears on tape 1**, and the **others will be blank**



An example of a TM with 3 tapes

## MULTITAPE TURING MACHINES

- The **transition function** is changed to allow for reading, writing, and moving the heads on some or all of the tapes simultaneously.

$$\delta: Q \times \Gamma^k \longrightarrow Q \times \Gamma^k \times \{L, R, S\}^k,$$

Where  $k$  is the **number of tapes**.

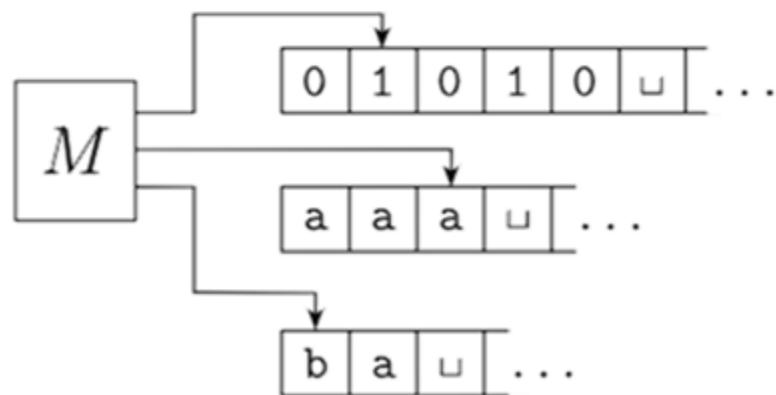
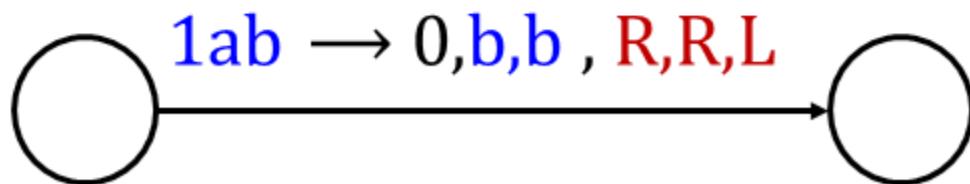
## MULTITAPE TURING MACHINES

- The **transition function** is changed to allow for reading, writing, and moving the heads on some or all of the tapes simultaneously.

$$\delta: Q \times \Gamma^k \longrightarrow Q \times \Gamma^k \times \{L, R, S\}^k,$$

Where  $k$  is the **number of tapes**.

$$\delta(q_i, a_1, \dots, a_k) = (q_j, b_1, \dots, b_k, L, R, \dots, L)$$



## MULTITAPE TURING MACHINES

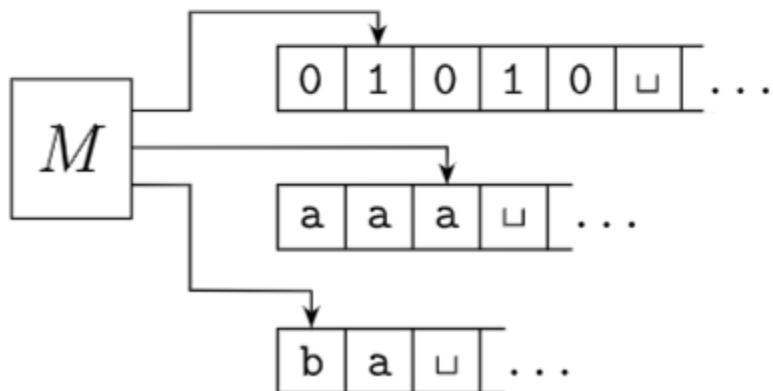
- **Theorem :** Every multitape Turing machine has an equivalent single-tape Turing

## MULTITAPE TURING MACHINES

- Theorem : Every **multitape Turing machine** has an **equivalent single-tape Turing**

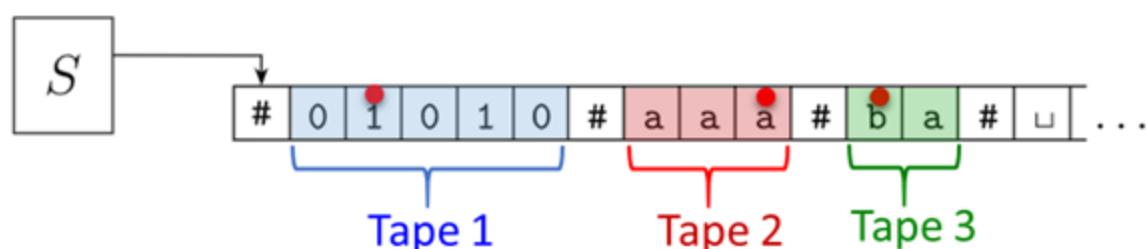
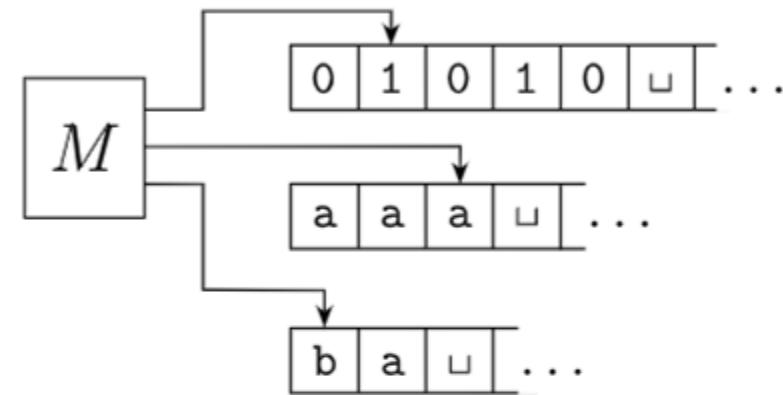
## MULTITAPE TURING MACHINES

- Theorem : Every **multitape Turing machine** has an **equivalent single-tape Turing**
- Proof Idea:
  - We show how to **convert** a **multitape TM  $M$**  to an equivalent **single-tape TM  $S$** .



## MULTITAPE TURING MACHINES

- Proof Idea(cont.):
  - Say that  $M$  has **k tapes**.
  - Then  $S$  simulates the effect of  $k$  tapes by
    - storing their information on its **single tape**.
  - It uses the **new symbol # as a delimiter**
    - to **separate** the contents of the different tapes.
  - In addition to the contents of these tapes,
    - $S$  must keep track of the **locations of the heads**.
      - by writing **a tape symbol with a dot** above it
      - to mark the place where the head on that tape would be.



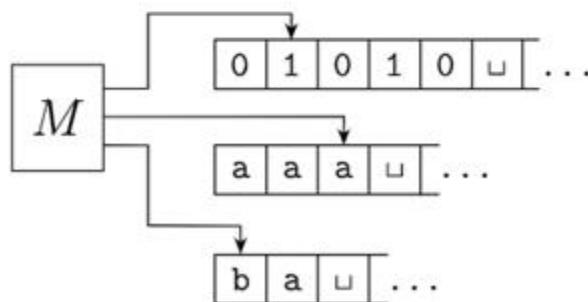
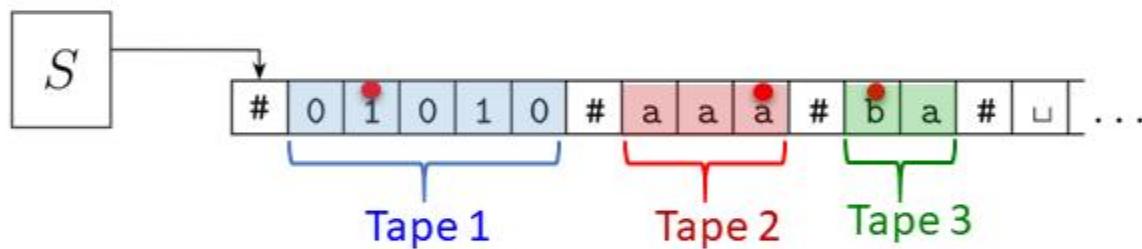
# MULTITAPE TURING MACHINES

- Proof :

$S$  = “On input  $w = w_1 \dots w_n$ :

1. First  $S$  puts its tape into the format that represents all  $k$  tapes of  $M$ . The formatted tape contains

$$\# w_1 w_2 \dots w_n \# \sqcup \# \sqcup \# \dots \#.$$



## MULTITAPE TURING MACHINES

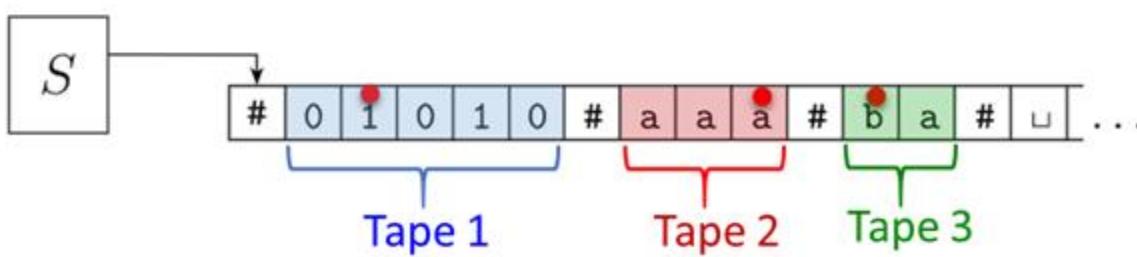
- Proof (cont.):

$S$  = “On input  $w = w_1 \dots w_n$ :

1. First  $S$  puts its tape into the format that represents all  $k$  tapes of  $M$ . The formatted tape contains

$$\# \overset{\bullet}{w_1} w_2 \dots w_n \# \overset{\bullet}{\sqcup} \# \overset{\bullet}{\sqcup} \# \dots \#.$$

2. To simulate a single move,  $S$  scans its tape from the first  $\#$ , which marks the left-hand end, to the  $(k + 1)$ st  $\#$ , which marks the right-hand end, in order to determine the symbols under the virtual heads. Then  $S$  makes a second pass to update the tapes according to the way that  $M$ 's transition function dictates.

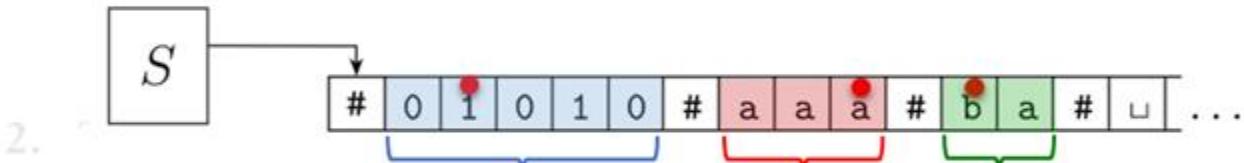


## MULTITAPE TURING MACHINES

- Proof (cont.):

$S$  = “On input  $w = w_1 \dots w_n$ :

1. First  $S$  puts its tape into the format that represents all  $k$  tapes of  $M$ . The formalized tape contains



- 2.

which marks the left-hand end, to the  $(k+1)$ st # which marks the right-hand end, in order to determine the symbols under the virtual heads. Then  $S$  makes a second pass to update the tapes according to the way that  $M$ 's transition function dictates.

3. If at any point  $S$  moves one of the virtual heads to the right onto a #, this action signifies that  $M$  has moved the corresponding head onto the previously unread blank portion of that tape. So  $S$  writes a blank symbol on this tape cell and shifts the tape contents, from this cell until the rightmost #, one unit to the right. Then it continues the simulation as before.”

## MULTITAPE TURING MACHINES

- **Corollary :** A language is Turing-recognizable **if and only if** some multitape Turing machine recognizes it.

A language is Turing-recognizable  $\leftrightarrow$  multitape Turing machine recognizes it

## MULTITAPE TURING MACHINES

- **Corollary :** A language is Turing-recognizable **if and only if** some multitape Turing machine recognizes it.

A language is Turing-recognizable  $\leftrightarrow$  multitape Turing machine recognizes it

- **Proof:**
  - **Part a ( $\rightarrow$ ):**
    - A **Turing-recognizable** language is **recognized by a single-tape** Turing machine, which is a **special case of a multitape** Turing machine.

## MULTITAPE TURING MACHINES

- **Corollary :** A language is Turing-recognizable **if and only if** some multitape Turing machine recognizes it.

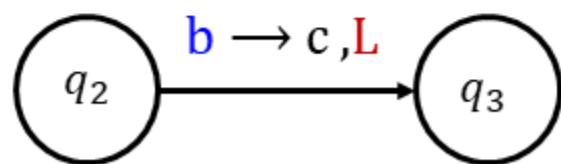
A language is Turing-recognizable  $\leftrightarrow$  multitape Turing machine recognizes it

- **Proof:**
  - **Part a ( $\rightarrow$ ):**
    - A Turing-recognizable language is recognized by an ordinary(single-tape) Turing machine, which is a special case of a multitape Turing machine.
  - **Part b ( $\leftarrow$ ):**
    - A **multitape** Turing machine **recognizes the language**.
      - Its **equivalent single-tape TM** **recognizes it**.
      - Then, it is **Turing-recognizable**.

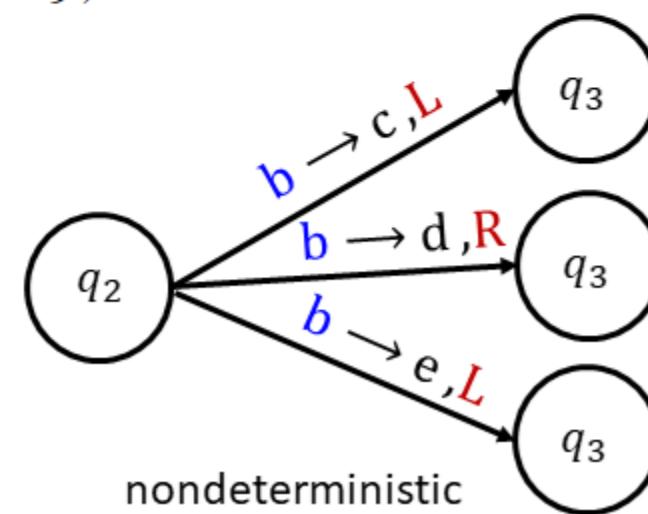
## NONDETERMINISTIC TURING MACHINES

- In a **nondeterministic Turing machine** :
  - **At any point in a computation**, the machine may **proceed** according to **several possibilities**.
- The **transition function** for a nondeterministic Turing machine has the form

$$\delta: Q \times \Gamma \longrightarrow \mathcal{P}(Q \times \Gamma \times \{L, R\}).$$



deterministic



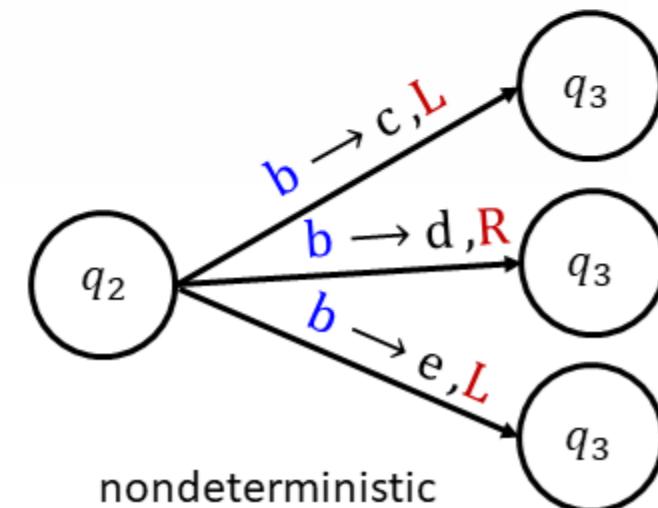
nondeterministic

## NONDETERMINISTIC TURING MACHINES

- In a **nondeterministic Turing machine** :
  - At any point in a computation, the machine may proceed according to several possibilities.
- The **transition function** for a nondeterministic Turing machine has the form

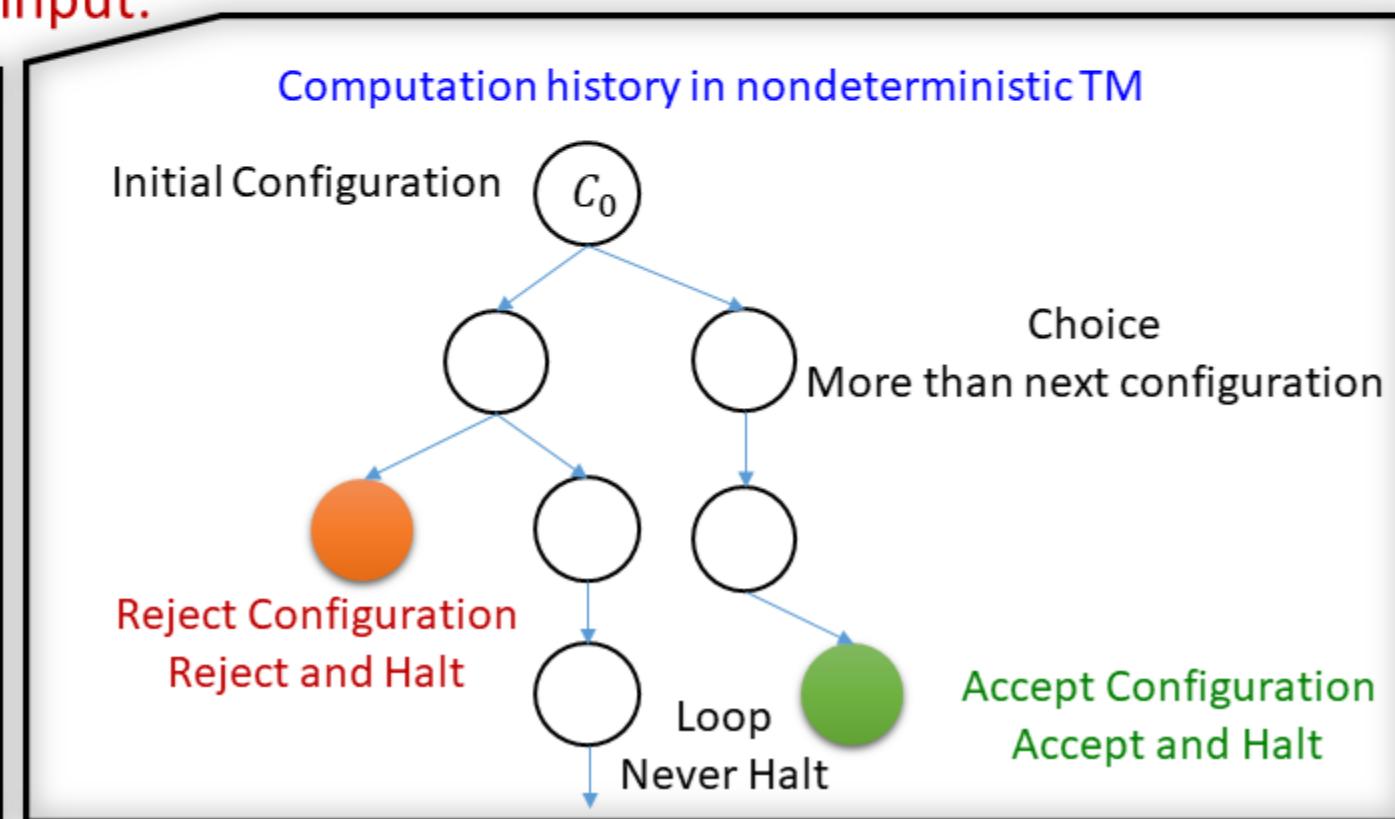
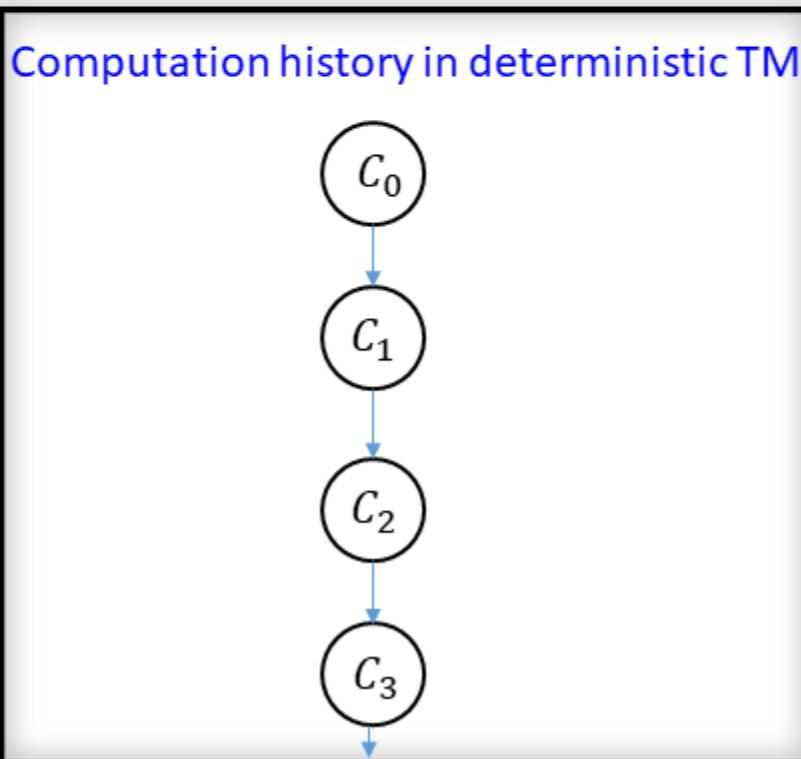
$$\delta: Q \times \Gamma \longrightarrow \mathcal{P}(Q \times \Gamma \times \{\text{L}, \text{R}\}).$$

- **At each moment** in the computation there can be **more than one** successor configuration.



## NONDETERMINISTIC TURING MACHINES

- The **computation** of a nondeterministic Turing machine **is a tree**
  - whose **branches** correspond to **different possibilities for the machine**.
- If **some branch** of the computation **leads to the accept state**,
  - the machine accepts its input.



## OUTCOMES OF A NONDETERMINISTIC TURING MACHINES

- **ACCEPT:**
  - If **any branch** of the computation accepts, then the NDTM will accept.
- **REJECT:**
  - If **all branches** of the computation halt and reject, then the NDTM rejects.
- **LOOP:**
  - Computation continues, but **accept** is **never** encountered.
  - **Some branches** in the computation history are **infinite**.

**Note:** NDTM = NONDETERMINISTIC TURING MACHINES

## NONDETERMINISTIC TURING MACHINES

- **Theorem :** Every nondeterministic Turing machine has an equivalent deterministic Turing machine.

## NONDETERMINISTIC TURING MACHINES

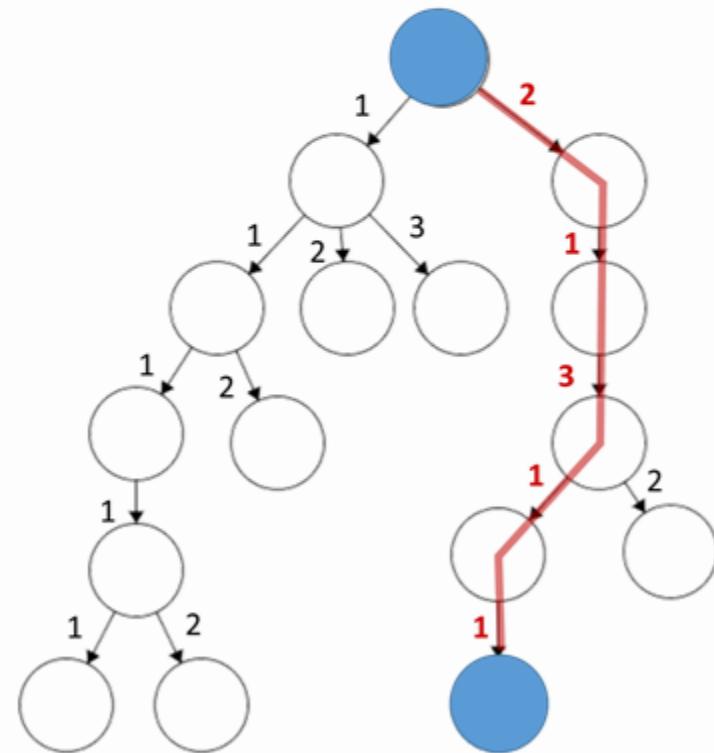
- **Theorem :** Every nondeterministic Turing machine has an equivalent deterministic Turing machine.
- **Proof Idea:**
  - We have to show how we can construct an equivalent of DTM (**D**) for a NDTM (**N**)
    - If **N** accepts (on any branch), then **D will accept.**
    - If **N halts on every branch without any accepts**, then **D will Halt and reject.**

## NONDETERMINISTIC TURING MACHINES

- **Theorem :** Every nondeterministic Turing machine has an equivalent deterministic Turing machine.
- **Proof Idea:**
  - We have to show how we can construct an equivalent of DTM (**D**) for a NDTM (**N**)
    - If **N** accepts (on any branch), then **D will accept.**
    - If **N halts on every branch without any accepts**, then **D will Halt and reject.**

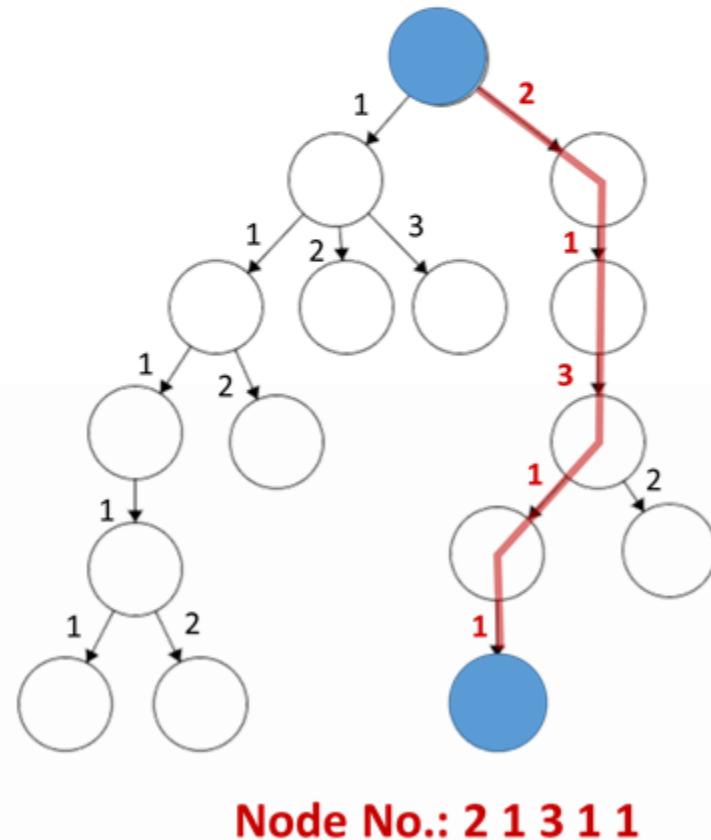
## NONDETERMINISTIC TURING MACHINES

- Proof Idea (cont.):
  - How to search the tree, looking for “ACCEPT” ?
  - How to show any node in a **computation history** by a **number** ?



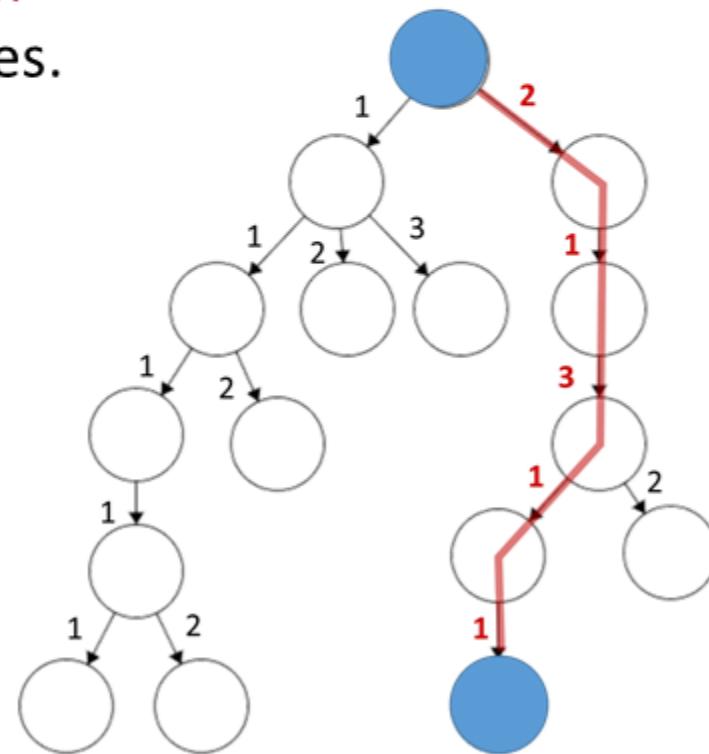
## NONDETERMINISTIC TURING MACHINES

- Proof Idea (cont.):
  - How to search the tree, looking for “ACCEPT” ?
  - How to show any node in a **computation history** by a **number** ?
- Which one of the following Search Method ?
  - Depth-First Search
  - Breadth-First Search
- The **depth-first search** strategy goes all



# NONDETERMINISTIC TURING MACHINES

- Proof Idea (cont.):
    - Depth-First Search
      - This strategy **goes all the way down one branch**
        - before backing up to explore other branches.
      - could **go forever down** one **infinite branch**
        - and **miss an accepting configuration** on some other branch.

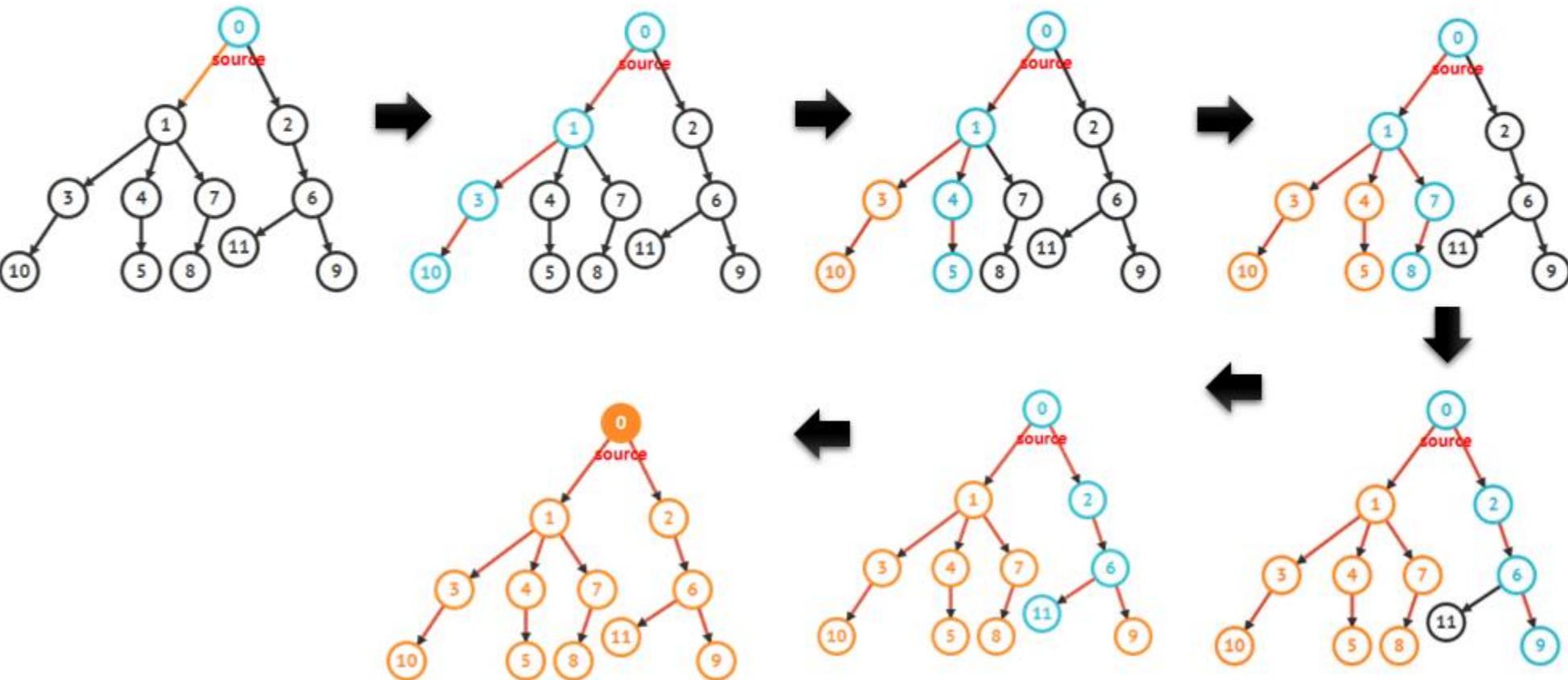


Node No.: 21311

### Chapter 3

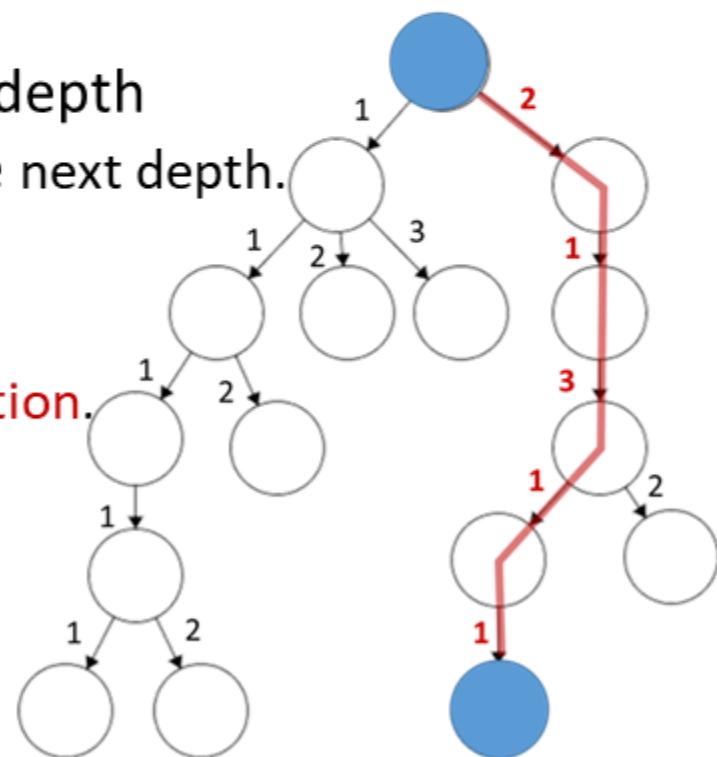
## NONDETERMINISTIC TURING MACHINES

- Proof Idea (cont.):
  - Depth-First Search



## NONDETERMINISTIC TURING MACHINES

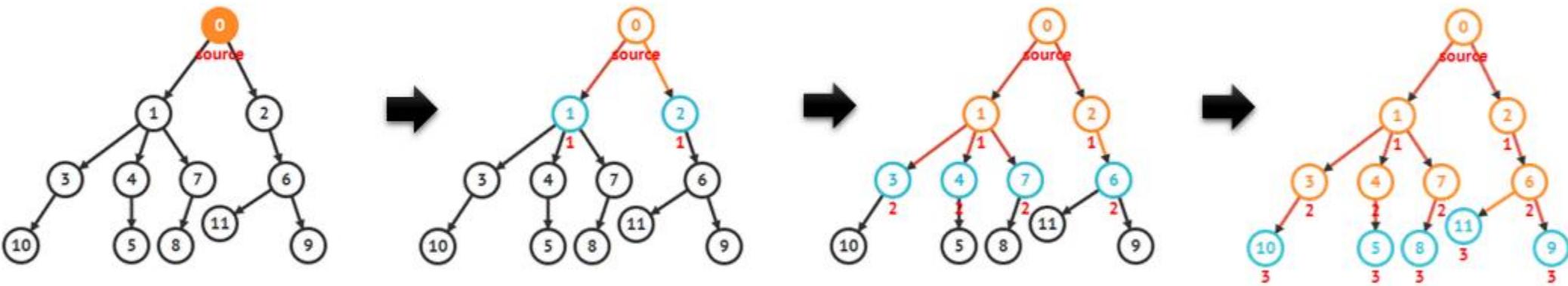
- Proof Idea (cont.):
  - Hence we design D to **explore the tree** by using **breadth-first search** instead.
  - Breadth-First Search
    - This strategy explores all branches to the same depth
      - before going onto explore any branch to the next depth.
    - This method **guarantees** that
      - D will **visit every node** in the tree
        - until it encounters an **accepting configuration**.



Node No.: 2 1 3 1 1

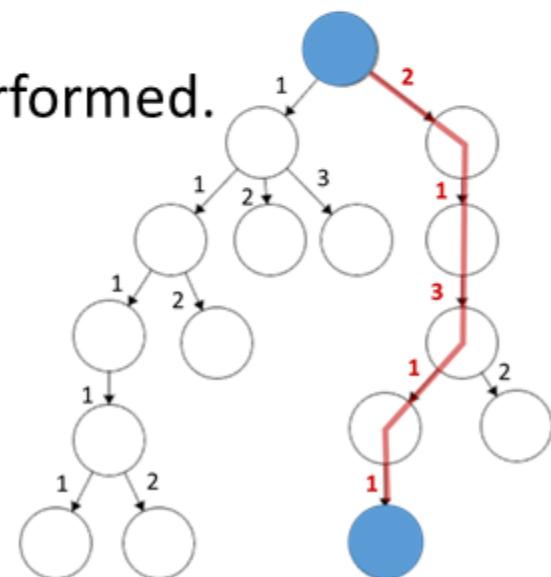
# NONDETERMINISTIC TURING MACHINES

- Proof Idea (cont.):
  - Breadth-First Search



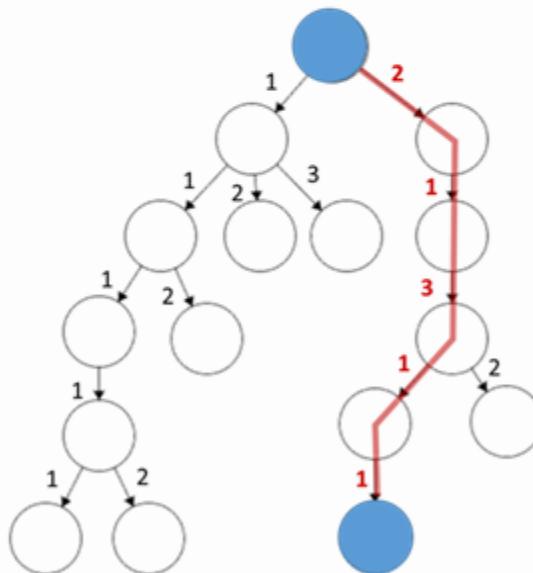
## NONDETERMINISTIC TURING MACHINES

- Proof Idea (cont.):
  - **Searching Strategy: Breadth-First Search**
- We need a numbering system for the nodes (**node no.**) to choose the **next node for searching** among of the many Nondeterministic choices.
- Our numbering system should support in idea that
  - **to examine a node** (configuration)
    - the **entire computation** from scratch should be performed.



## NONDETERMINISTIC TURING MACHINES

- Proof Idea (cont.):
  - **Searching Strategy: Breadth-Firth Search**
  - How many choices are there at each step in the computation ?



## NONDETERMINISTIC TURING MACHINES

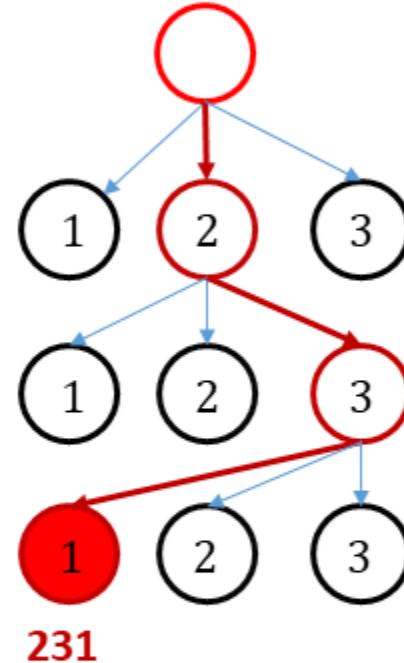
- Proof Idea (cont.):
  - **Searching Strategy: Breadth-Firth Search**
  - How many choices are there at each step in the computation ?
  - Every node in the tree can have at most **b** children,
    - where **b** is the size of the largest set of possible choices given by N's transition function.

## NONDETERMINISTIC TURING MACHINES

- Proof Idea (cont.):
  - Searching Strategy: Breadth-First Search
  - How many choices are there at each step in the computation ?
  - Every node in the tree can have at most  $b$  children,
    - where  $b$  is the size of the largest set of possible choices given by  $N$ 's transition function.
  - Then, to every node in the tree we assign an address (node. no)
    - that is a string over the alphabet  $\Gamma_b = \{1, 2, \dots, b\}$ .
    - and shows the Path of computation from initial configuration to that node.

## NONDETERMINISTIC TURING MACHINES

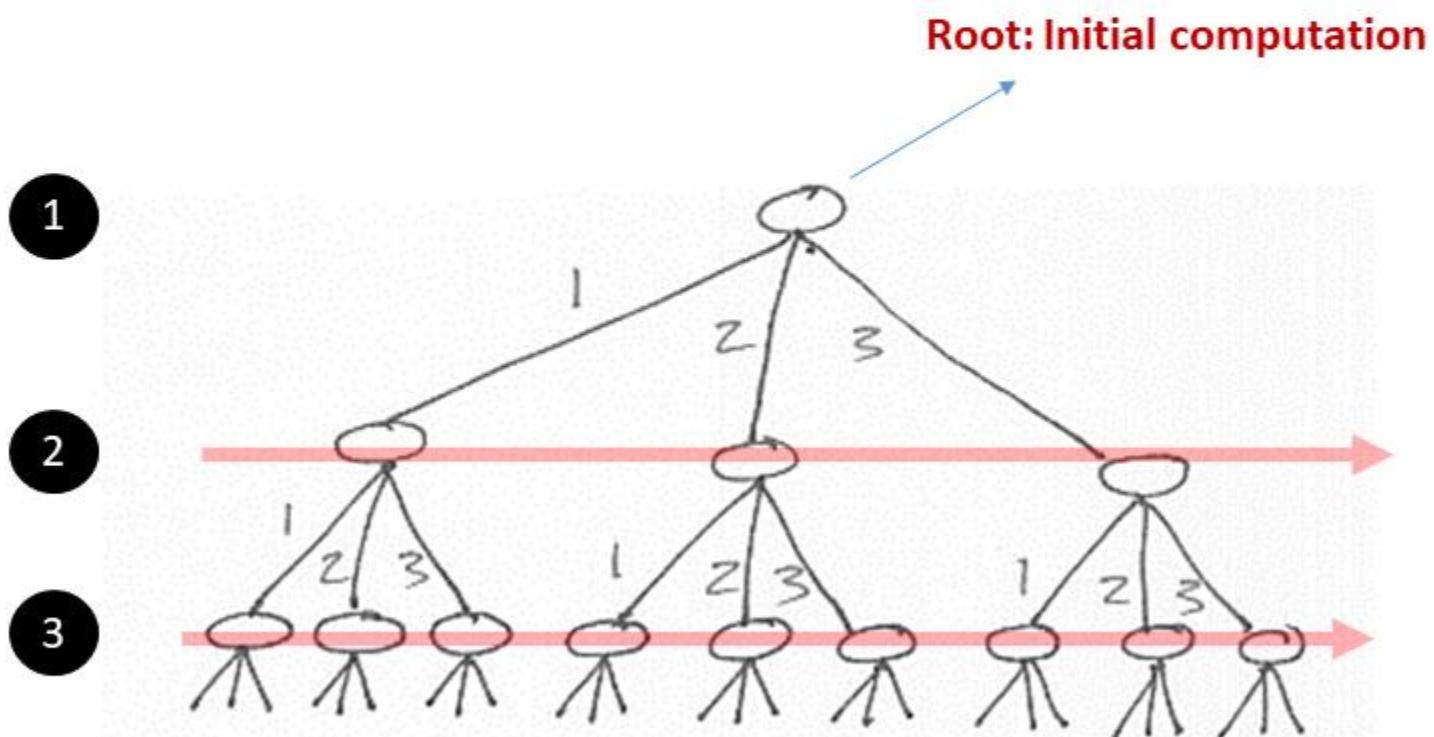
- Proof Idea (cont.):
  - Suppose :  $\Gamma_b = \{1,2,3\}$  in the computation tree.
  - **Address 231 :**
    - starting at the **root**,
    - going to its **2nd child**,
    - going to that node's **3rd child**,
    - and finally going to that node's **1st child**.



## NONDETERMINISTIC TURING MACHINES

- Proof Idea (cont.):

- Breadth-First Search Order

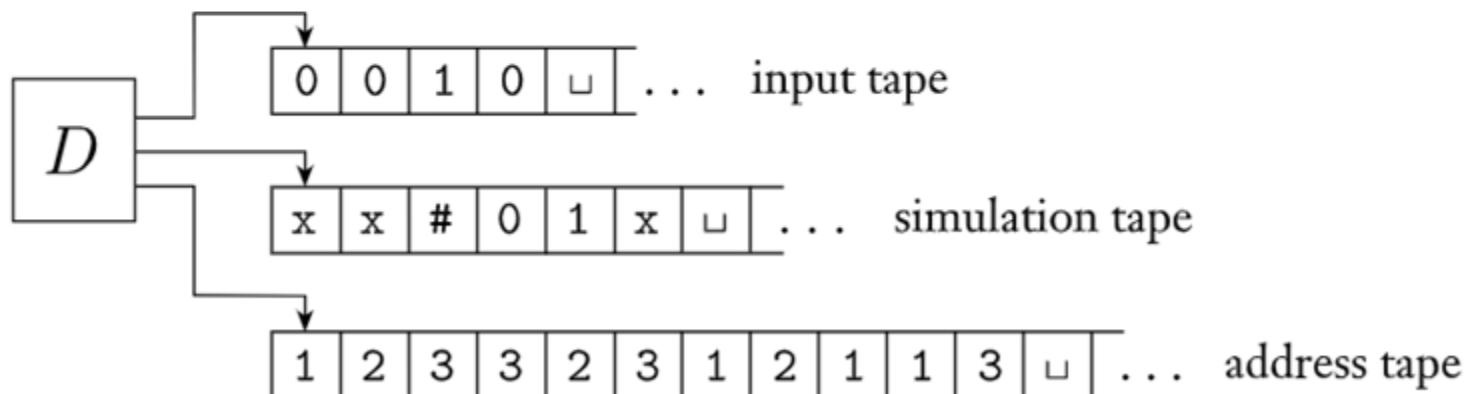


$\varepsilon$
1
2
3
11
12
13
21
22
23
31
32
33
111
112
113
121
122
123

Counting in base 3

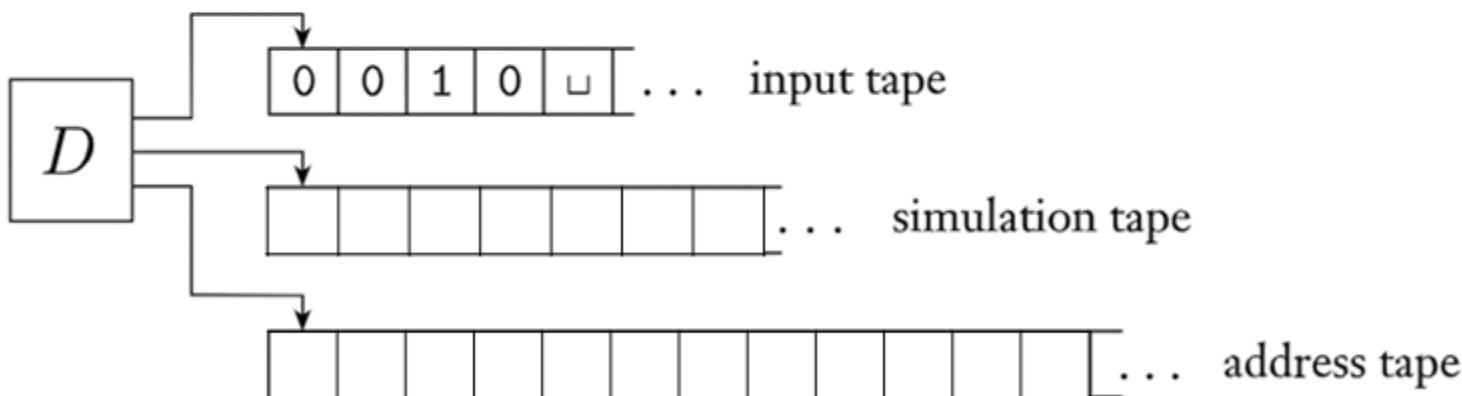
## NONDETERMINISTIC TURING MACHINES

- Proof :
  - The simulating deterministic  $TM D$  has **three tapes**.
    - This arrangement is **equivalent** to having **a single tape**.
  - **Tap e 1:** always contains the **input string** and is **never altered** .
  - **Tap e 2:** maintains a copy of **N's tape** on some branch of its nondeterministic computation.
  - **Tape 3:** keeps track of D's location in N's nondeterministic computation tree.



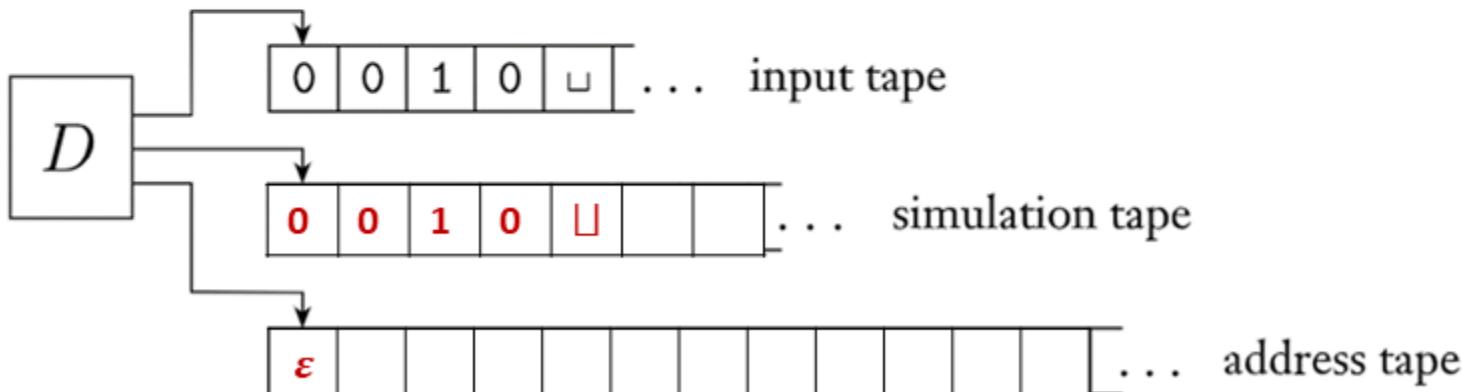
## NONDETERMINISTIC TURING MACHINES

- **Theorem :** Every nondeterministic Turing machine has an equivalent deterministic Turing machine.
- **Proof (cont.) :**
  - 1. Initially, tape 1 contains the input  $w$ , and tapes 2 and 3 are empty.



## NONDETERMINISTIC TURING MACHINES

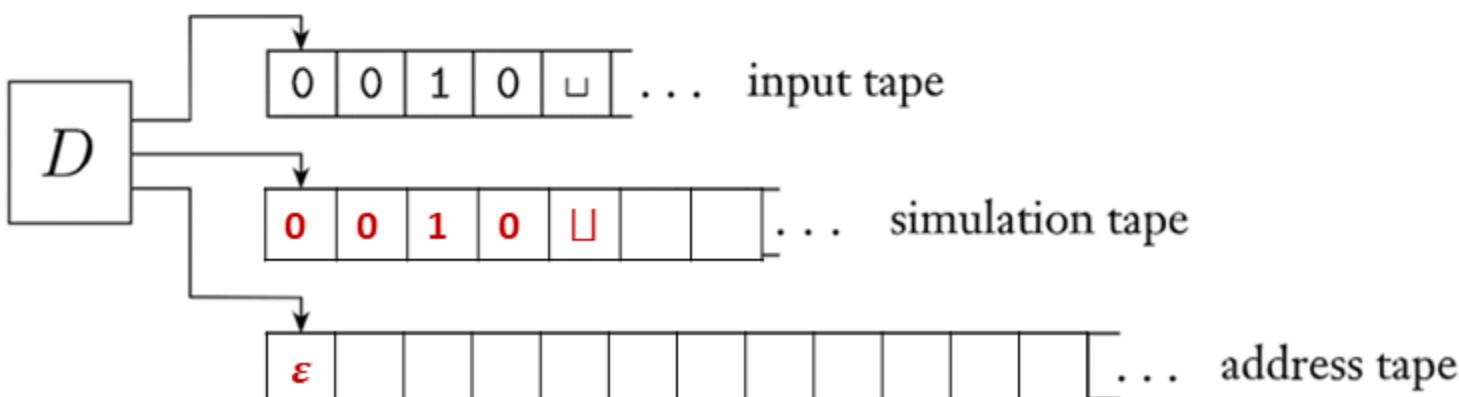
- **Theorem :** Every nondeterministic Turing machine has an equivalent deterministic Turing machine.
- **Proof :**
  - 1. Initially, tape 1 contains the input  $w$ , and tapes 2 and 3 are empty.
  - 2. Copy tape 1 to tape 2 and initialize the string on tape 3 to be  $\epsilon$ .



## NONDETERMINISTIC TURING MACHINES

- Proof :

1. Initially, tape 1 contains the input  $w$ , and tapes 2 and 3 are empty.
2. Copy tape 1 to tape 2 and initialize the string on tape 3 to be  $\epsilon$ .
3. Use tape 2 to simulate  $N$  with input  $w$  on one branch of its nondeterministic computation. Before each step of  $N$ , consult the next symbol on tape 3 to determine which choice to make among those allowed by  $N$ 's transition function. If no more symbols remain on tape 3 or if this nondeterministic choice is invalid, abort this branch by going to stage 4. Also go to stage 4 if a rejecting configuration is encountered. If an accepting configuration is encountered, accept the input.



## NONDETERMINISTIC TURING MACHINES

- Proof :

1. Initially, tape 1 contains the input  $w$ , and tapes 2 and 3 are empty.
2. Copy tape 1 to tape 2 and initialize the string on tape 3 to be  $\epsilon$ .
3. Use tape 2 to simulate  $N$  with input  $w$  on one branch of its nondeterministic computation. Before each step of  $N$ , consult the next symbol on tape 3 to determine which choice to make among those allowed by  $N$ 's transition function. If no more symbols remain on tape 3 or if this nondeterministic choice is invalid, abort this branch by going to stage 4. Also go to stage 4 if a rejecting configuration is encountered. If an accepting configuration is encountered, *accept* the input.
4. Replace the string on tape 3 with the next string in the string ordering. Simulate the next branch of  $N$ 's computation by going to stage 2.

$\epsilon$	
1	
2	
3	
11	
12	
13	
21	
22	
23	
31	
32	
33	
111	
112	
113	
121	
122	
123	

## NONDETERMINISTIC TURING MACHINES

- **Corollary :** A language is **Turing-recognizable** if and only if **some nondeterministic Turing machine recognizes it.**
- **Proof :**
  - Any DTM is automatically a NDTM, and so one direction of this corollary follows immediately.
  - The other direction follows from the previous Theorem.

## NONDETERMINISTIC TURING MACHINES

- We call a **nondeterministic** Turing machine a **decider**
  - if **all branches halt** on all inputs.
- Corollary : A language is **decidable if and only if** some nondeterministic Turing machine **decides** it.