

Turing Machines

Tanjila Alam Sathi

Lecturer, CSE Department

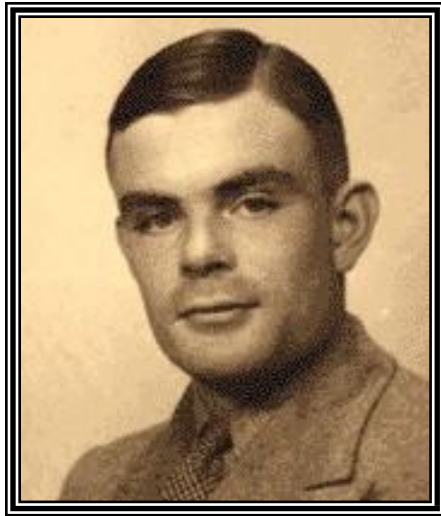
Introduction

- **Undecidable**: specific problems that cannot be solved.

Devices of Increasing Computational Power

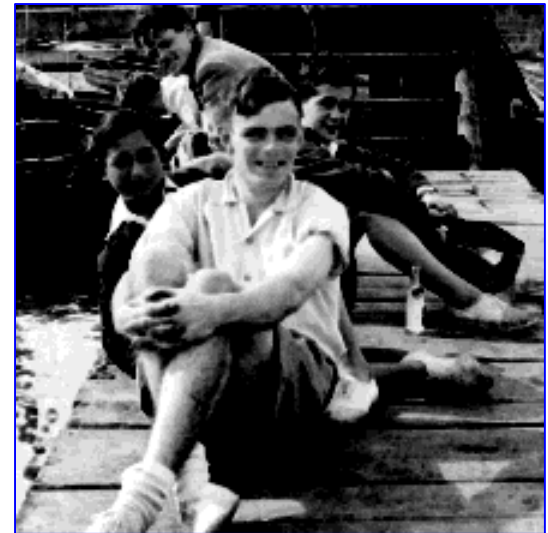
- So far:
 - Finite Automata – good for devices with small amounts of memory, relatively simple control
 - Pushdown Automata – stack-based automata
- But both have limitations for even simple tasks, too restrictive as general purpose computers
- **Enter the Turing Machine**
 - More powerful than either of the above
 - Essentially a finite automaton but with unlimited memory
 - Although theoretical, can do everything a general purpose computer of today can do
 - If a TM can't solve it, neither can a computer

Alan Turing



1912 (23 June): Birth, Paddington, London

1954 (7 June): Death (suicide) by cyanide poisoning, Wilmslow, Cheshire



The Vision

- Alan Turing, regarded by some as the father of computer science, was among the first to envision the power of a computer and the importance of software over hardware
- — He knew the computer would mean unlimited potential in the creation and implementation of artificial intelligence and was determined to be the first to act on this belief.

The Turing Machine

In order to argue for this claim, he needed a clear concept of “mechanical procedure.”

His idea — which came to be called the **Turing machine** — was this:

(1) A tape of infinite length

(2) Finitely many squares of the tape have a single symbol from a finite language.

(3) Someone (or something) that can read the squares and write in them.

(4) At any time, the machine is in one of a finite number of internal states.

(5) The machine has instructions that determine what it does given its internal state and the symbol it encounters on the tape. It can

- ◆ change its internal state;

- ◆ change the symbol on the square;

- ◆ move forward;

- ◆ move backward;

- ◆ halt (i.e. stop).

Functions

It is essential to the idea of a Turing machine that it is not a physical machine, but an abstract one — a set of procedures.

It makes no difference whether the machine is embodied by a person in a boxcar on a track, or a person with a paper and pencil, or a smart and well-trained flamingo.

Turing's Theorem

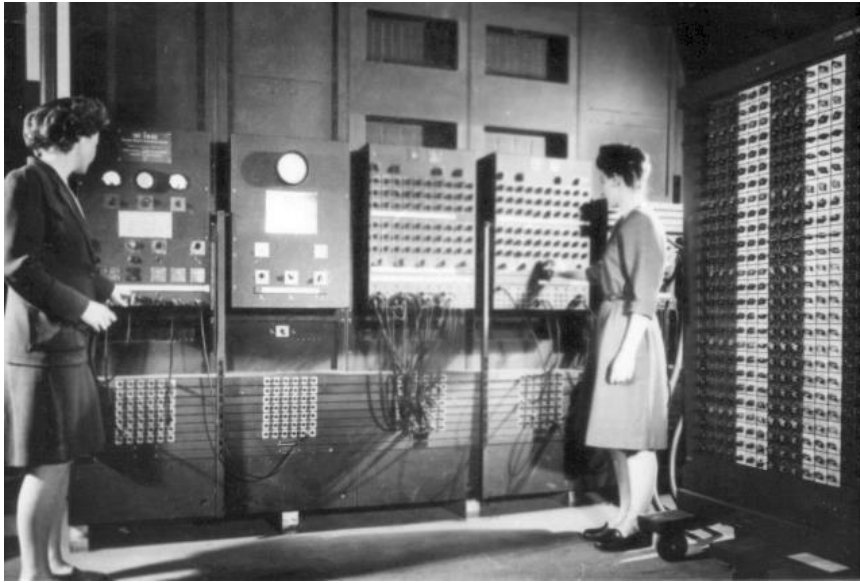
In the 1936 paper Turing proved that there are “general-purpose” Turing machines that can compute whatever any other Turing machine.

This is done by coding the function of the special-purpose machine as instructions of the other machine — that is by “programming” it. This is called Turing's theorem.

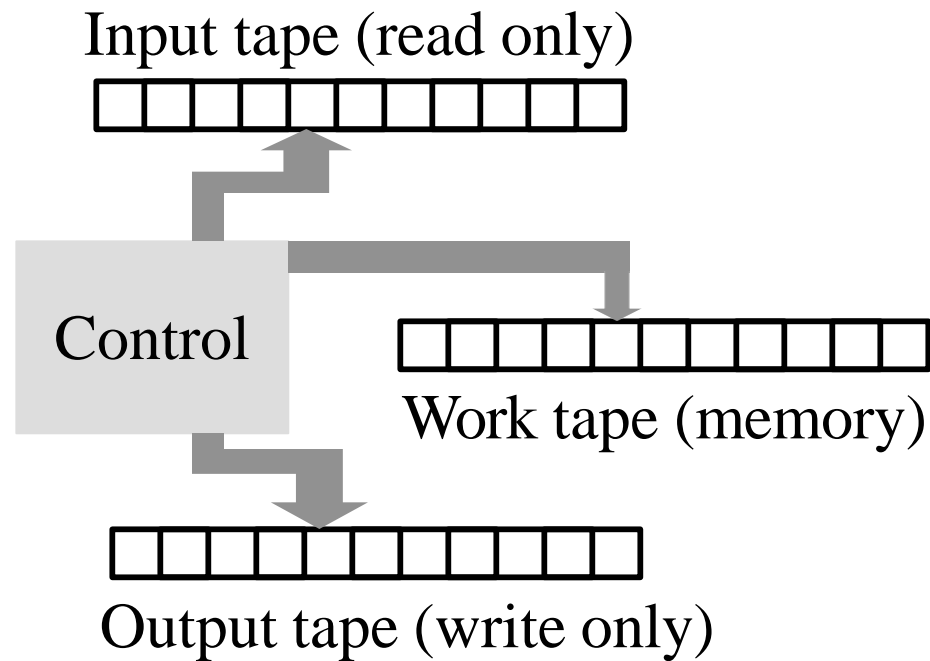
These are **universal Turing machines**, and the idea of a coding for a particular function fed into a universal Turing machine is basically our conception of a computer and a stored program.

The concept of the universal Turing machine is just the concept of the computer as we know it.

First computers: custom computing machines



1950 -- Eniac: the control is hardwired manually for each problem.



1940: **VON NEUMANN:**
DISTINCTION BETWEEN DATA AND INSTRUCTIONS

Can Machines Think?

In “Computing machinery and intelligence,” written in 1950, Turing asks whether machines can think.

He claims that this question is too vague, and proposes, instead, to replace it with a different one.

That question is: Can machines pass the “imitation game” (now called the Turing test)? If they can, they are intelligent.

Turing is thus the first to have offered a rigorous test for the determination of intelligence quite generally.

The Turing Test

The game runs as follows. You sit at a computer terminal and have an electronic conversation. You don't know who is on the other end; it could be a person or a computer responding as it has been programmed to do.

If you can't distinguish between a human being and a computer from your interactions, then the computer is intelligent.

Note that this is meant to be a **sufficient** condition of intelligence only. There may be other ways to be intelligent.

Artificial Intelligence

The Church-Turning Thesis

Turing, and a logician called **Alonzo Church (1903-1995)**, independently developed the idea (not yet proven by widely accepted) that **whatever can be computed by a mechanical procedure can be computed by a Turing machine.**

This is known as the Church-Turing thesis.



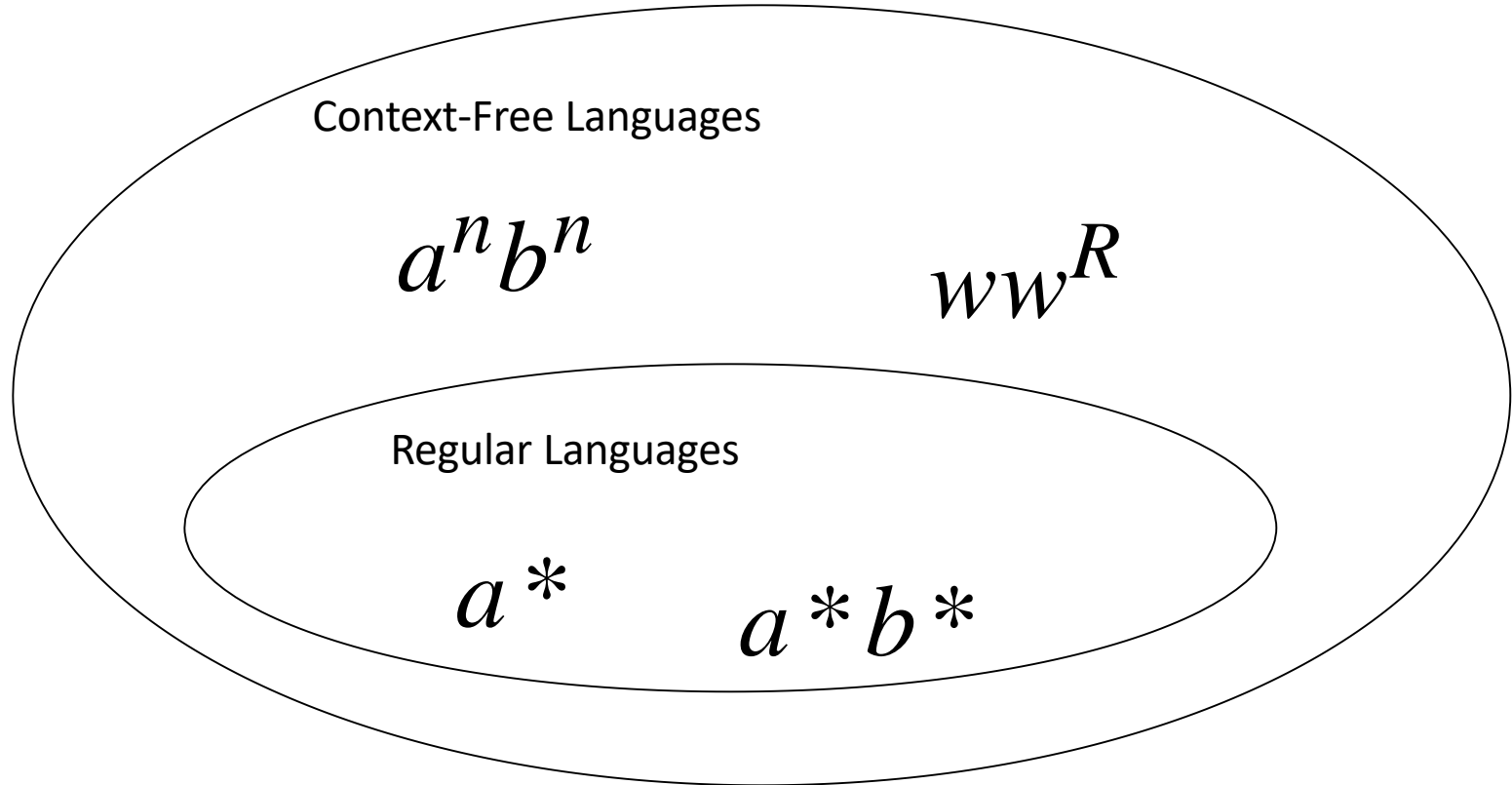
Turing Machines

- ❑ TM's discovered in 1936
- ❑ A TM consists of a **finite control** (i.e. a finite state automaton) that is connected to an infinite tape.
- ❑ There is a tape divided into squares or cells
- ❑ Each cell can hold any one of a finite number of symbols

The Language Hierarchy

$a^n b^n c^n$?

ww ?



Languages accepted by
Turing Machines

$a^n b^n c^n$

ww

Context-Free Languages

$a^n b^n$

ww^R

Regular Languages

a^*

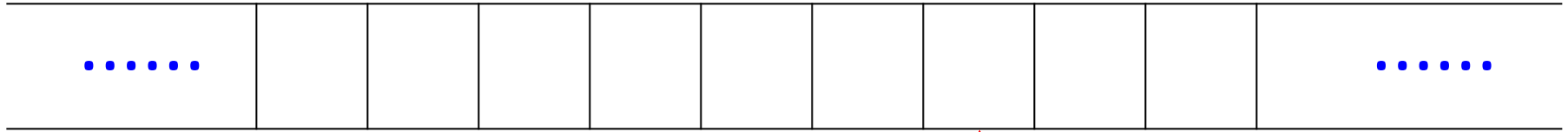
$a^* b^*$

Turing Machines

- ❑ TM's described in 1936
- ❑ A TM consists of a **finite control** (i.e. a finite state automaton) that is connected to an infinite tape.
- ❑ There is a tape divided into squares or cells
 - ❑ Each cell can hold any one of **a finite number of symbols**

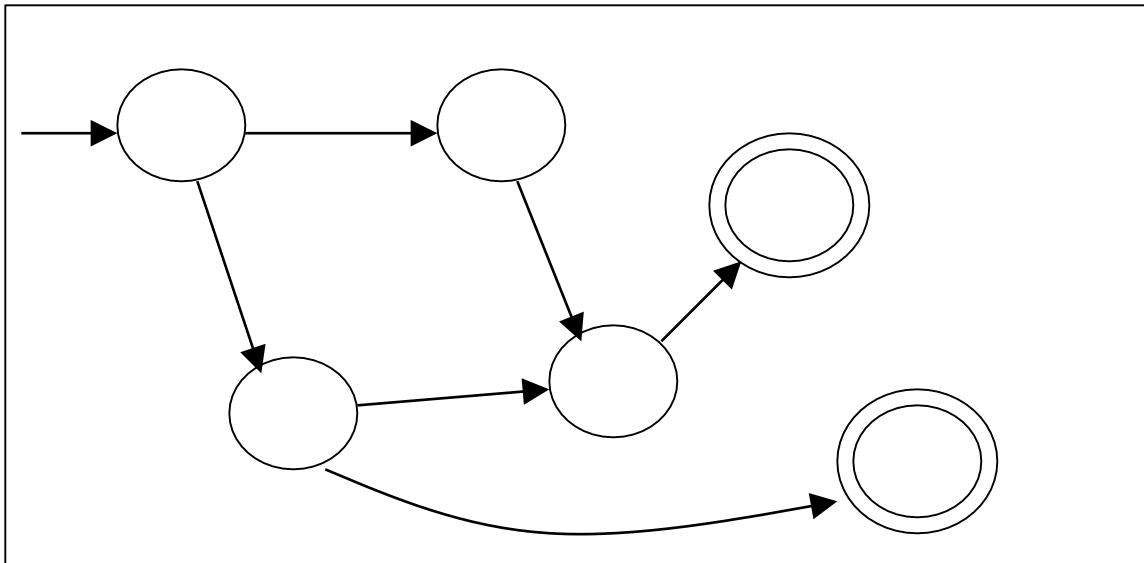
A Turing Machine

Tape



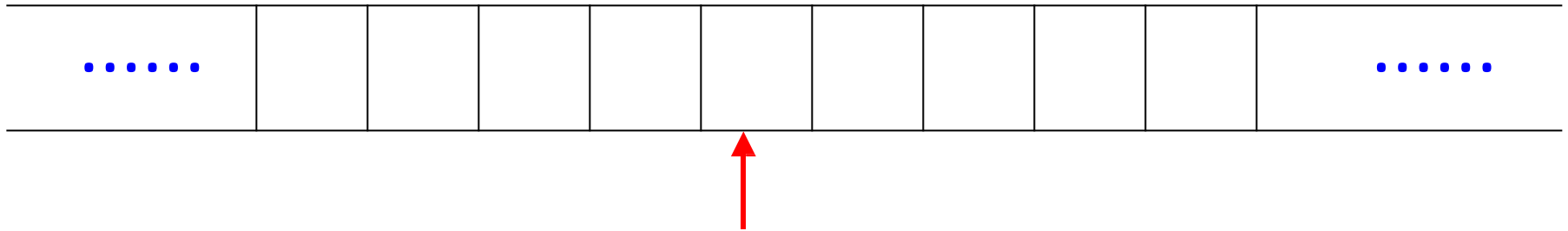
Read-Write head

Control Unit(like FSM or PDA)



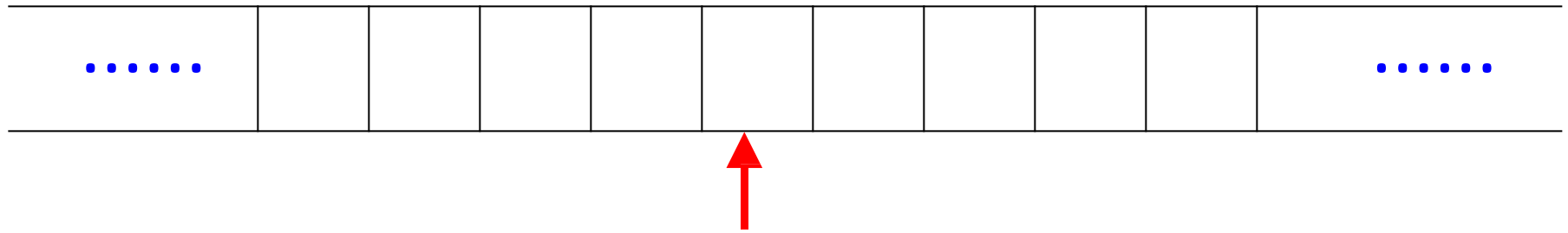
The Tape

No boundaries -- infinite length



Read-Write head

The head moves Left or Right



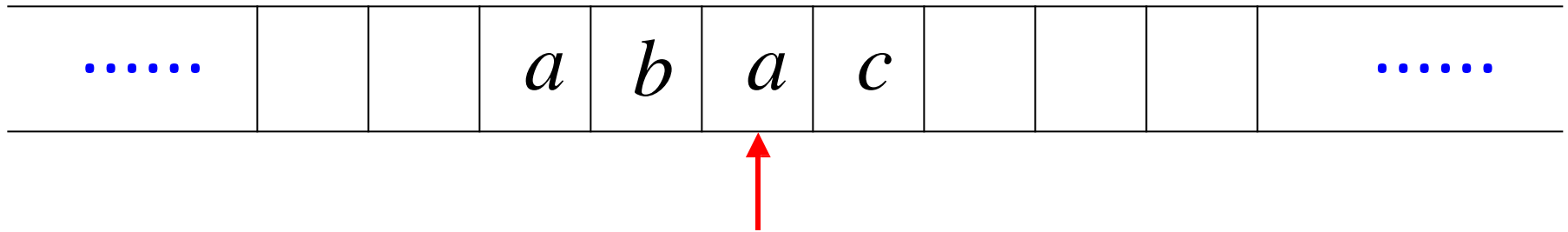
Read-Write head

The head at each time step:

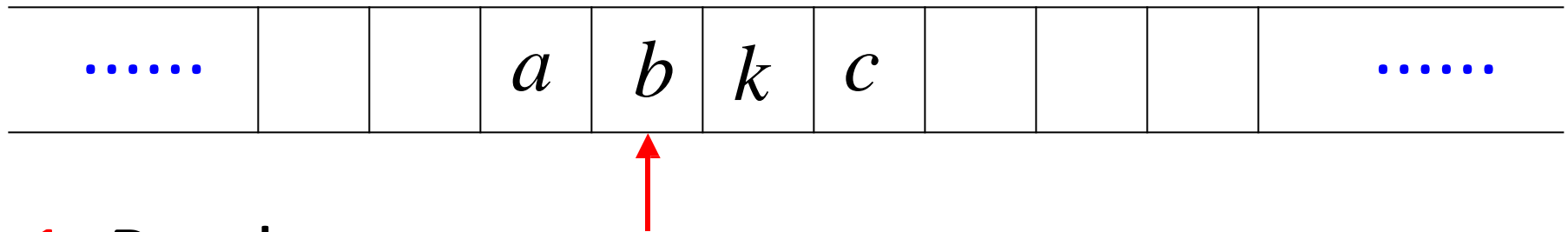
1. Reads a symbol
2. Writes a symbol
3. Moves Left or Right

Example:

Time 0

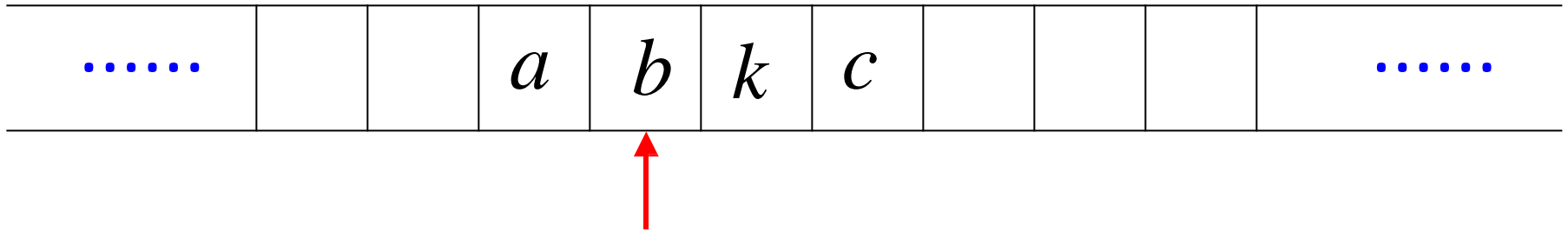


Time 1

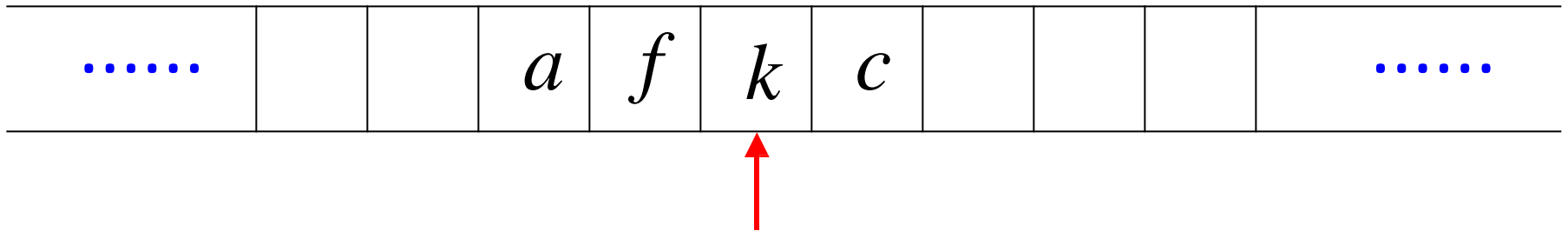


1. Reads a
2. Writes k
3. Moves Left

Time 1

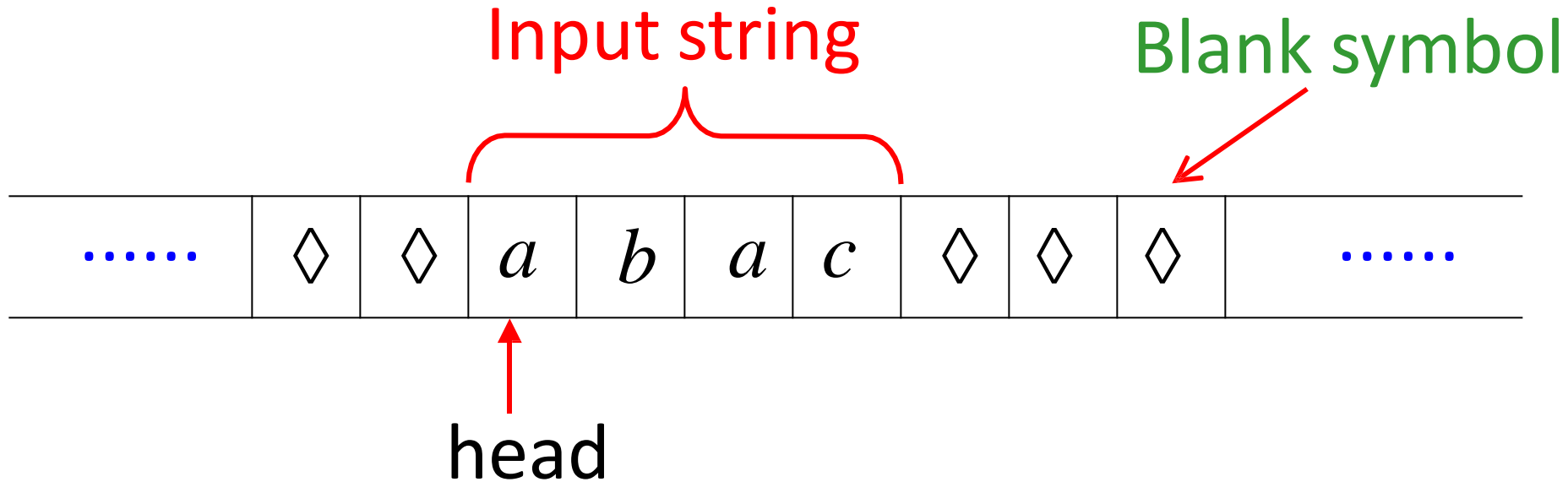


Time 2

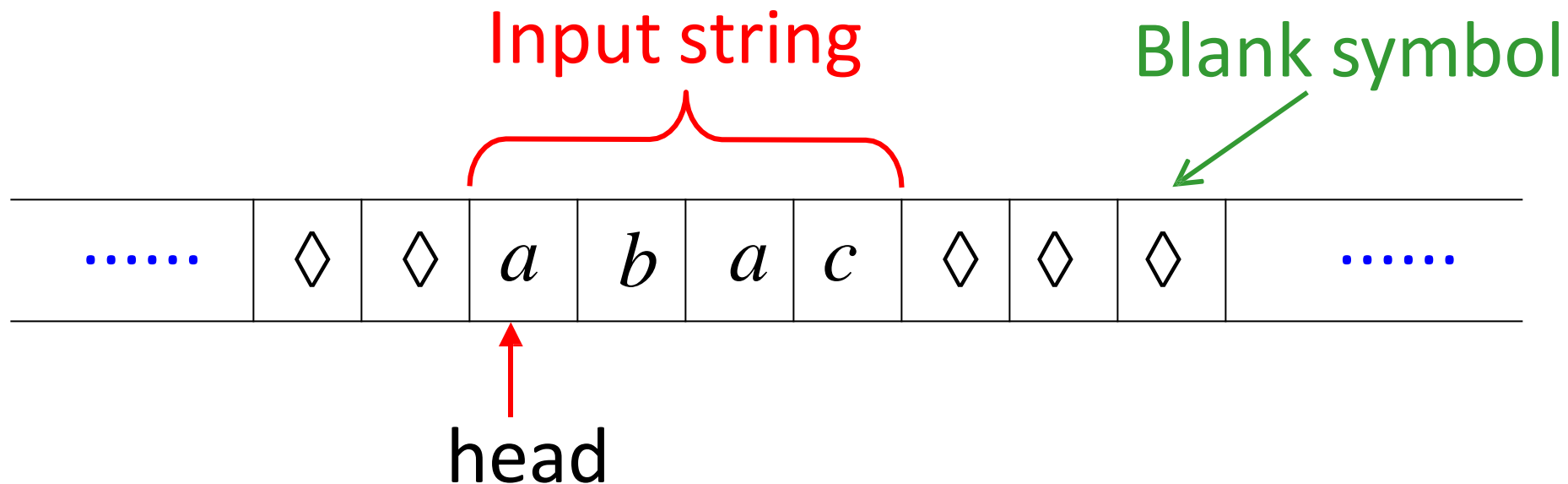


1. Reads b
2. Writes f
3. Moves Right

The Input String



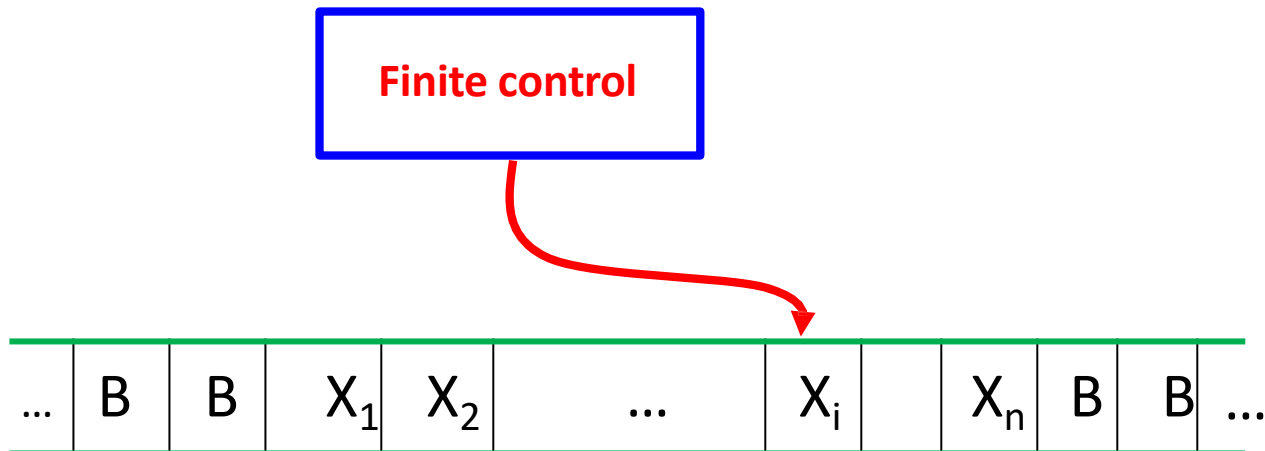
Head starts at the leftmost position of the input string



Remark: the input string is never empty

Picture of a Turing Machine

- The tape consists of cells where each cell holds a symbol from the tape alphabet.
- Initially the *input* consists of a finite-length string of symbols and is placed on the tape.
- To the left of the input and to the right of the input, extending to infinity, are placed *blanks*.
- The *tape head* is initially positioned at the leftmost cell holding the input.

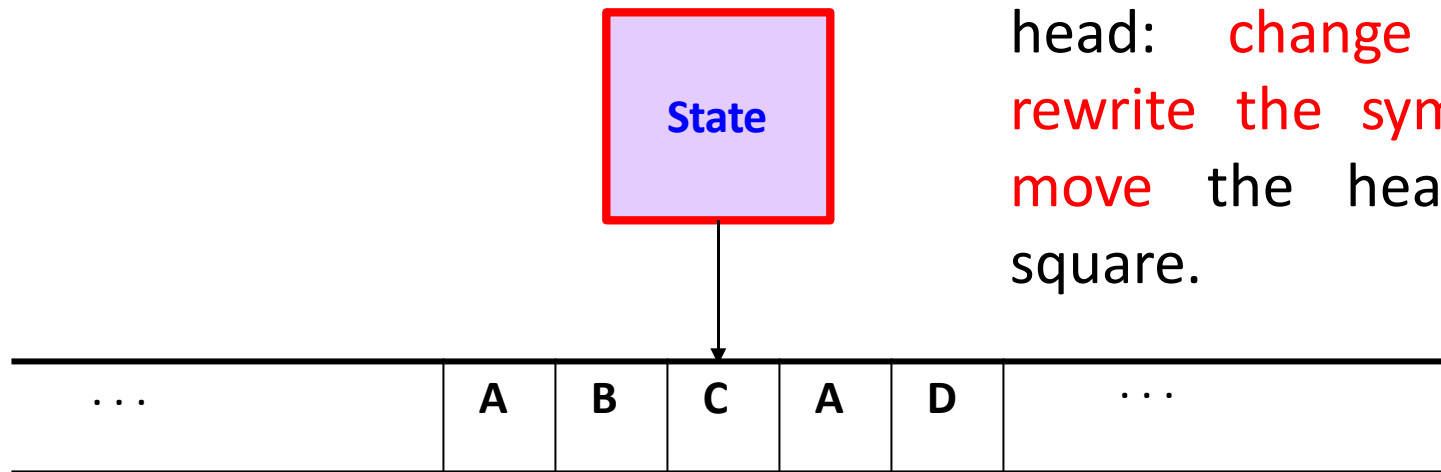


Picture of a Turing Machine

- The TM is said to be scanning that cell. Initially the tape head is at the leftmost cell that holds the input
- **In one move the TM will:**
 - ❖ **Change state**, which may be the same as the current state
 - ❖ **Write a tape symbol in the current cell**, which may be the same as the current symbol
 - ❖ **Move the tape head left or right one cell**
 - ❖ The special states for rejecting and accepting take effect immediately

Picture of a Turing Machine

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



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

Turing-Machine Formalism

- A TM is described by 07 tuples:

$$M = (Q, \Sigma, \Gamma, \delta, q_0, B, F)$$

1. A finite set of *states* (Q , typically).
2. An *input alphabet* (Σ , typically).
3. A *tape alphabet* (Γ , typically; contains Σ).
4. A *transition function* (δ , typically).
5. A *start state* (q_0 , in Q , typically).
6. A *blank symbol* (B , in $\Gamma - \Sigma$, typically).
 - ◆ All tape except for the input is blank initially.
7. A set of *final states* ($F \subseteq Q$, typically).

Conventions

- a, b, \dots are input symbols.
- \dots, X, Y, Z are tape symbols.
- \dots, w, x, y, z are strings of input symbols.
- α, β, \dots are strings of tape symbols.

The Transition Function

- Takes two arguments:
 1. A state, in Q .
 2. A tape symbol in Γ .
- $\delta(q, Z)$ is either undefined or a triple of the form (p, Y, D) .
 - p is the next state, in Q .
 - Y is the new tape symbol in Γ , written in the cell being scanned, replacing whatever symbol was there.
 - D is a *direction*, L or R (direction of head moves)

Actions of the PDA

- If $\delta(q, Z) = (p, Y, D)$ then, in state q , scanning Z under its tape head, the TM:
 1. Changes the state to p .
 2. Replaces Z by Y on the tape.
 3. Moves the head one square in direction D .
 - ◆ $D = L$: move left; $D = R$: move right.

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, & halts.
- If it reaches a blank, it changes it to a 1 & moves left.

Example: Turing Machine

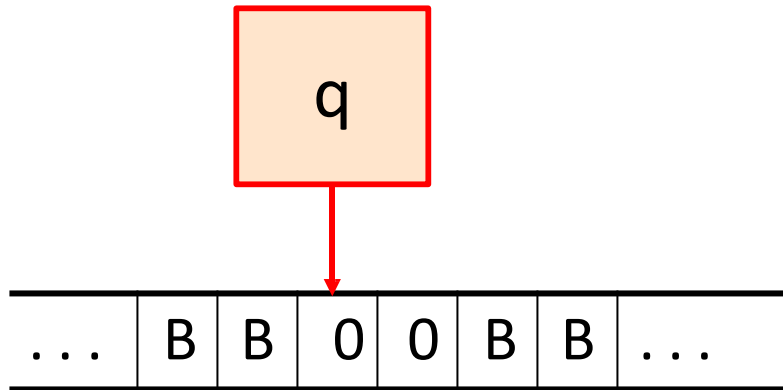
- States = {q (start), f (final)}.
- Input symbols = {0, 1}.
- Tape symbols = {0, 1, B}.
- $\delta(q, 0) = (q, 0, R)$.
- $\delta(q, 1) = (f, 0, R)$.
- $\delta(q, B) = (q, 1, L)$.

Simulation of TM

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

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

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

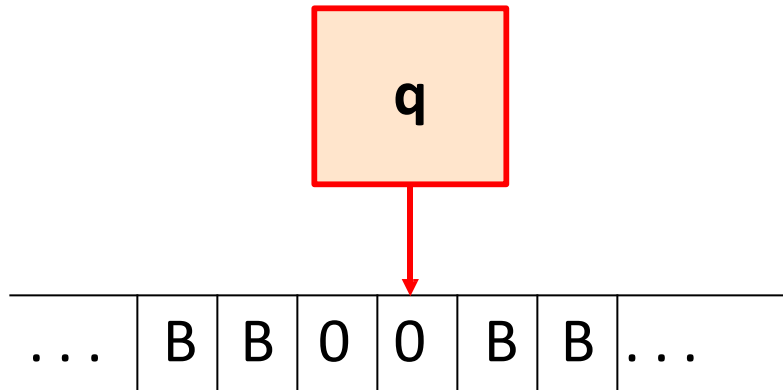


Simulation of TM

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

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

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

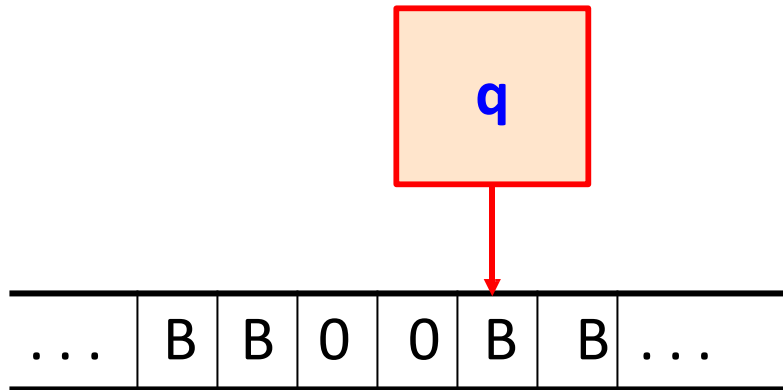


Simulation of TM

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

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

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

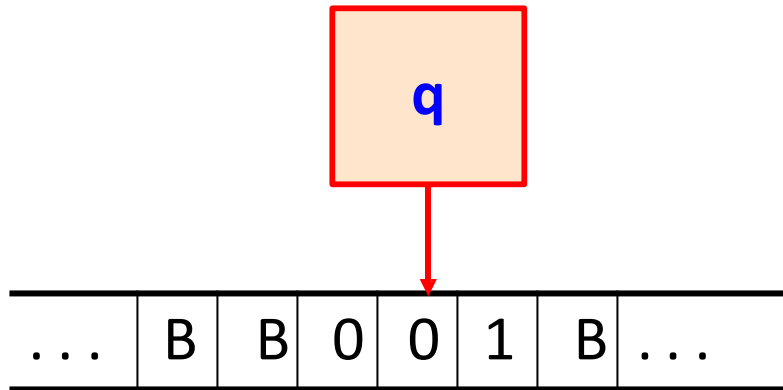


Simulation of TM

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

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

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

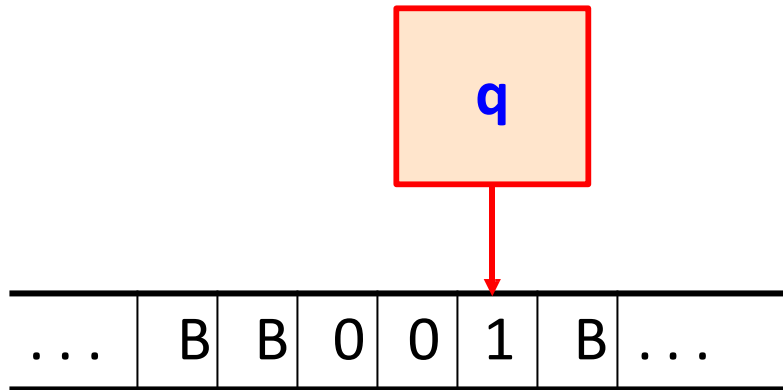


Simulation of TM

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

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

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

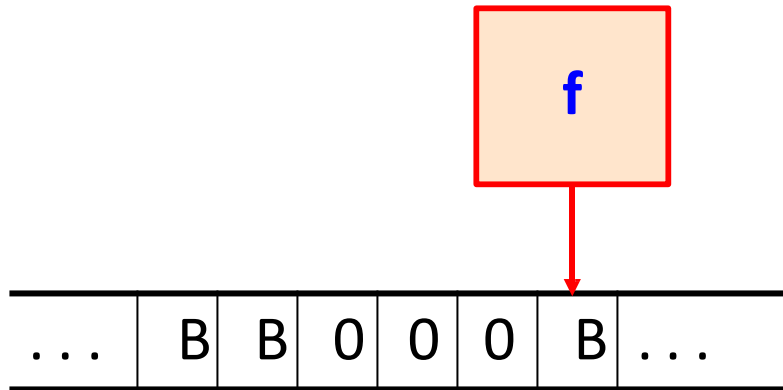


Simulation of TM

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

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

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



No move is possible.
The TM halts and
accepts.

Instantaneous Descriptions of a Turing Machine

- Initially, a TM has a tape consisting of a string of input symbols surrounded by an infinity of blanks in both directions.
- The TM is in the start state, and the head is at the leftmost input symbol.

TM ID's

- An ID is a string $\alpha q \beta$, where $\alpha \beta$ is the tape between the leftmost and rightmost nonblanks (inclusive).
- The state q is immediately to the left of the tape symbol scanned.
- If q is at the right end, it is scanning B .
 - If q is scanning a B at the left end, then consecutive B 's at and to the right of q are part of α .

TM ID's

- As for PDA's we may use symbols \vdash and \vdash^* to represent “becomes in one move” & “becomes in zero or more moves,”
- **Example:** The moves of the previous TM are

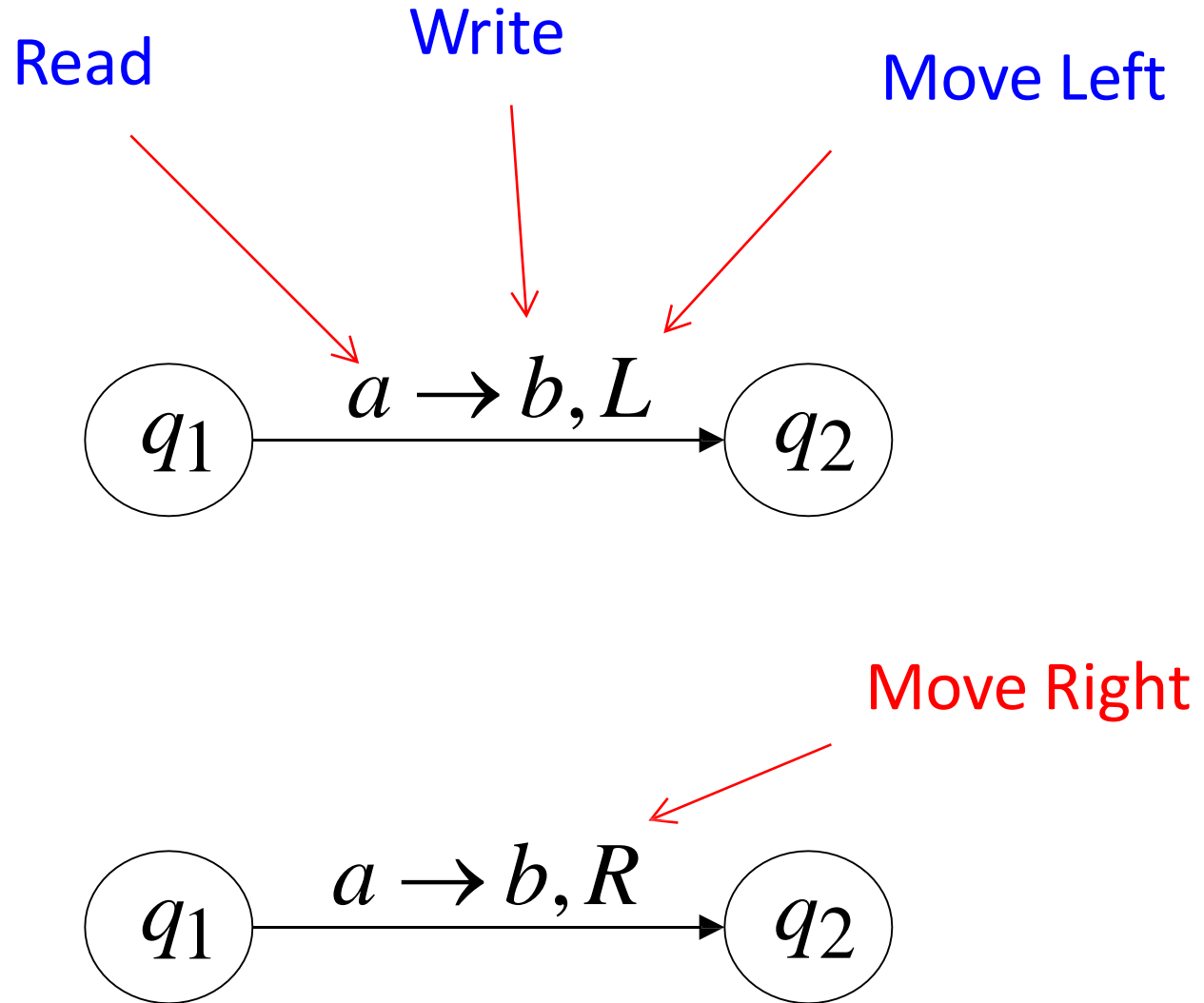
$q00 \vdash 0q0 \vdash 00q \vdash 0q01 \vdash 00q1 \vdash 000f$



Formal Definition of Moves

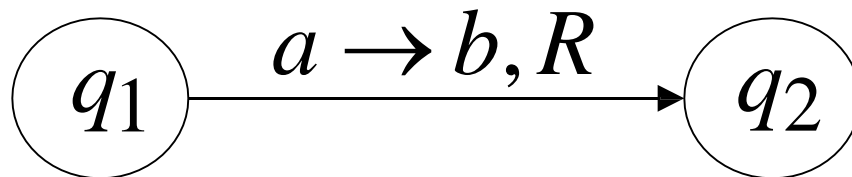
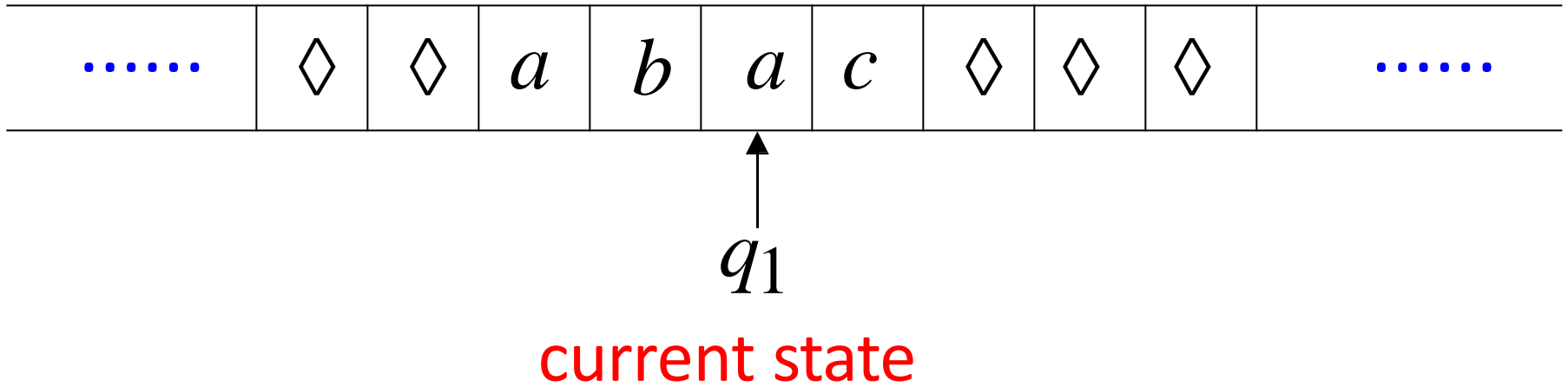
1. If $\delta(q, Z) = (p, Y, R)$, then
 - ◆ $\alpha q Z \beta \vdash \alpha Y p \beta$
 - ◆ If Z is the blank B , then also $\alpha q \vdash \alpha Y p$
2. If $\delta(q, Z) = (p, Y, L)$, then
 - ◆ For any X , $\alpha X q Z \beta \vdash \alpha p X Y \beta$
 - ◆ In addition, $q Z \beta \vdash p B Y \beta$

States & Transitions

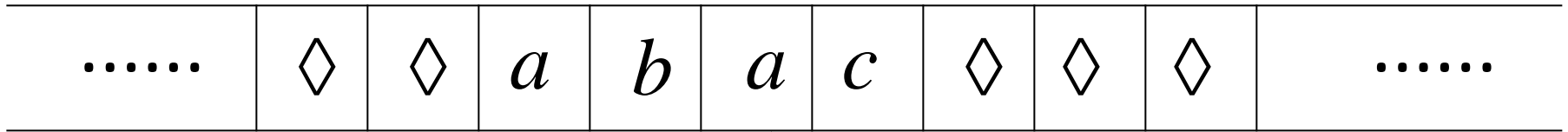


Example:

Time 1

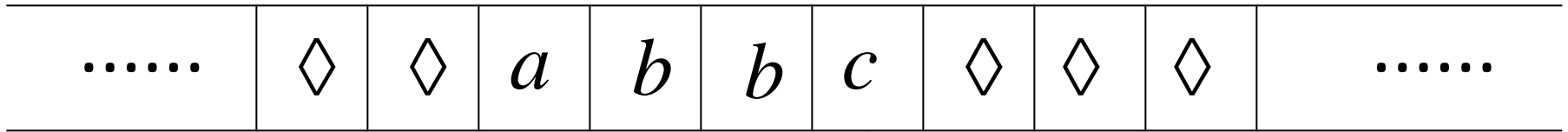


Time 1

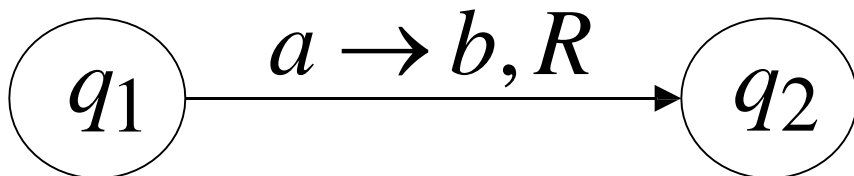


q_1

Time 2

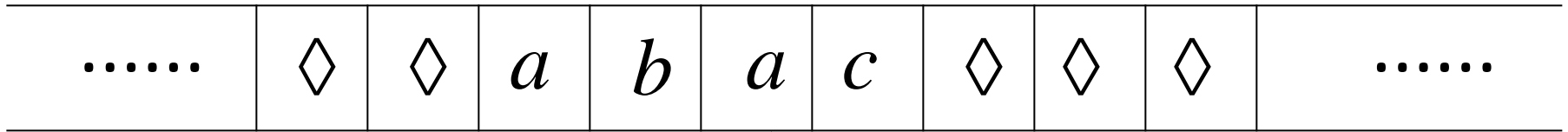


q_2



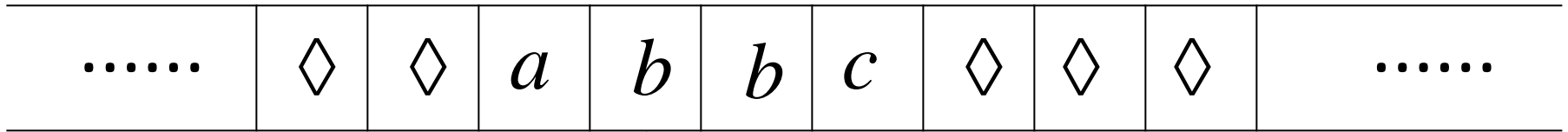
Example:

Time 1

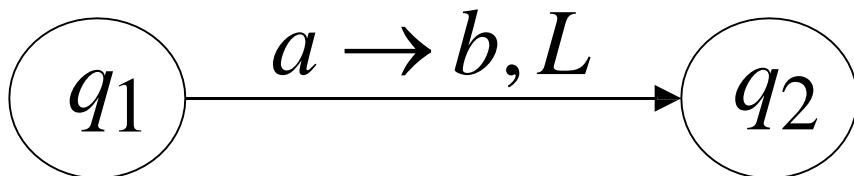


q_1

Time 2

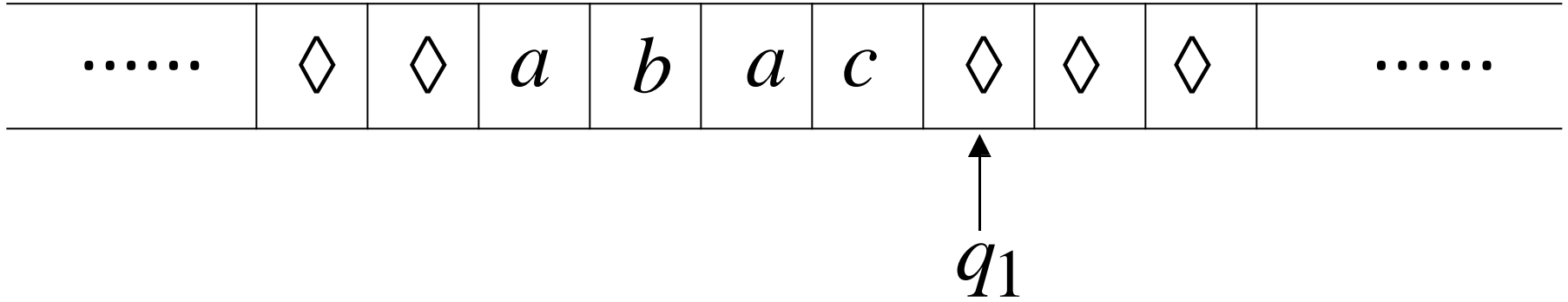


q_2

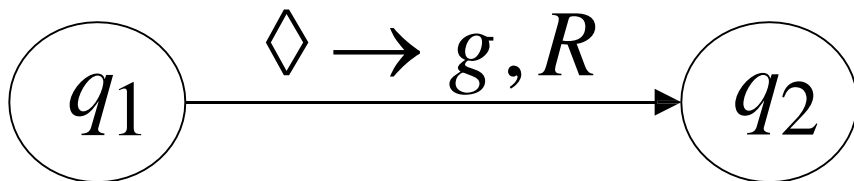
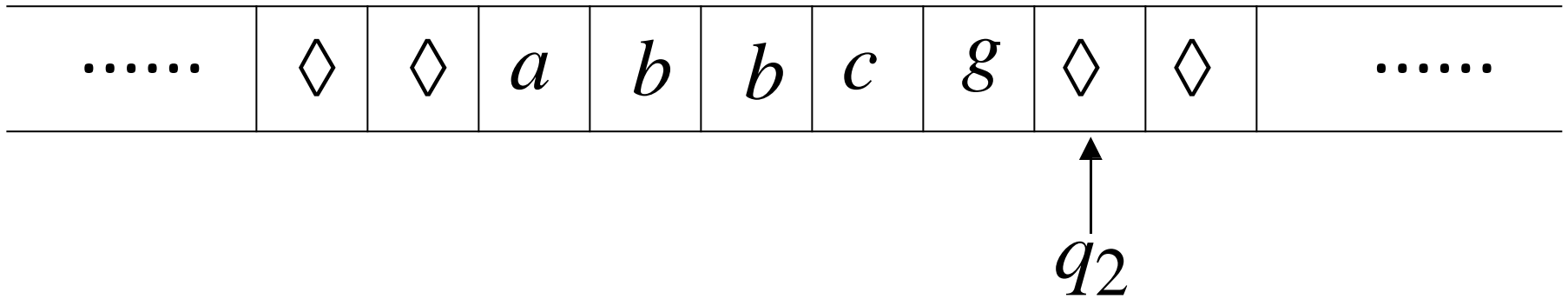


Example:

Time 1



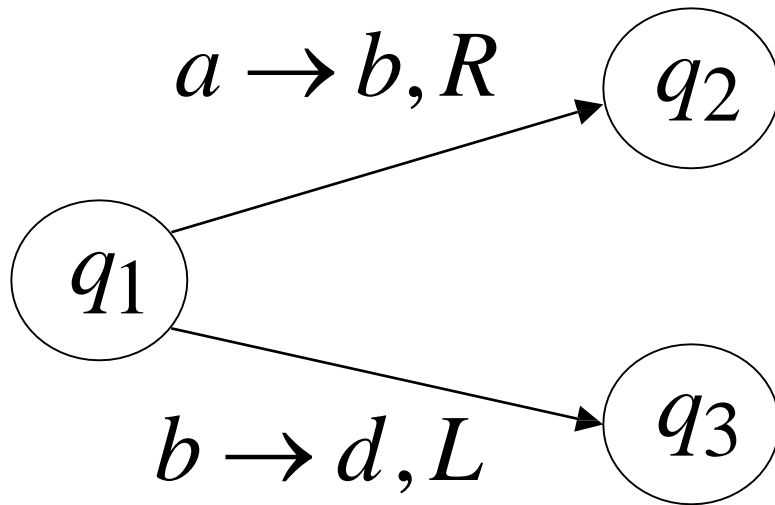
Time 2



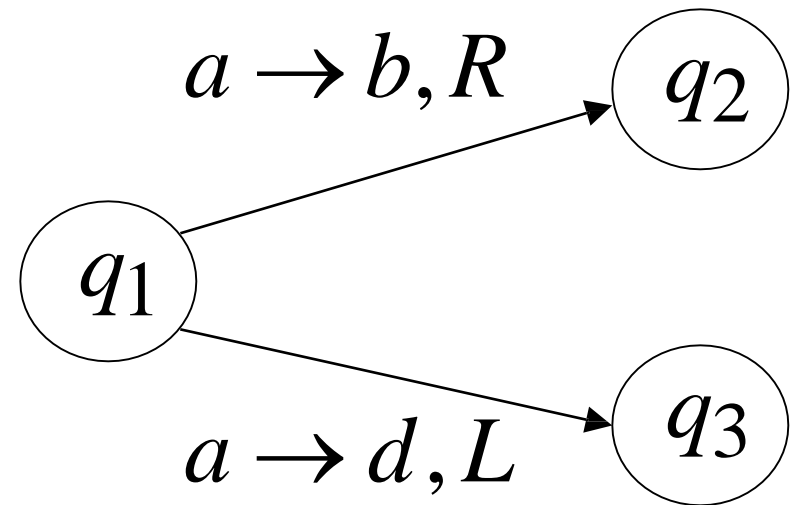
Determinism

Turing Machines are deterministic

Allowed



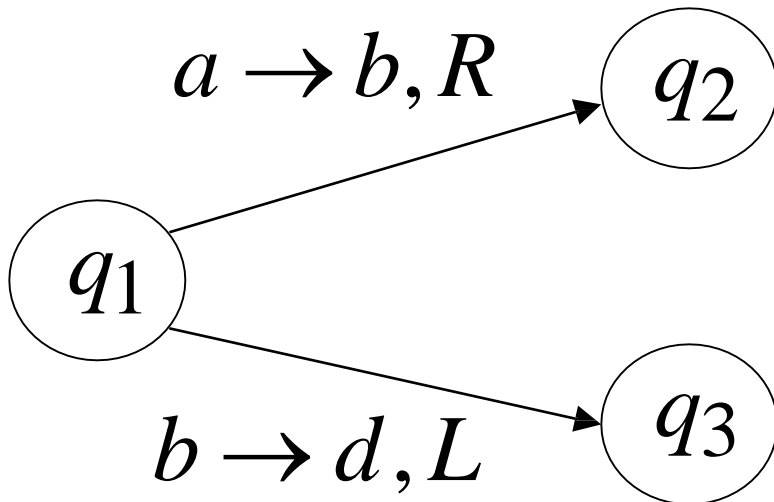
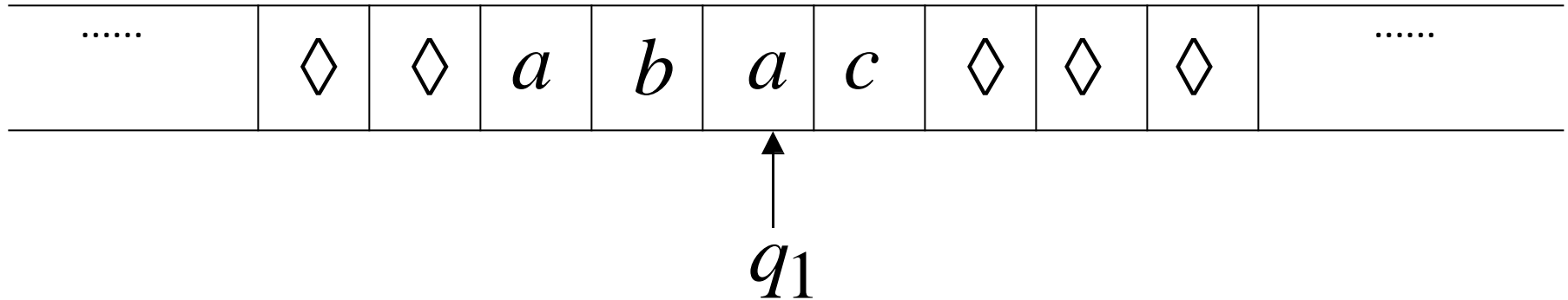
Not Allowed



No lambda transitions allowed

Partial Transition Function

Example:



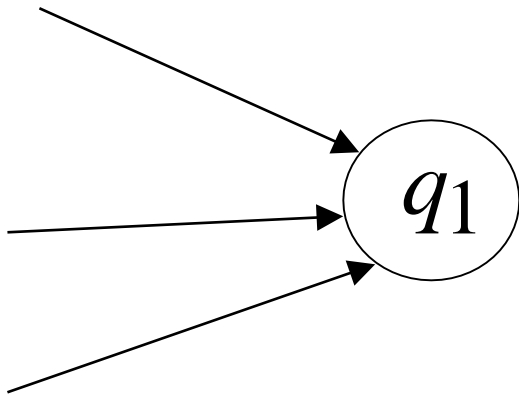
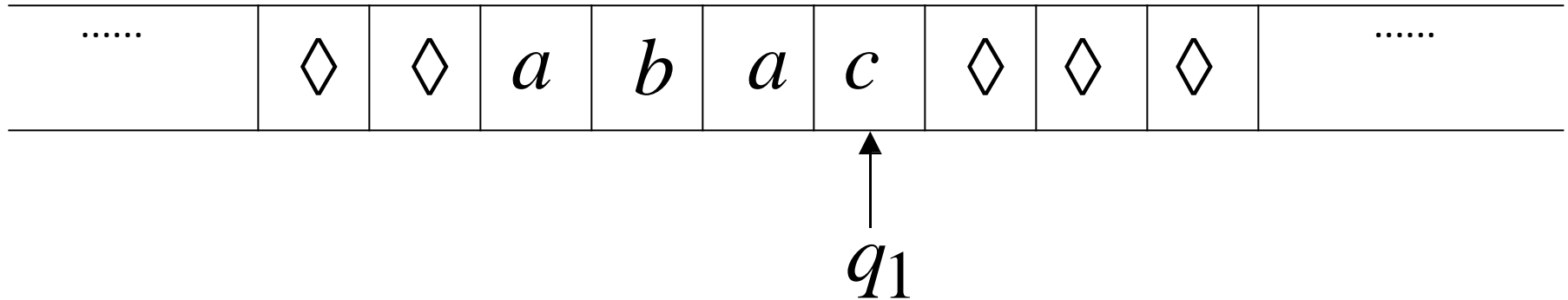
Allowed:

No transition
for input symbol c

Halting

The machine *halts* if there are
no possible transitions to follow

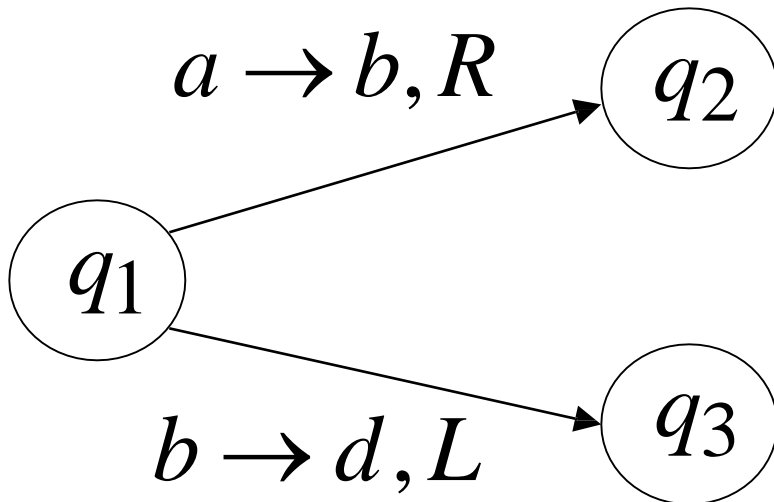
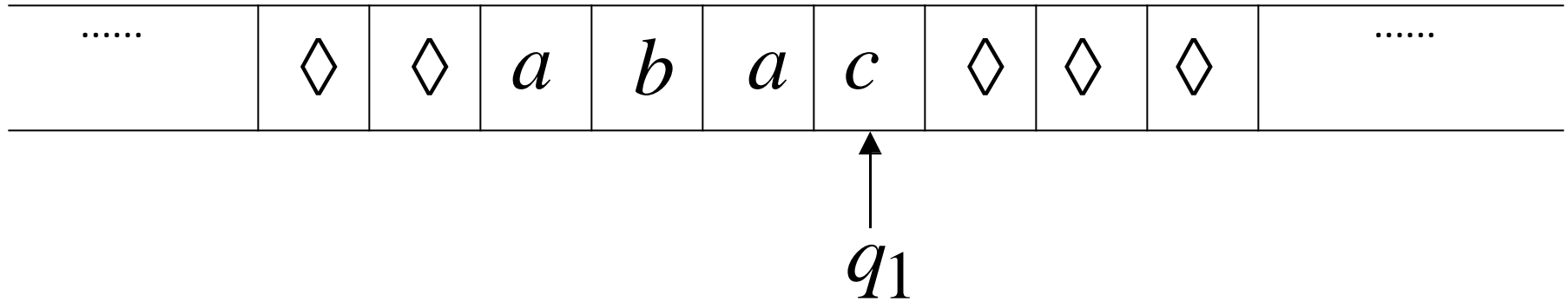
Halting Example 1:



No transition from q_1

HALT!!!

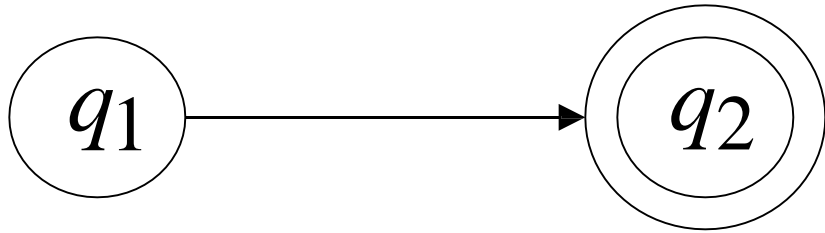
Example:



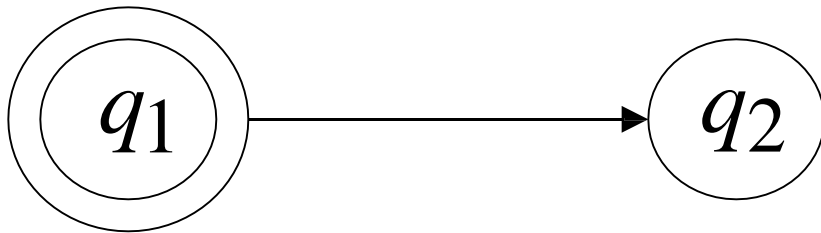
No possible transition

HALT!!!

Final States



Allowed

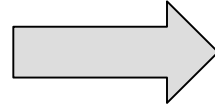


Not Allowed

- Final states have no outgoing transitions
- In a final state the machine halts

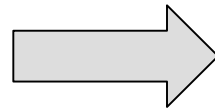
Acceptance

Accept Input



If machine halts
in a final state

Reject Input



If machine halts
in a non-final state

or

If machine enters
an *infinite loop*

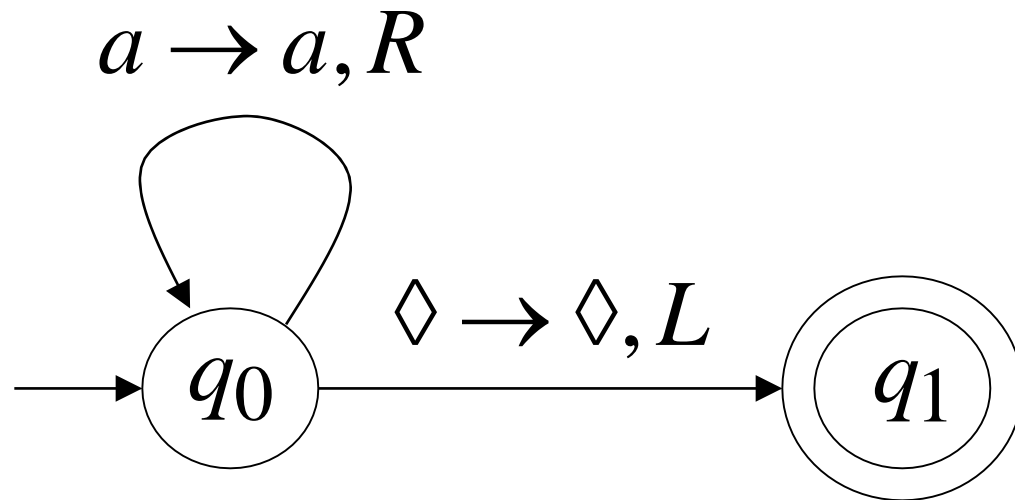
Observation:

In order to accept an input string,
it is not necessary to scan all the
symbols in the string

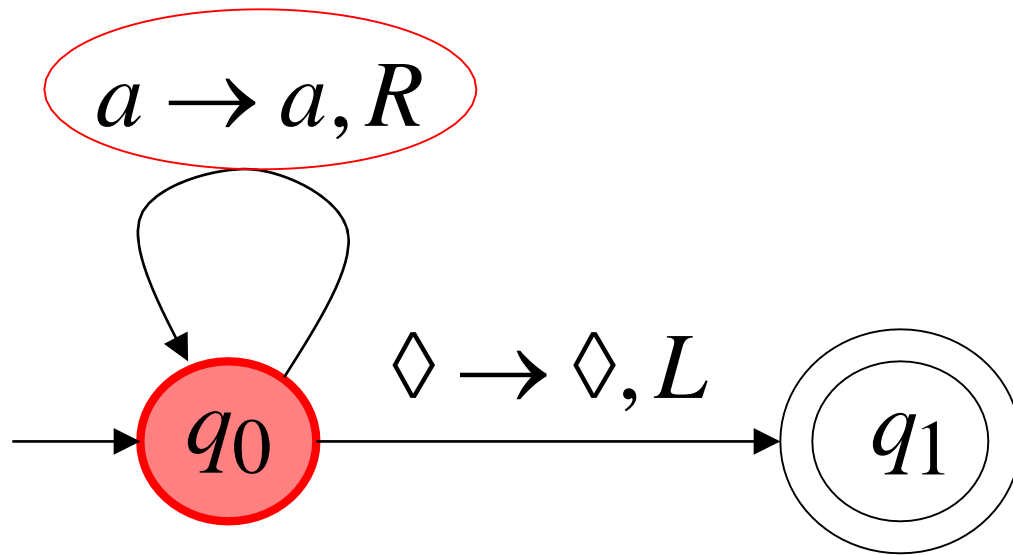
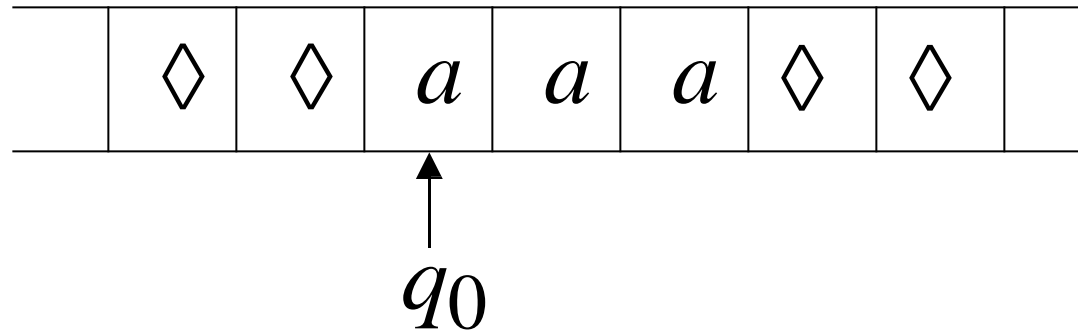
Turing Machine Example

Input alphabet $\Sigma = \{a, b\}$

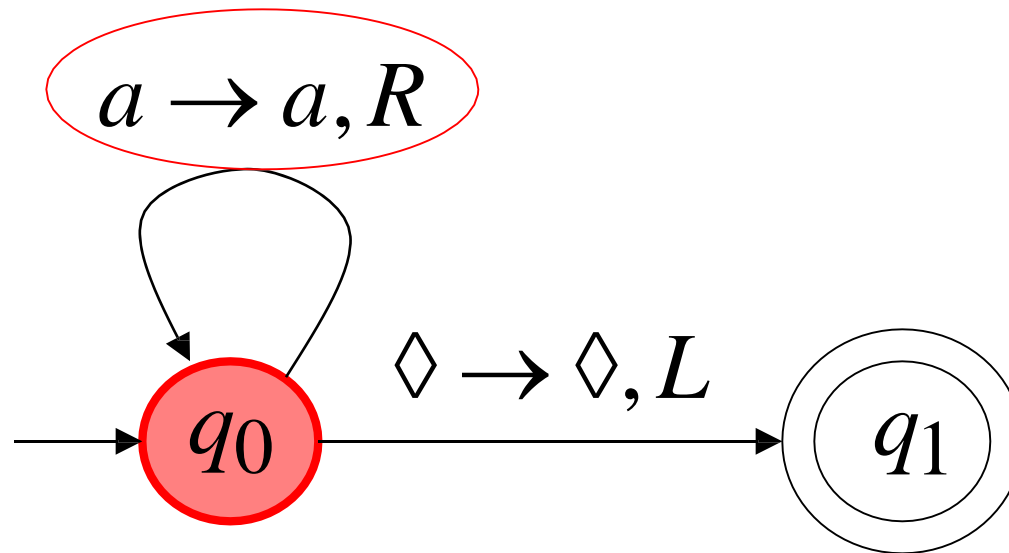
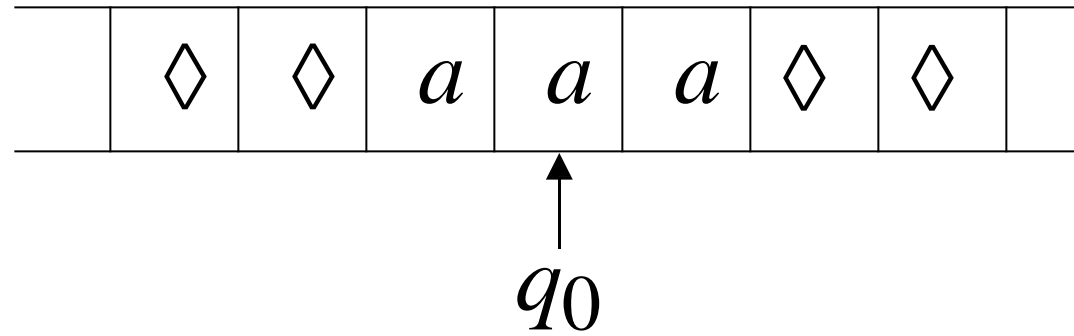
Accepts the language: a^*



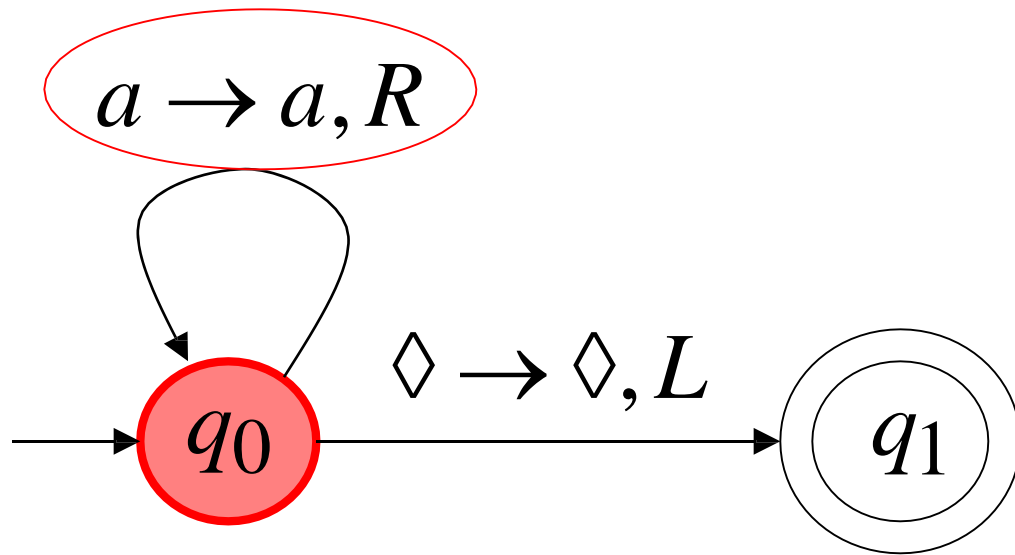
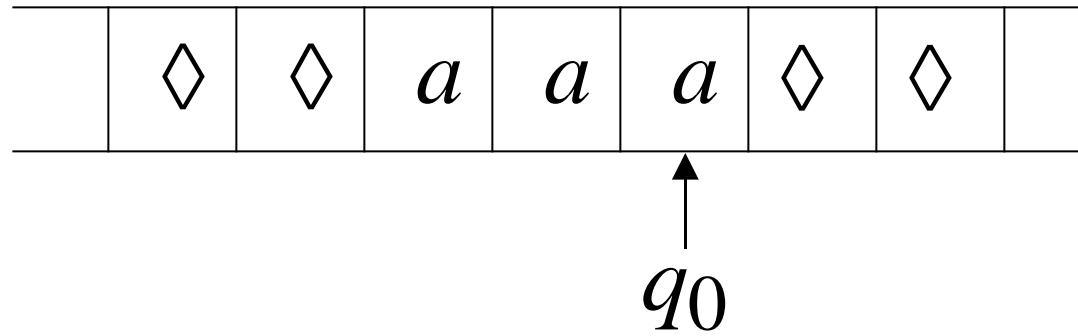
Time 0



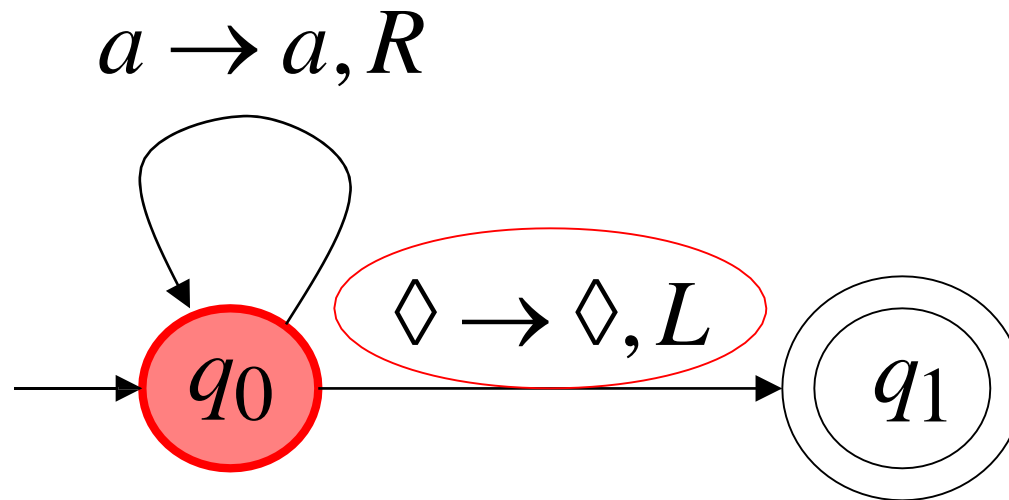
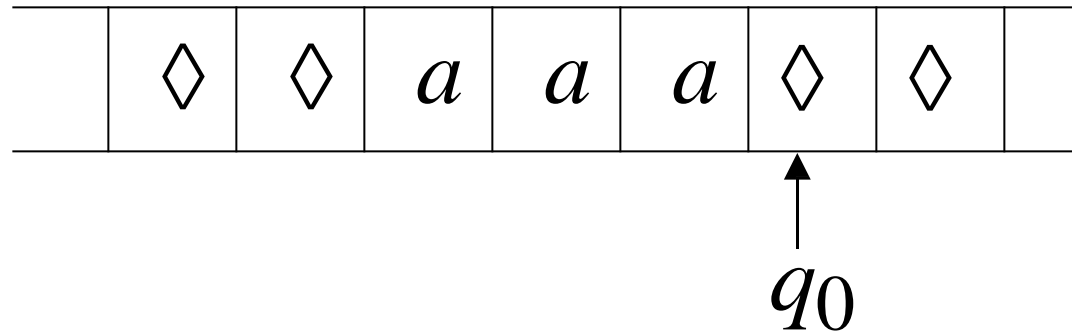
Time 1



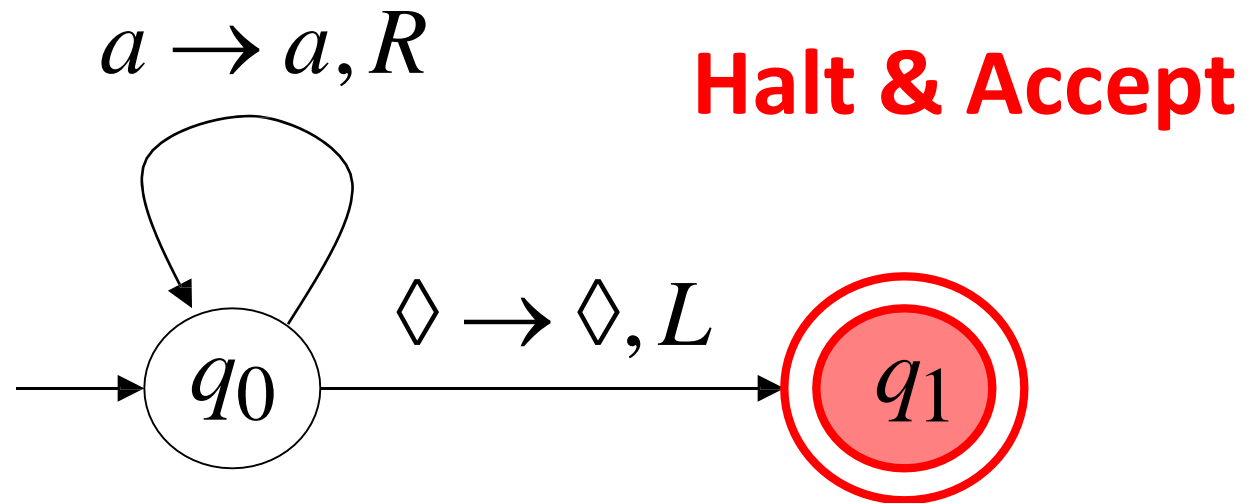
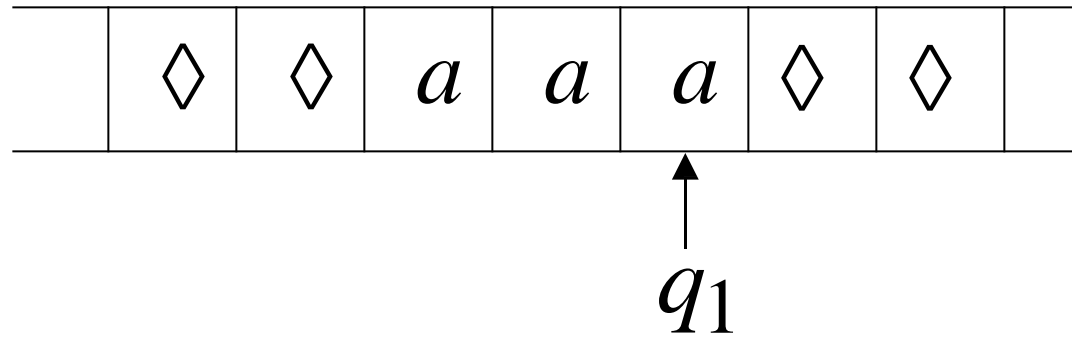
Time 2



Time 3

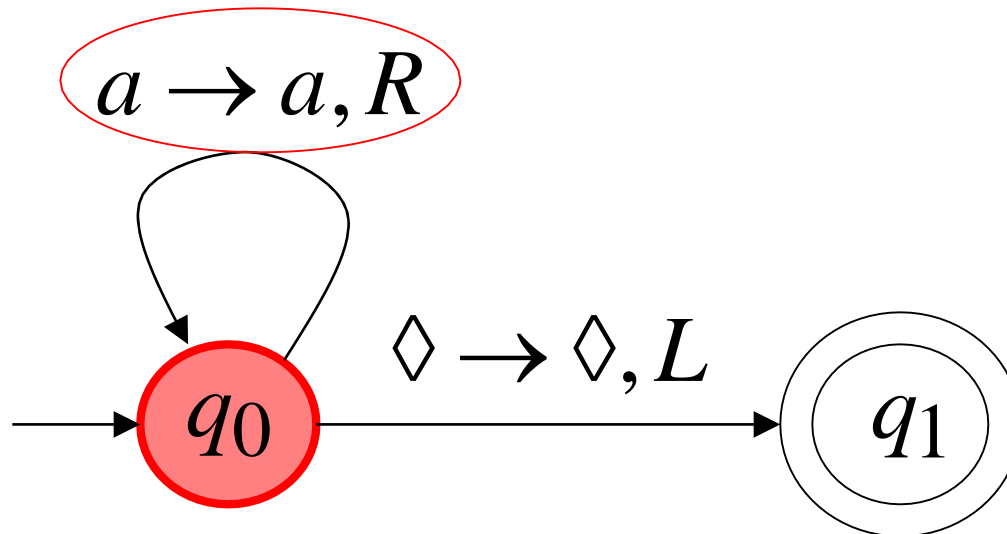
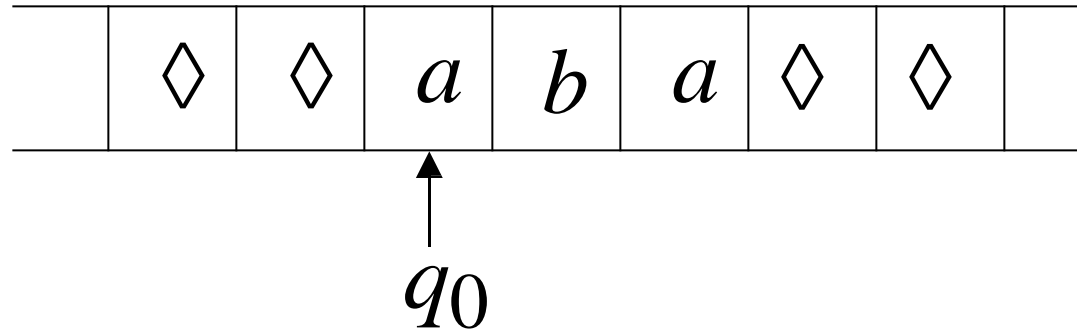


Time 4

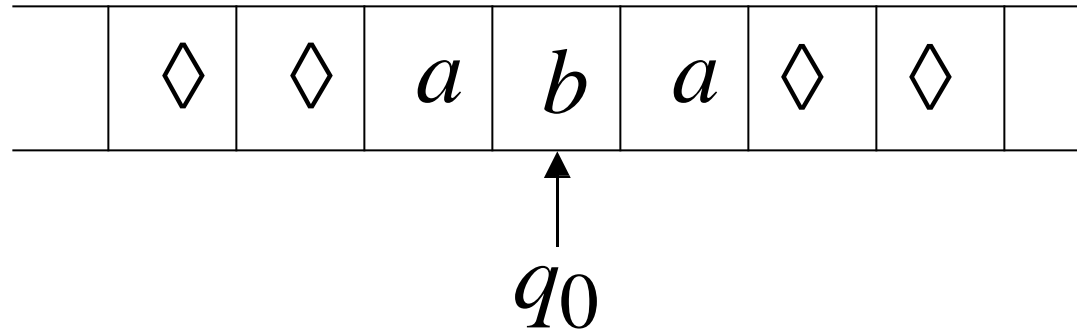


Rejection Example

Time 0

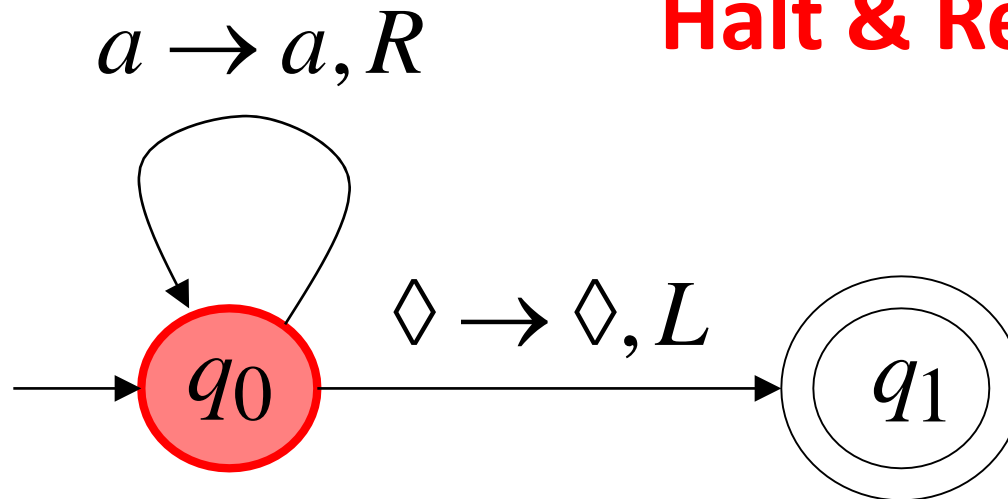


Time 1



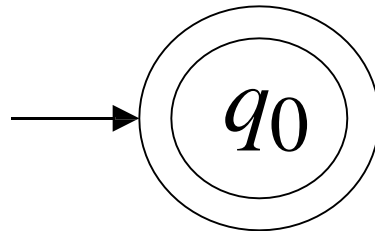
No possible Transition

Halt & Reject

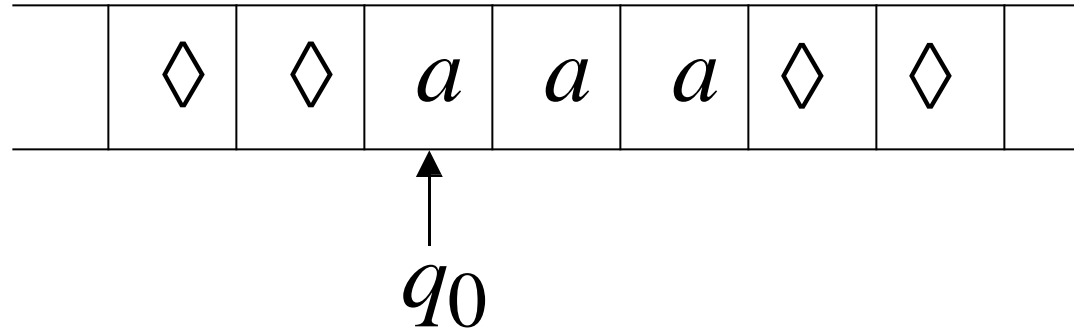


A simpler machine for same language
but for input alphabet $\Sigma = \{a\}$

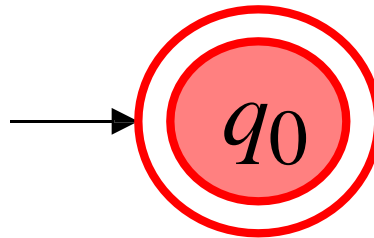
Accepts the language: a^*



Time 0



Halt & Accept



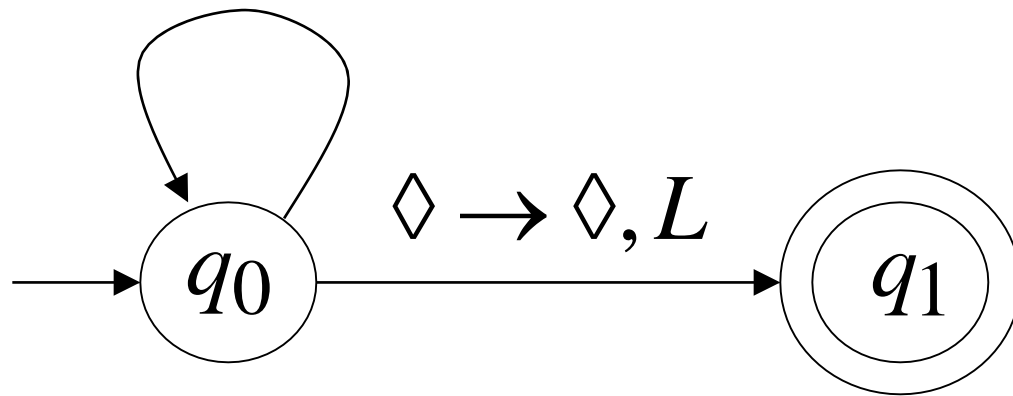
Not necessary to scan input

Infinite Loop Example

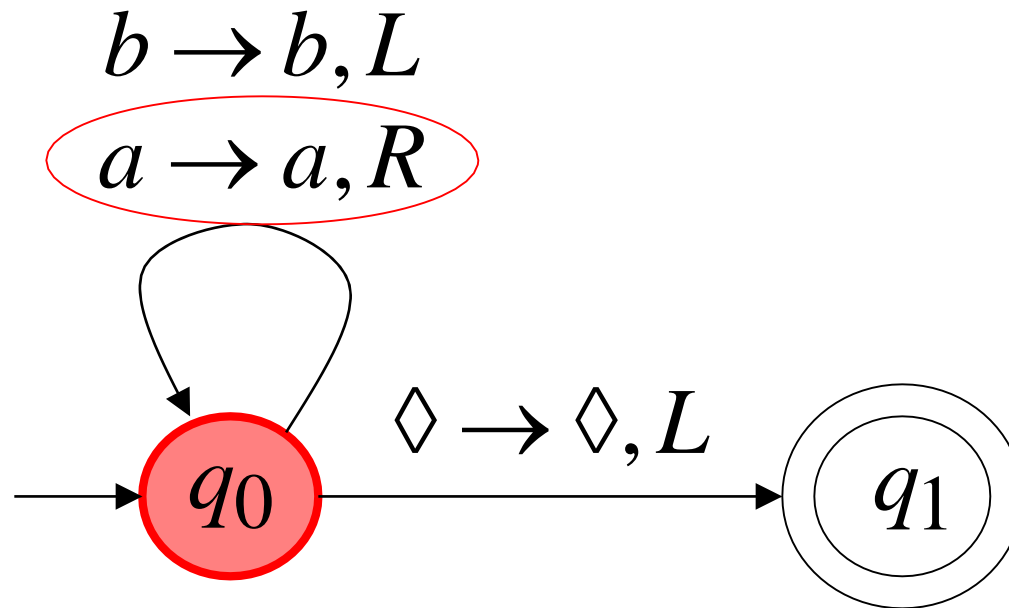
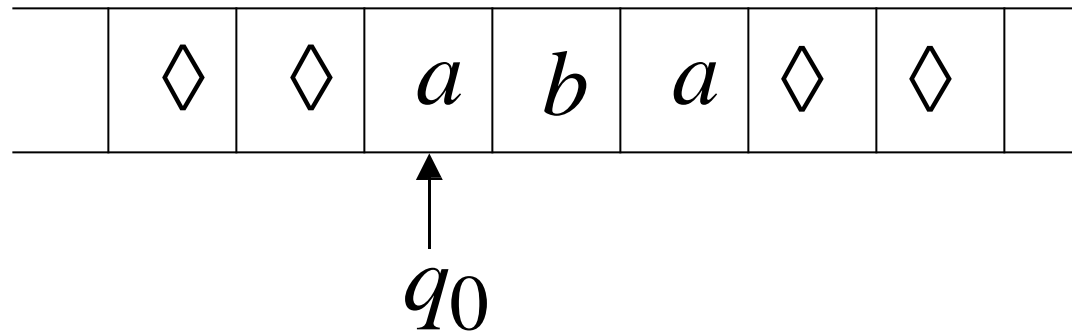
A Turing machine for language $a^* + b(a + b)^*$

$b \rightarrow b, L$

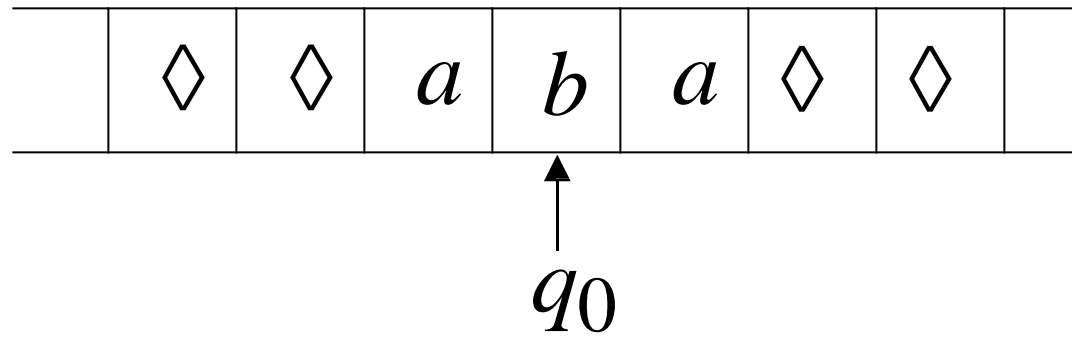
$a \rightarrow a, R$



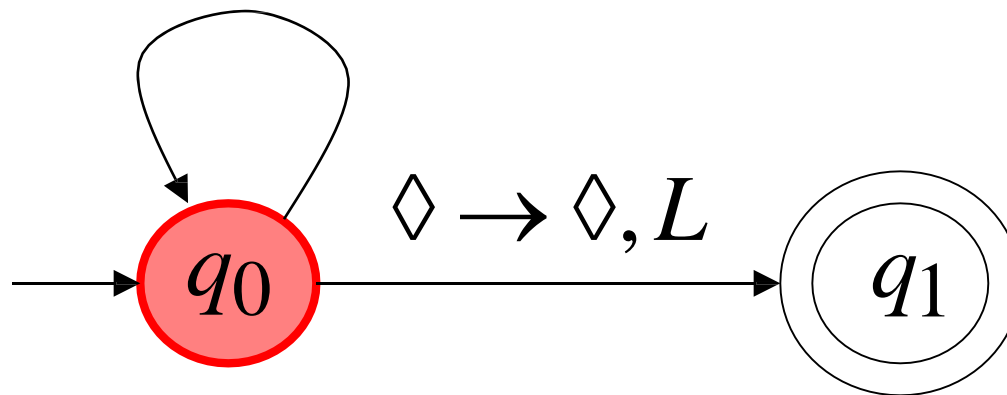
Time 0



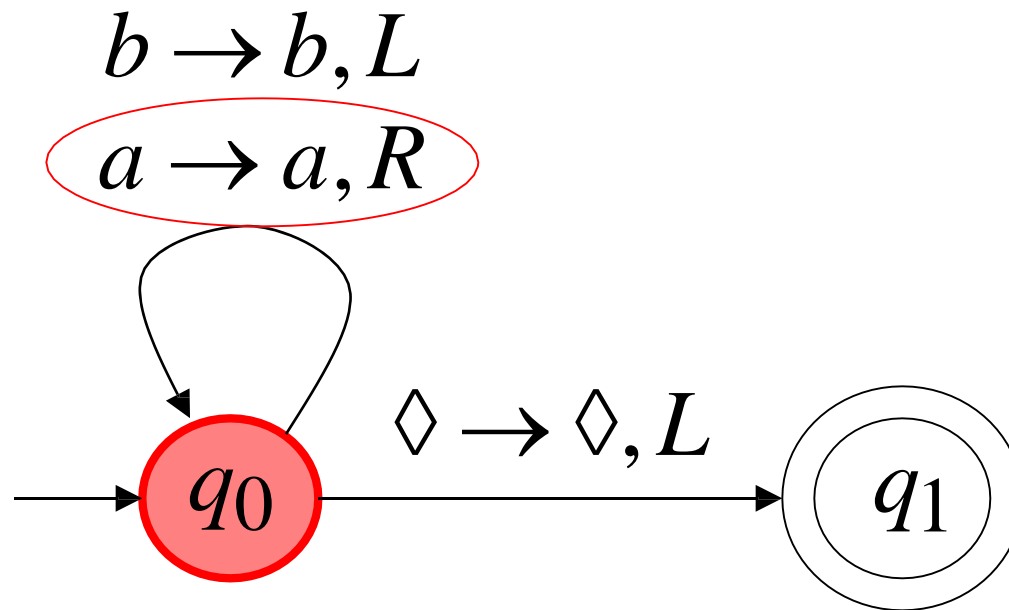
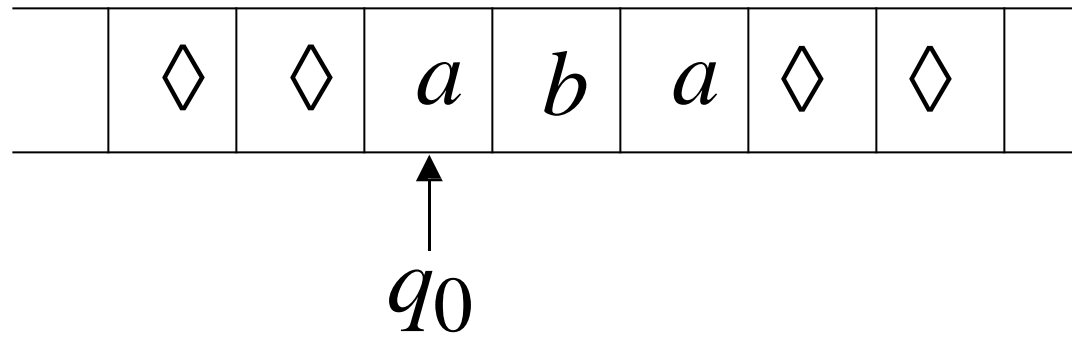
Time 1



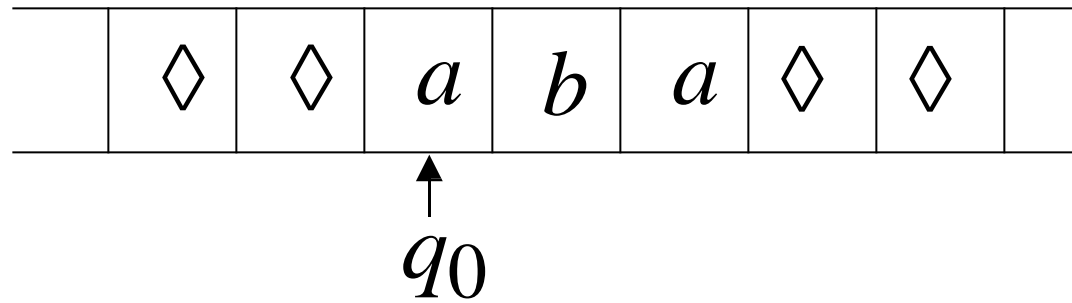
$b \rightarrow b, L$
 $a \rightarrow a, R$



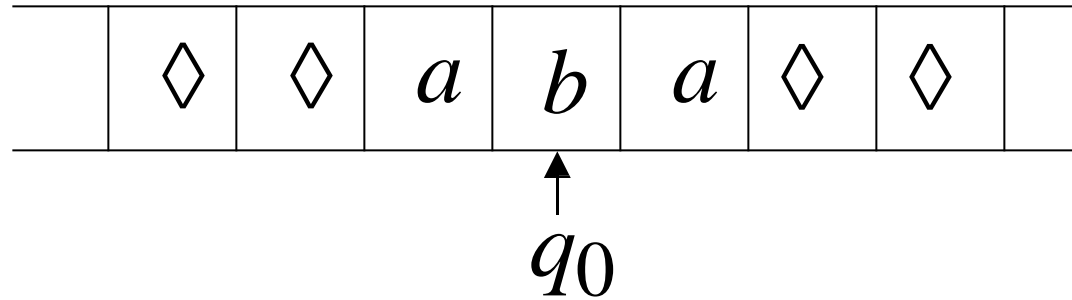
Time 2



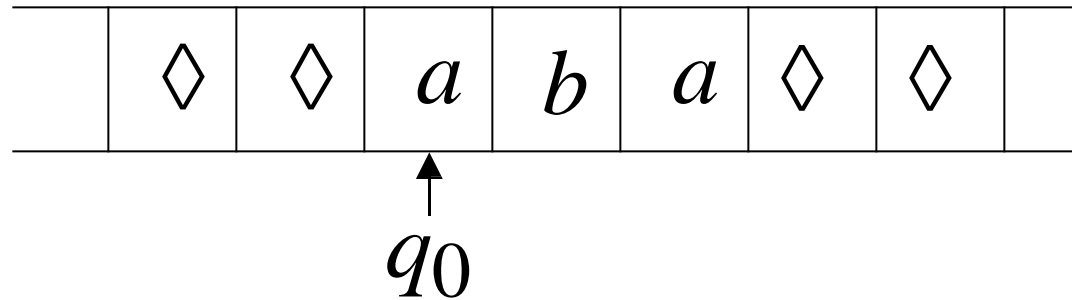
Time 2



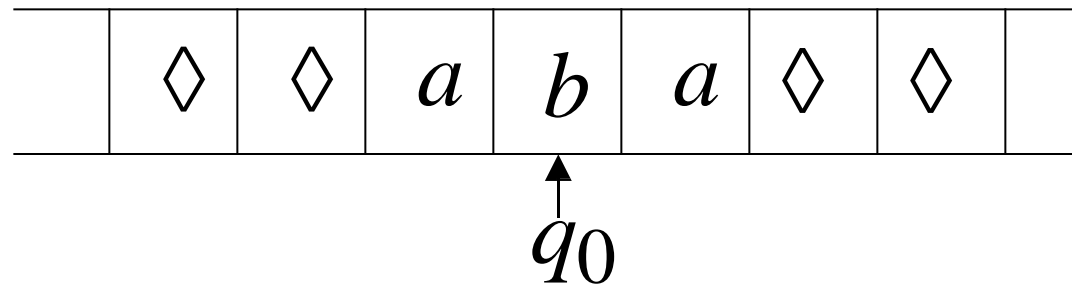
Time 3



Time 4



Time 5



Infinite loop

Because of the **infinite loop**:

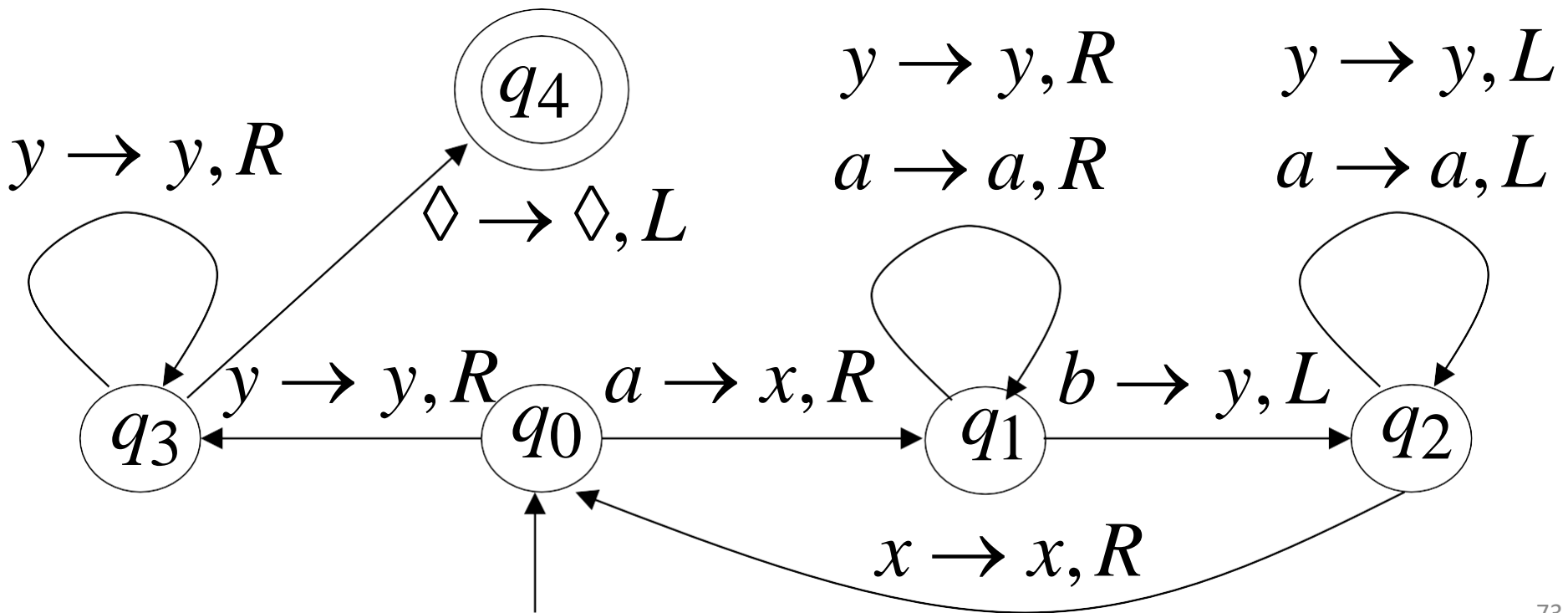
- The accepting state cannot be reached
- The machine never halts
- The input string is **rejected**

Another Turing Machine Example

Turing machine for the language

$\{a^n b^n\}$

$n \geq 1$



Basic Idea:

Match **a**'s with **b**'s:

Repeat:

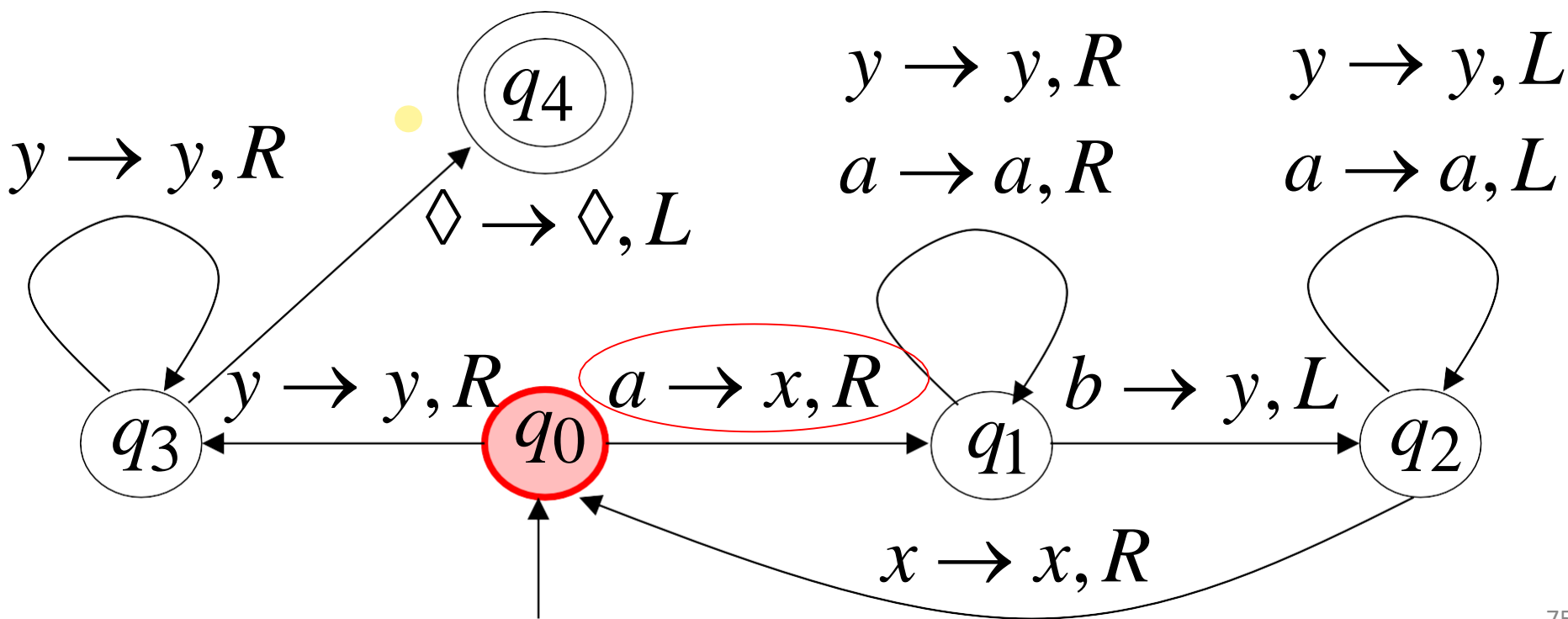
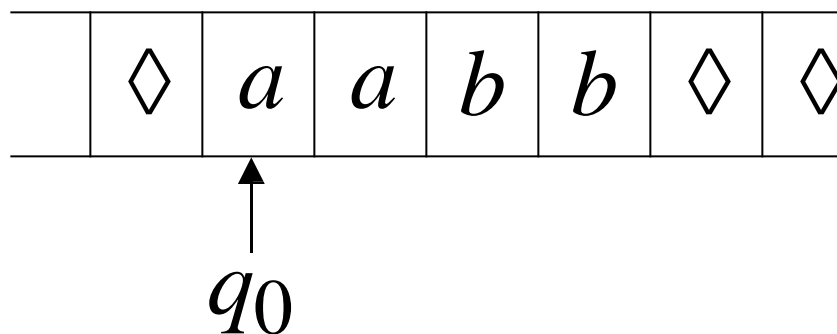
- replace leftmost **a** with **x**

- find leftmost **b** and replace it with **y**

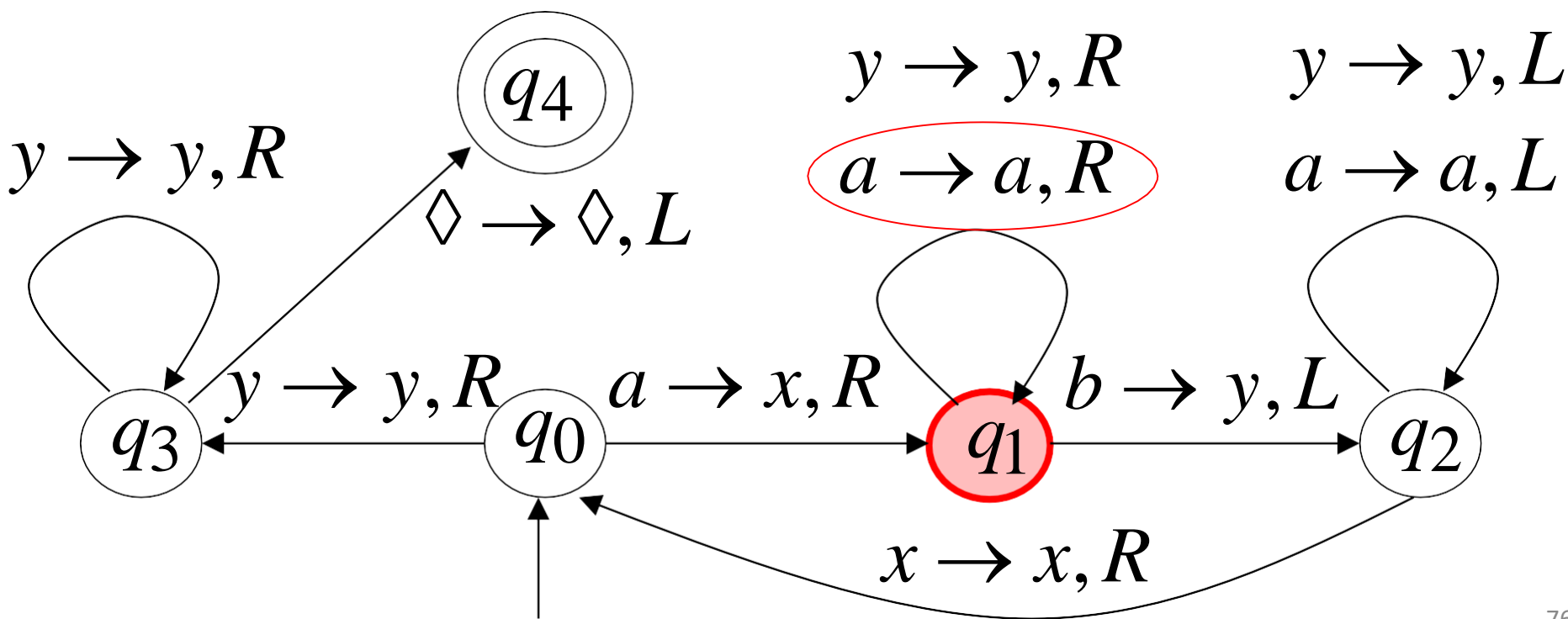
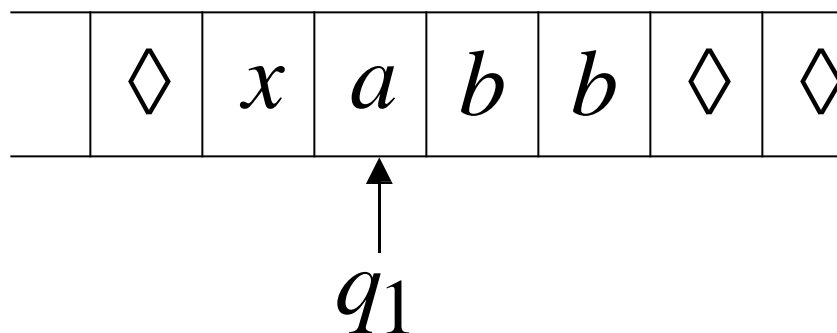
Until there are no more **a**'s or **b**'s

If there is a remaining **a** or **b** reject

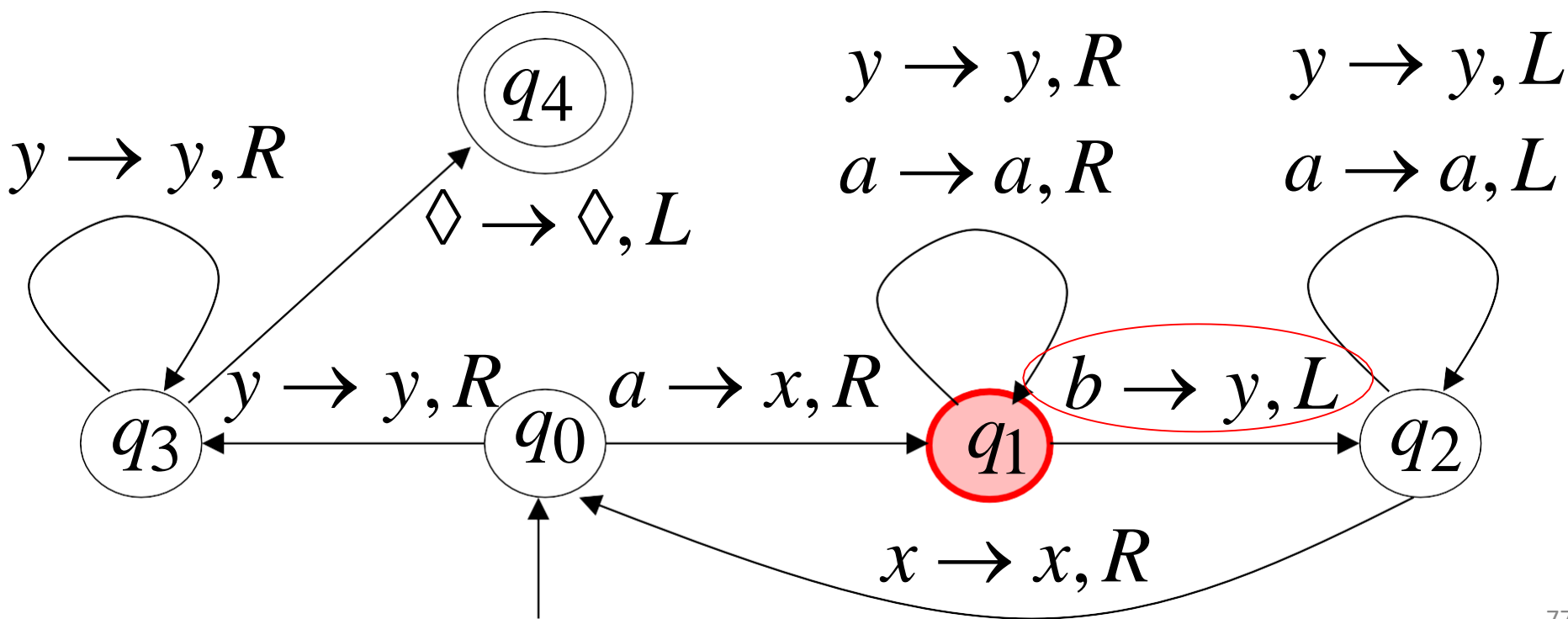
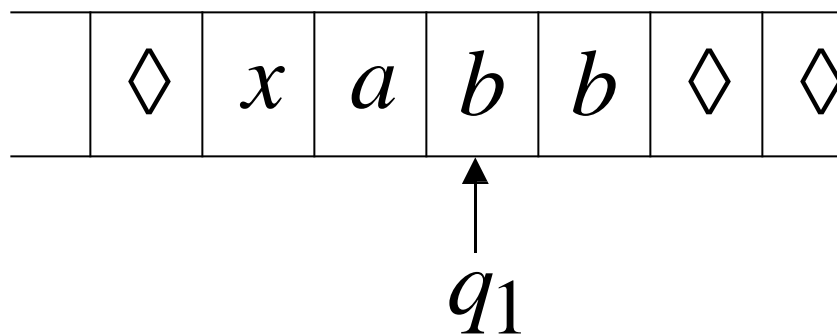
Time 0



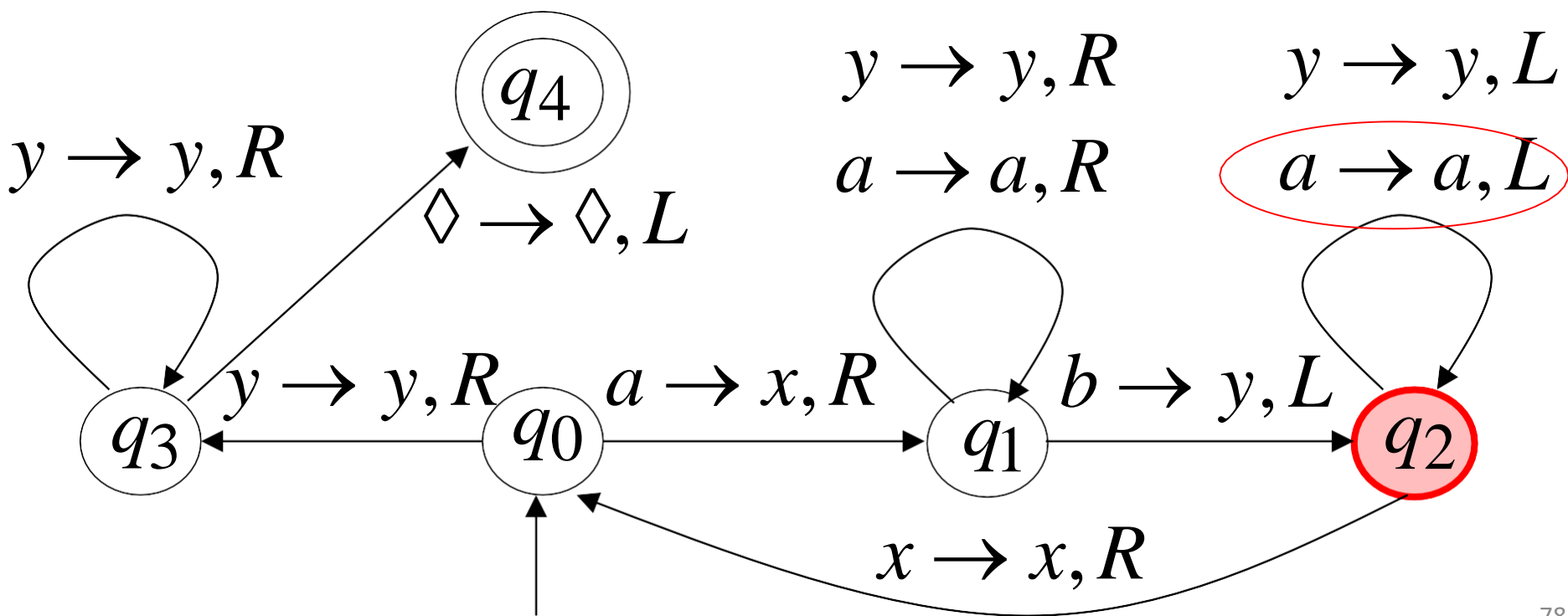
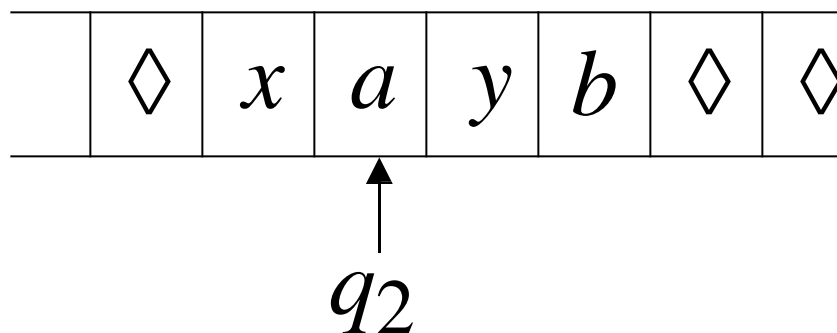
Time 1



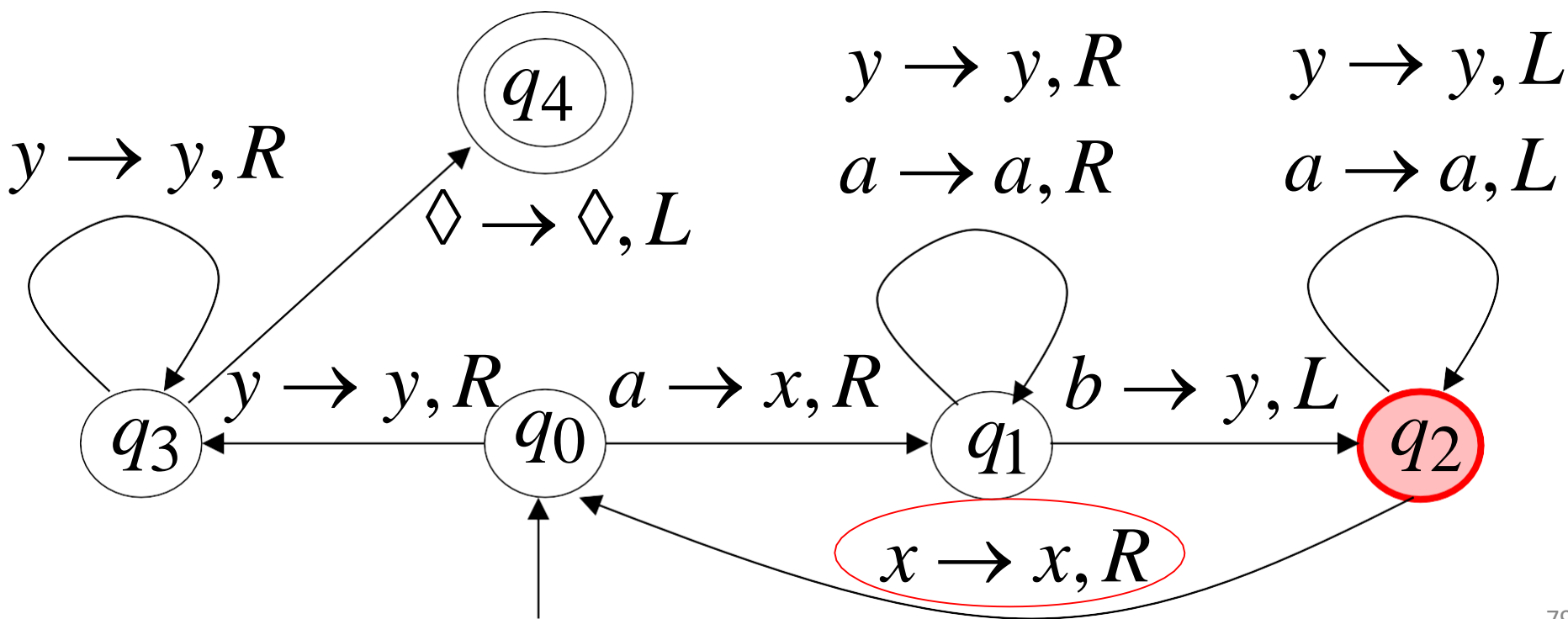
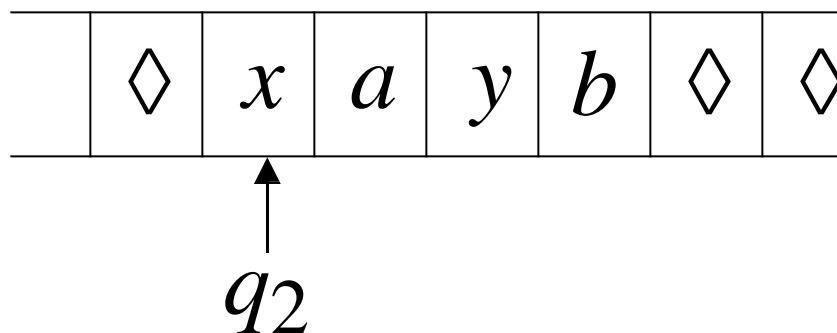
Time 2



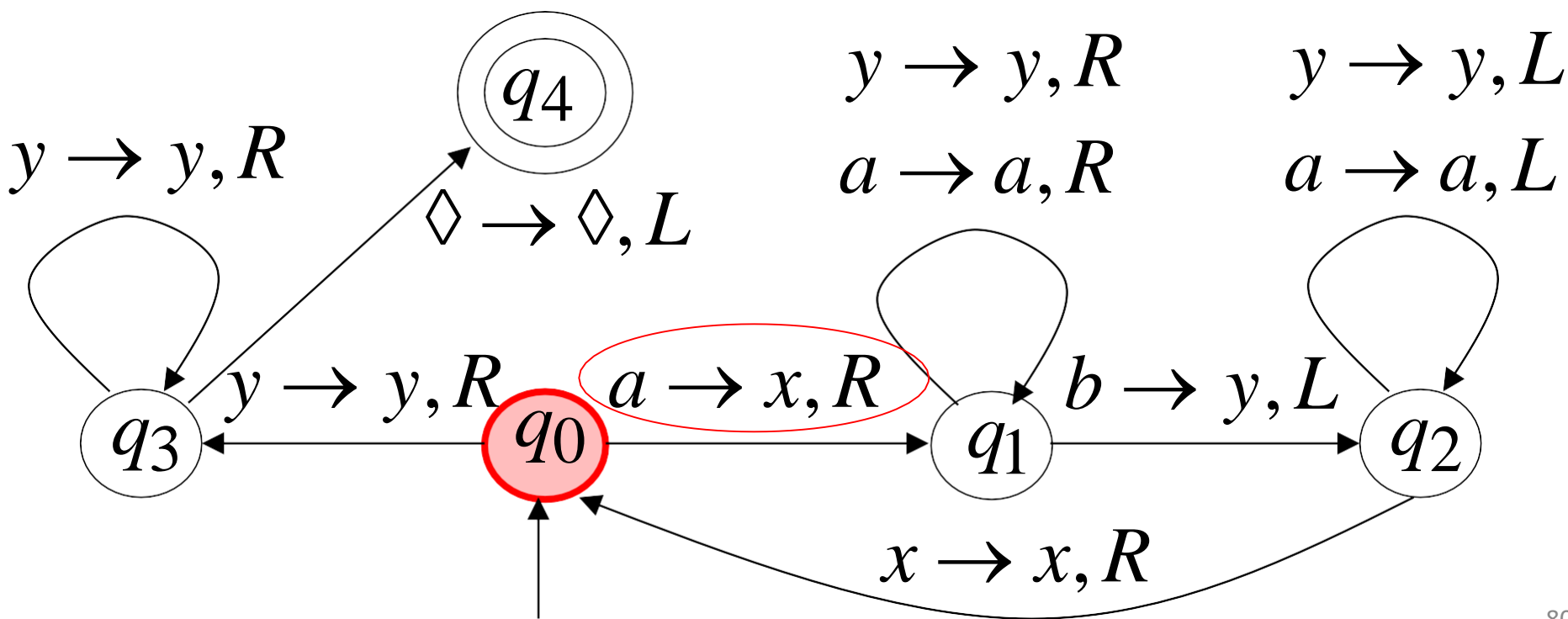
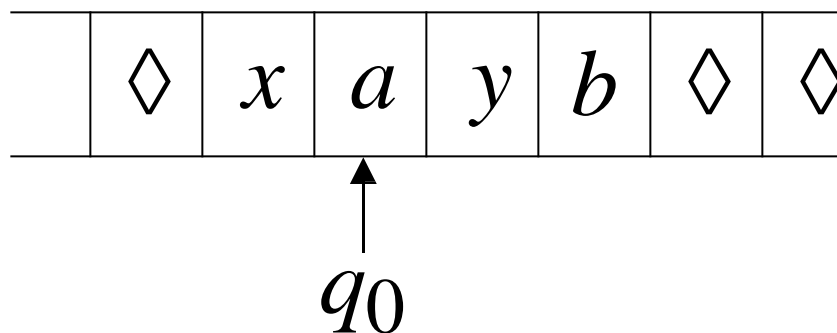
Time 3



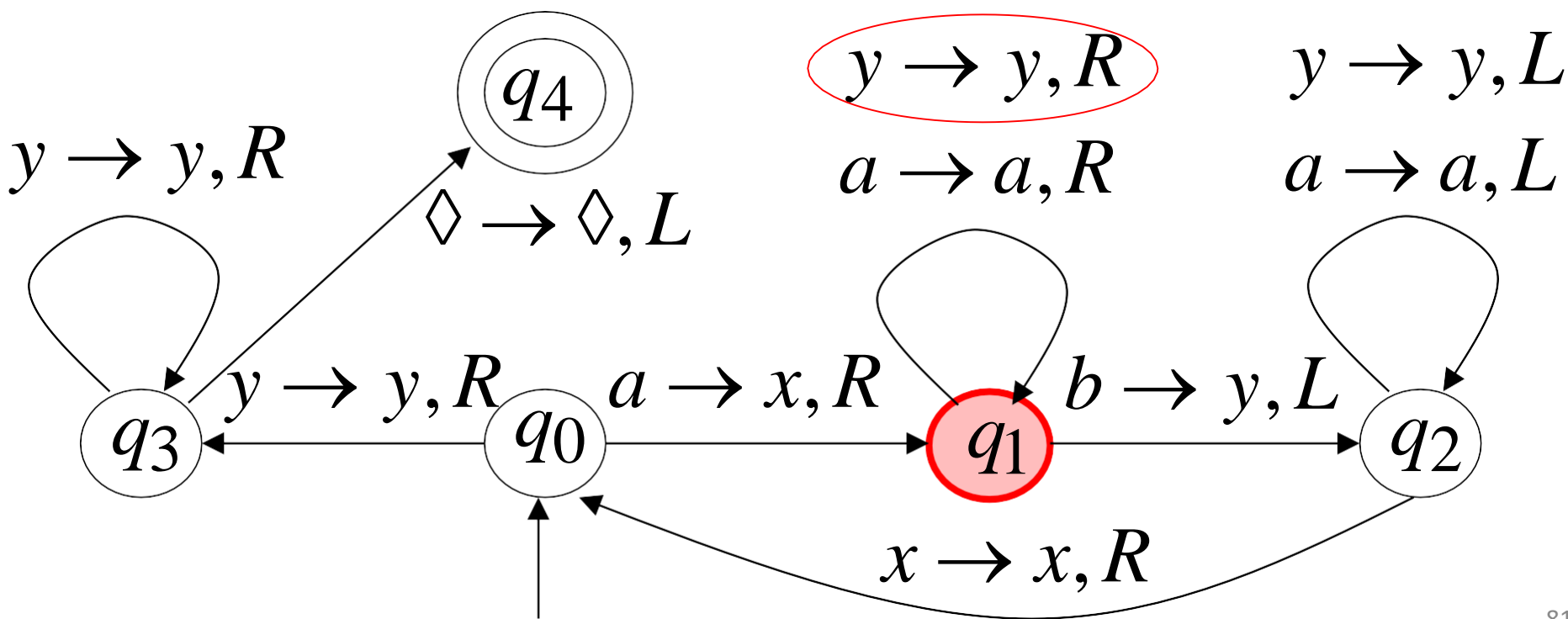
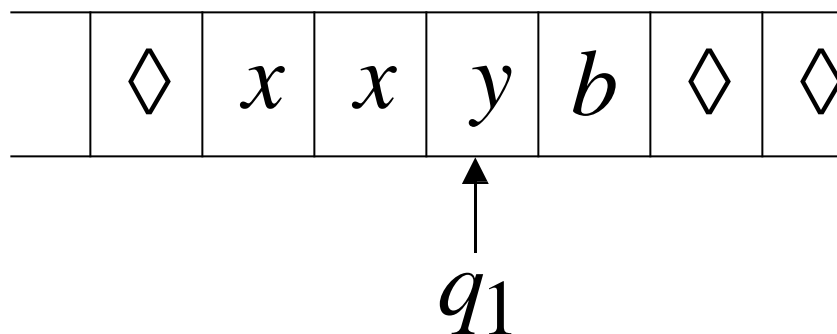
Time 4



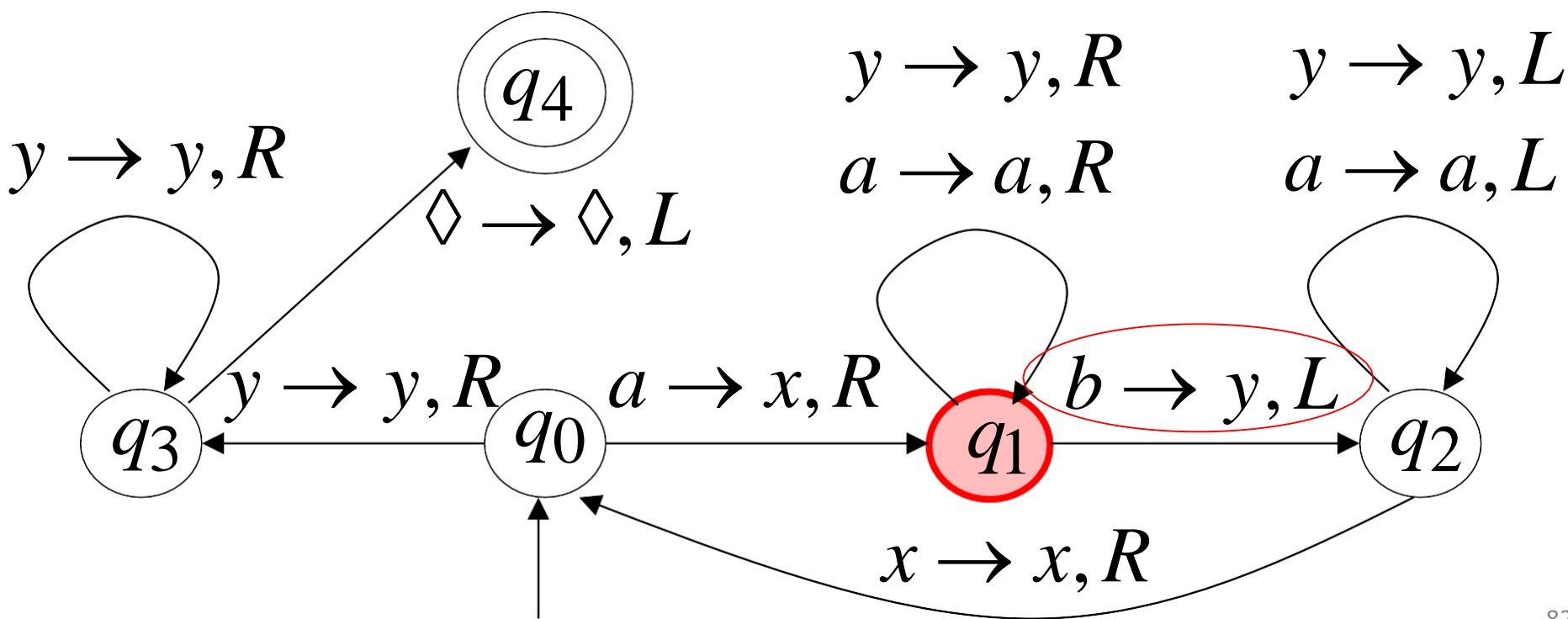
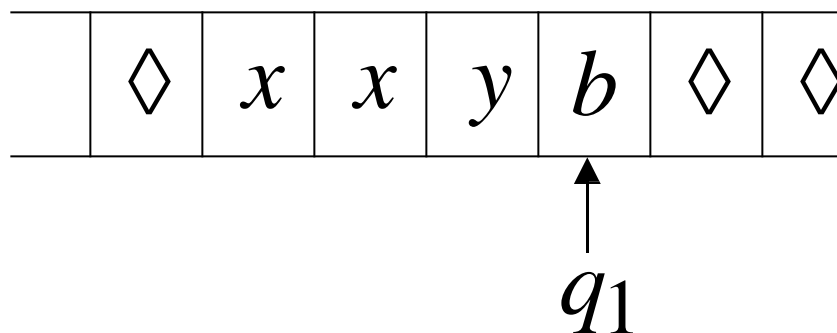
Time 5



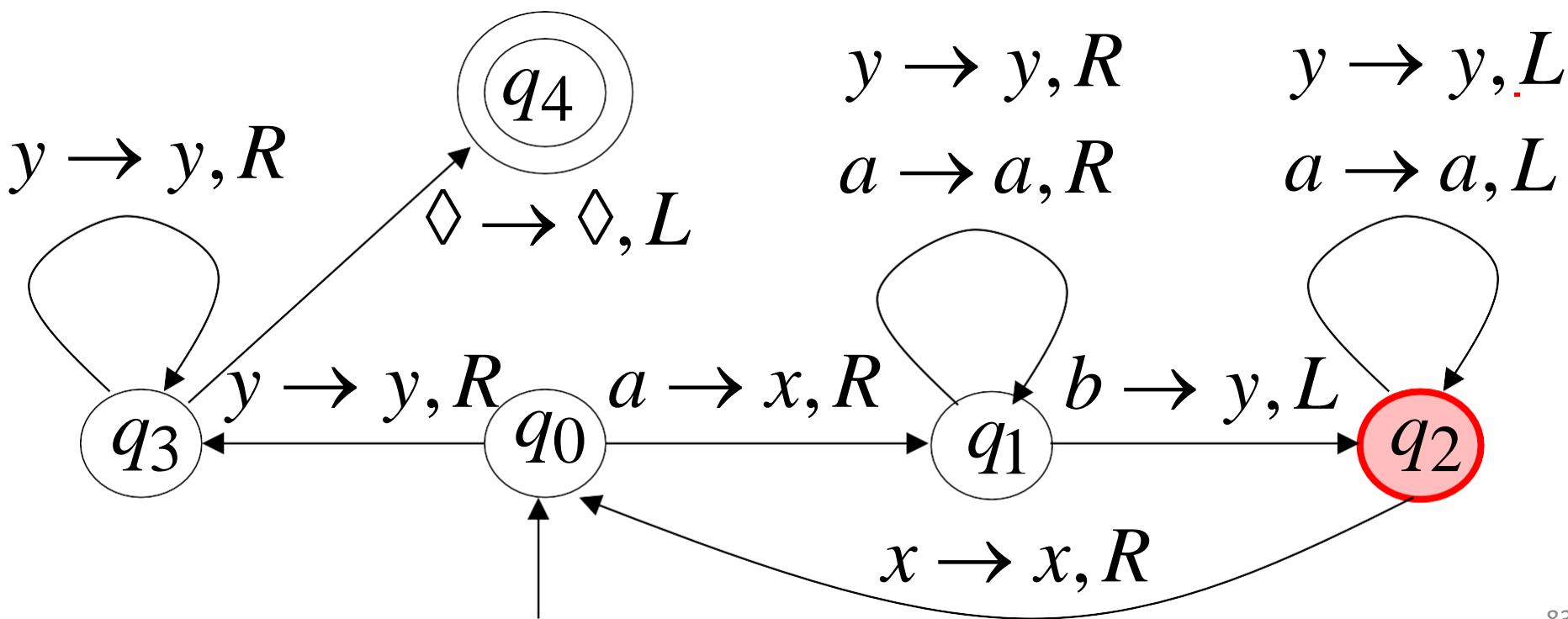
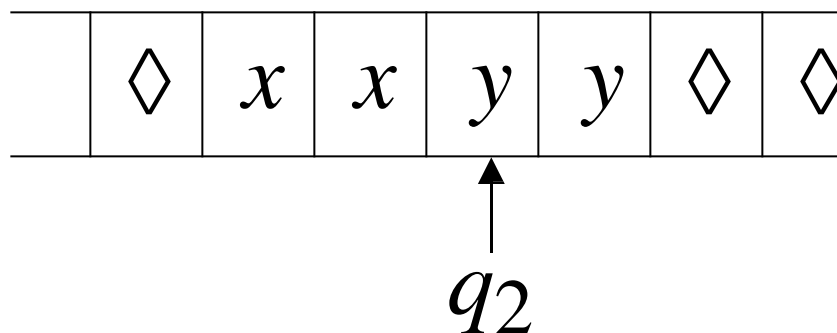
Time 6



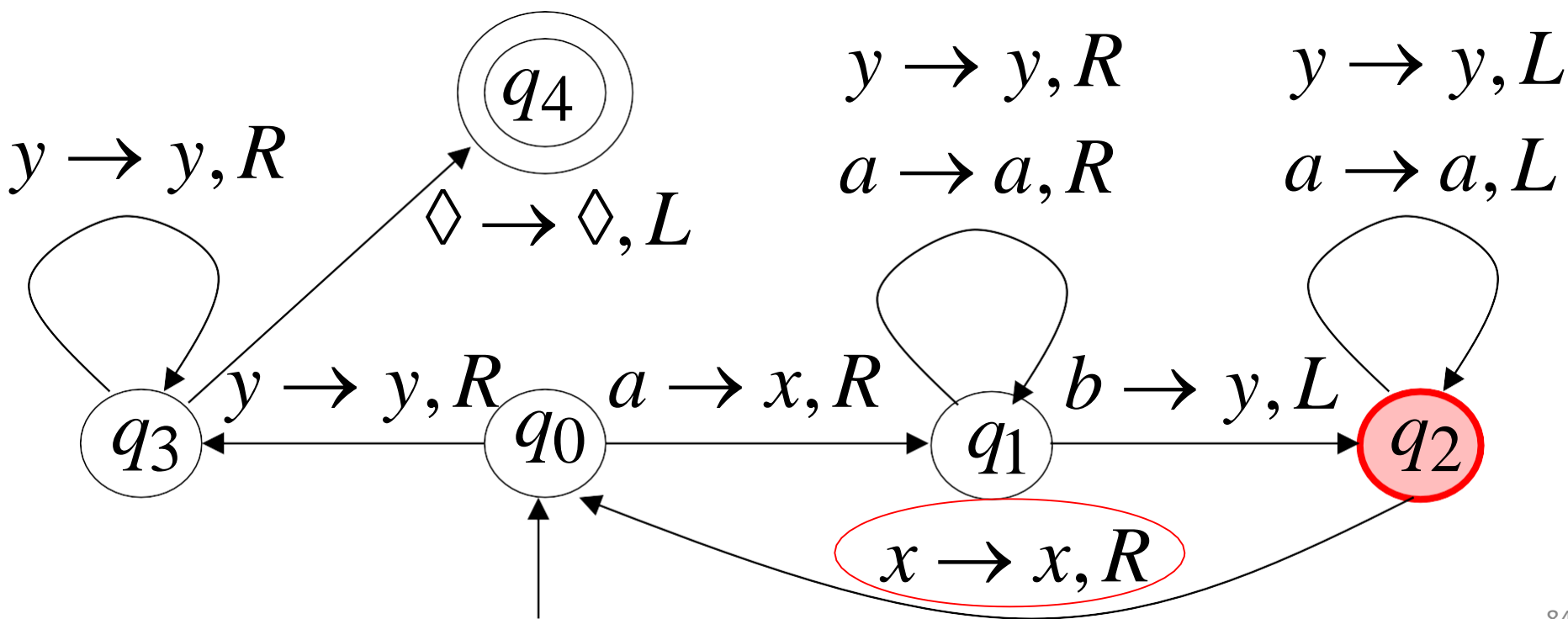
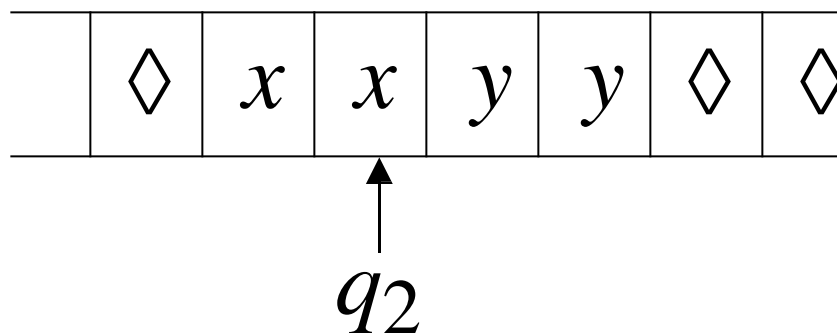
Time 7



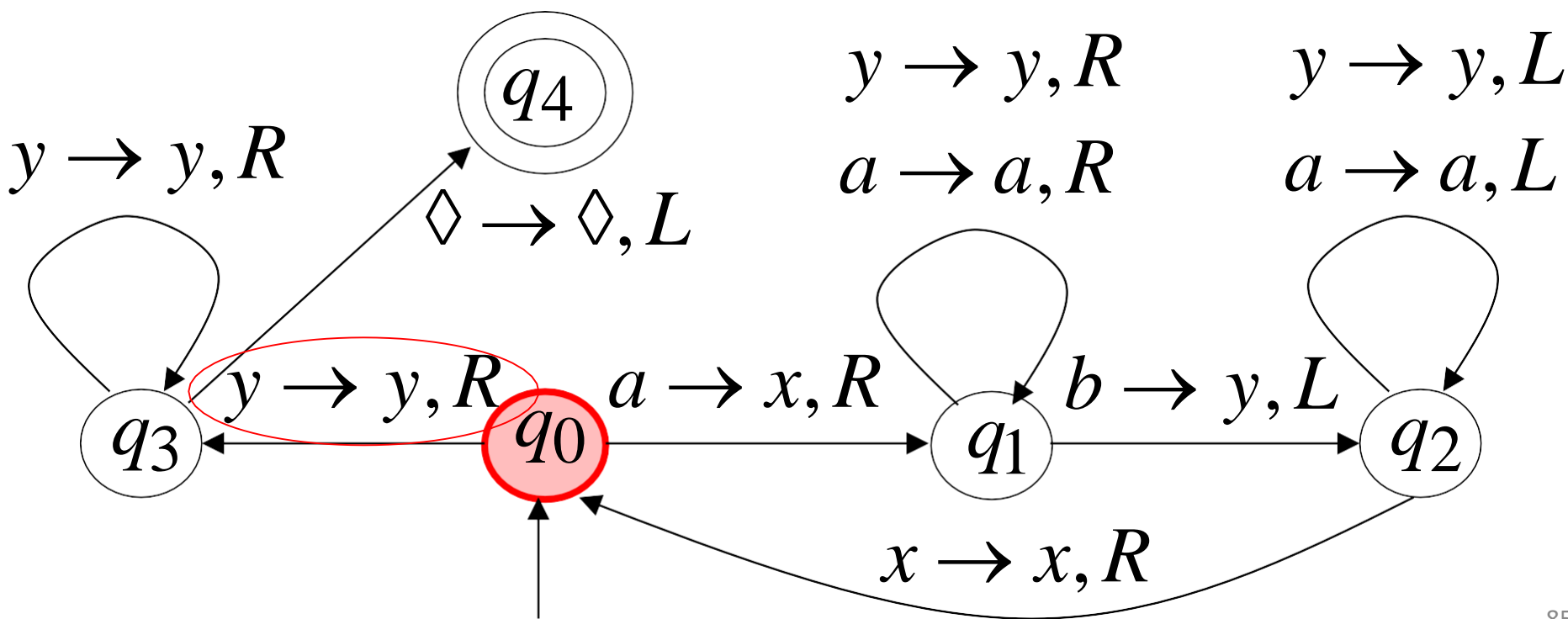
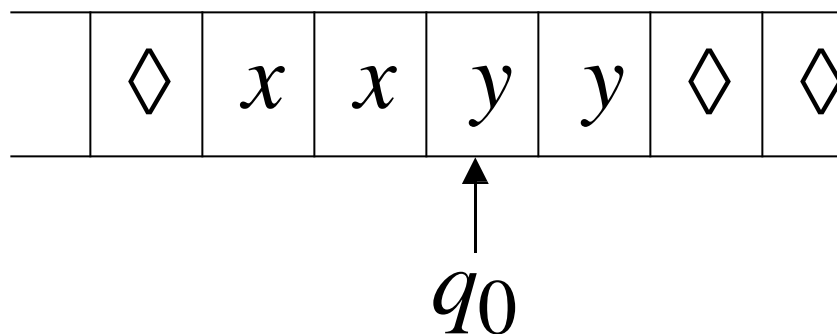
Time 8



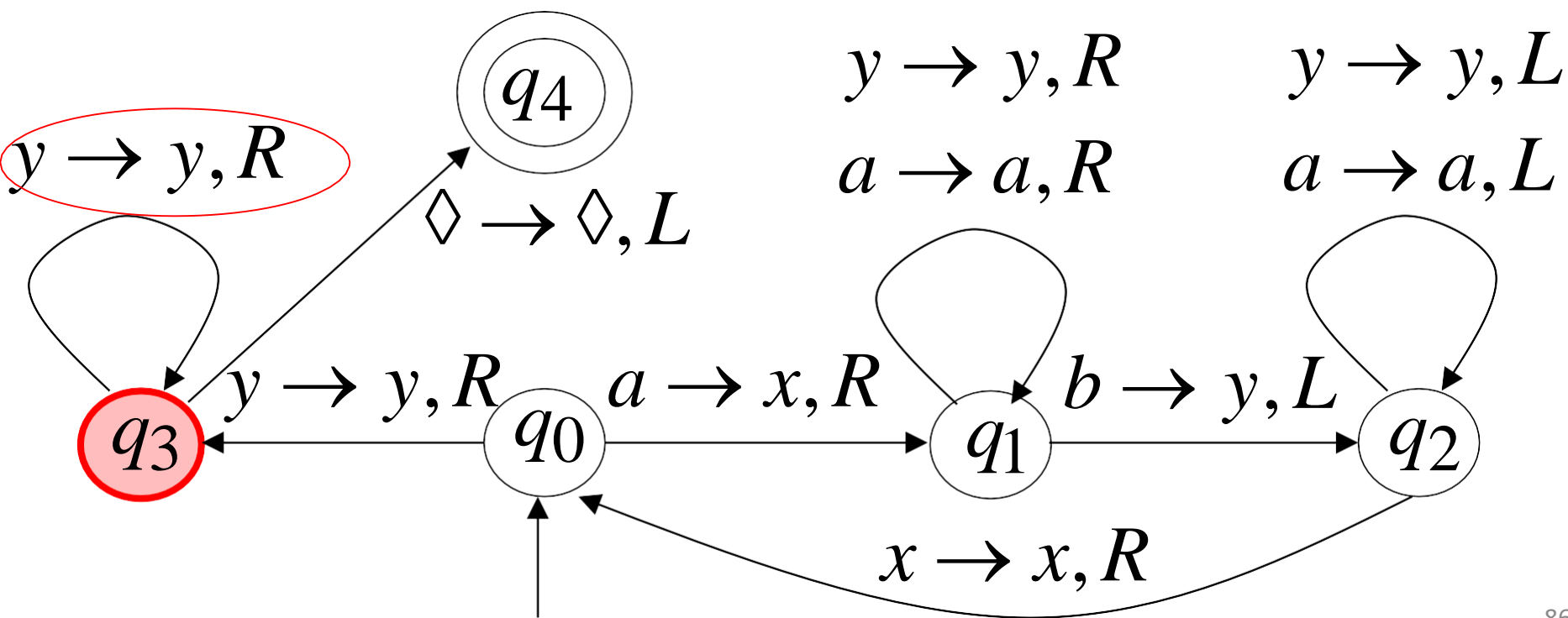
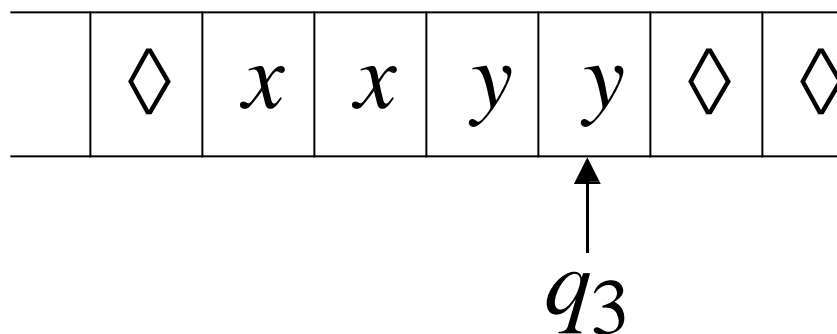
Time 9



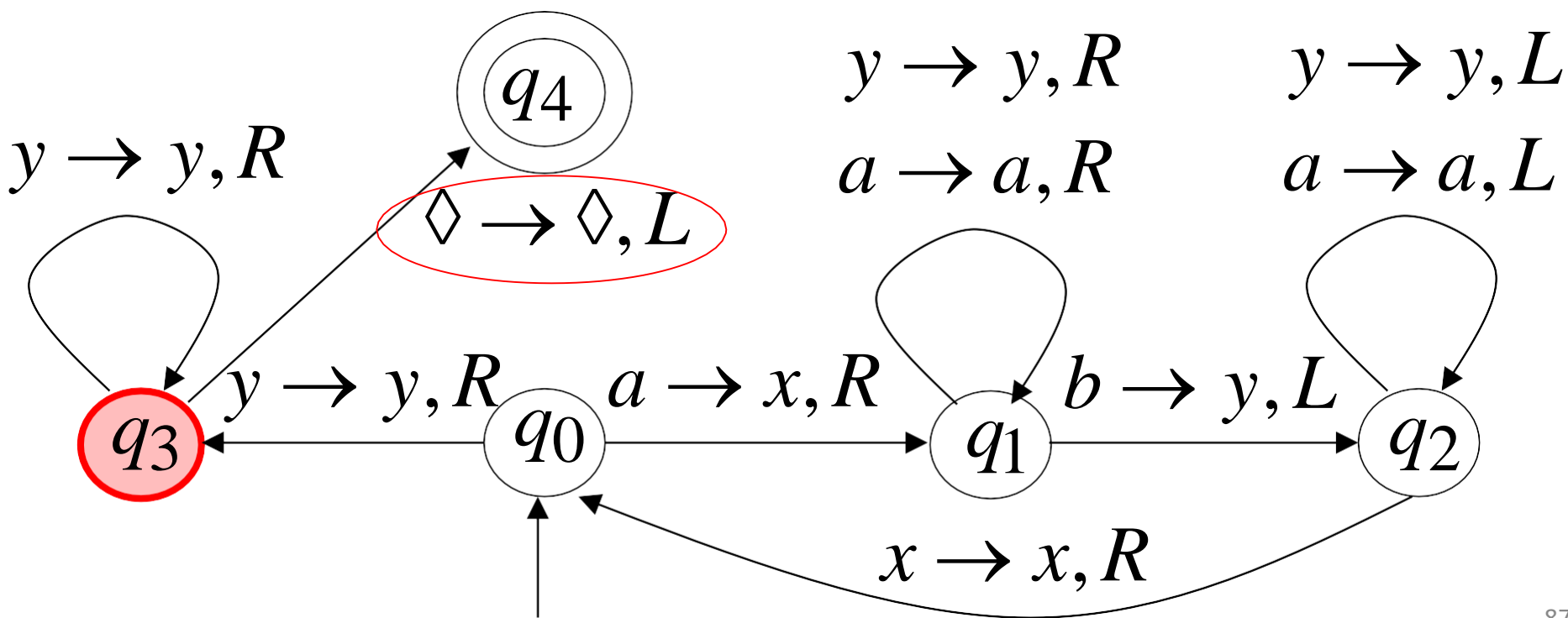
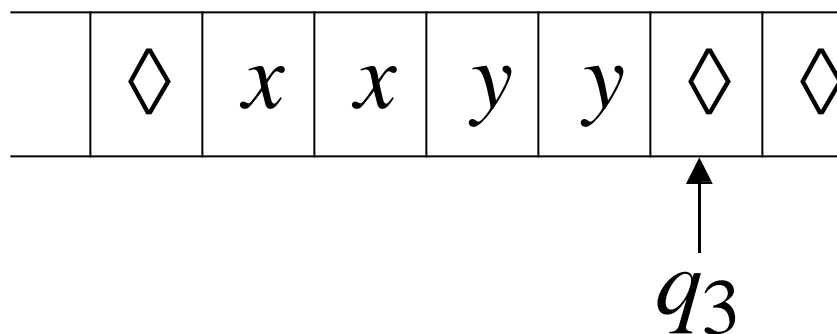
Time 10



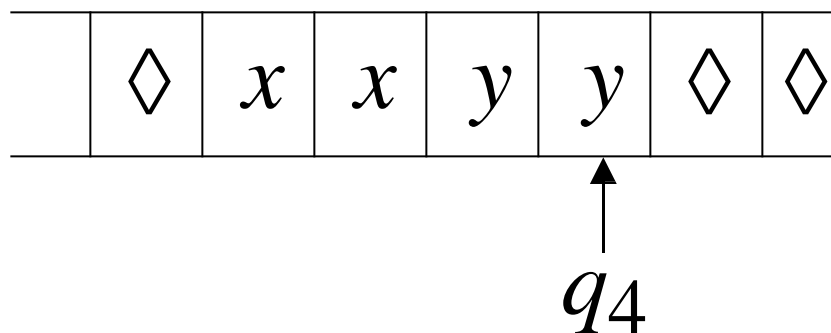
Time 11



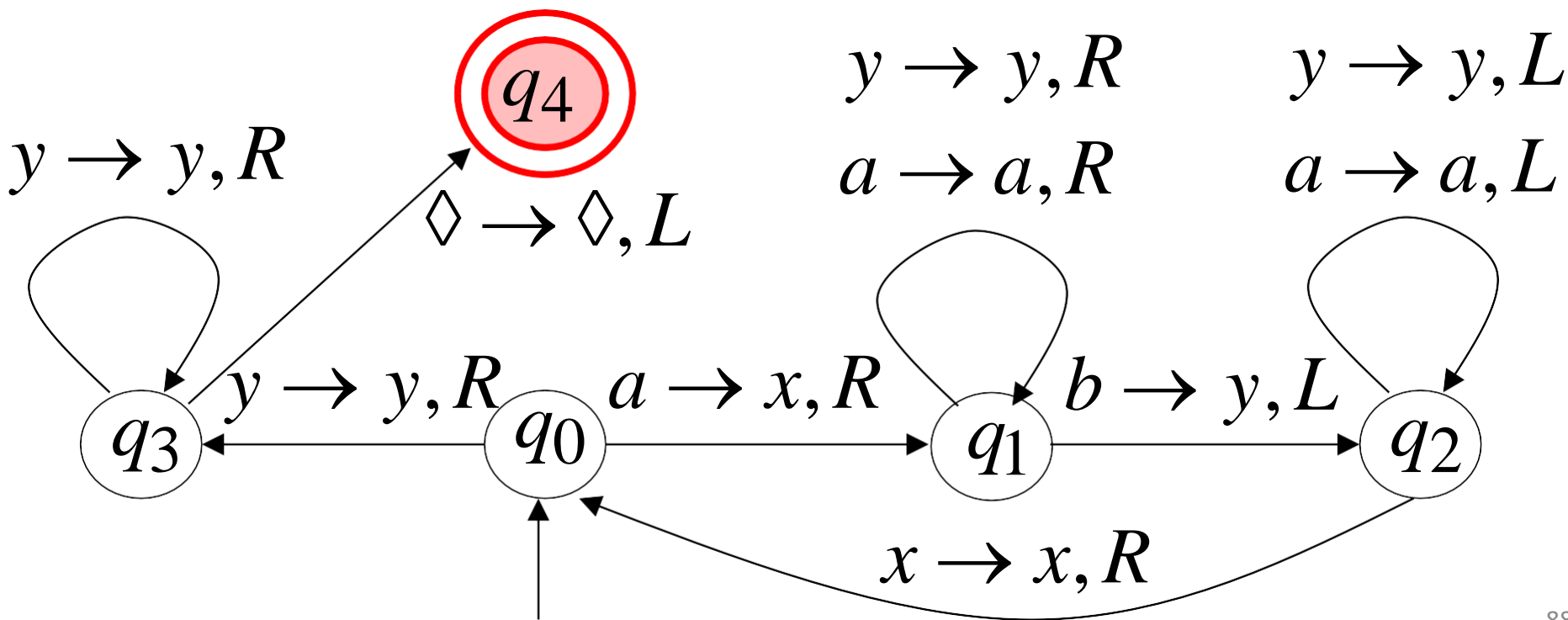
Time 12



Time 13



Halt & Accept



- $M = (\{q_0, q_1, q_2, q_3, q_4\}, \{0, 1\}, \{0, 1, x, y, B\}, \delta, q_0, B, \{q_4\})$
 - $M = (\{q_0, q_1, q_2, q_3, q_4\}, \{a, b\}, \{a, b, x, y, B\}, \delta, q_0, B, \{q_4\})$
 - $a=0, b=1$
 - **Accepted**
 - **ID:** $q_0 0011 \vdash x q_1 011 \vdash x 0 q_1 11 \vdash x q_2 0Y1 \vdash q_2 x 0 y 1$
 $\vdash x q_0 0 y 1 \vdash x x q_1 y 1 \vdash x x y q_1 1 \vdash x x q_2 y y \vdash x q_2 x y y \vdash$
 $x x q_0 y y \vdash x x y q_3 y \vdash x x y y q_3 B \vdash x x y q_4 y B$
 - **Rejected**
 - **ID:** $q_0 0010 \vdash x q_1 010 \vdash x 0 q_1 10 \vdash x q_2 0Y0 \vdash q_2 x 0 y 0$
 $\vdash x x q_1 y 0 \vdash x x y q_1 0 \vdash x x y 0 q_1 B$
- In state q_1 , M has no move on tape symbol B

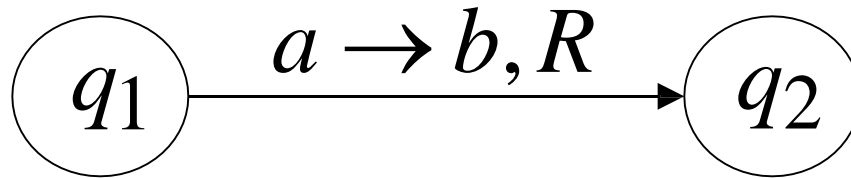
Observation:

If we modify the
machine for the language $\{a^n b^n\}$

we can easily construct
a machine for the language $\{a^n b^n c^n\}$

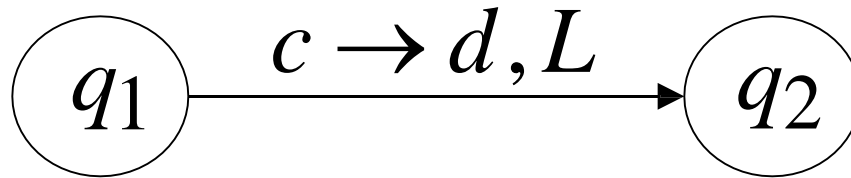
Formal Definitions for Turing Machines

Transition Function



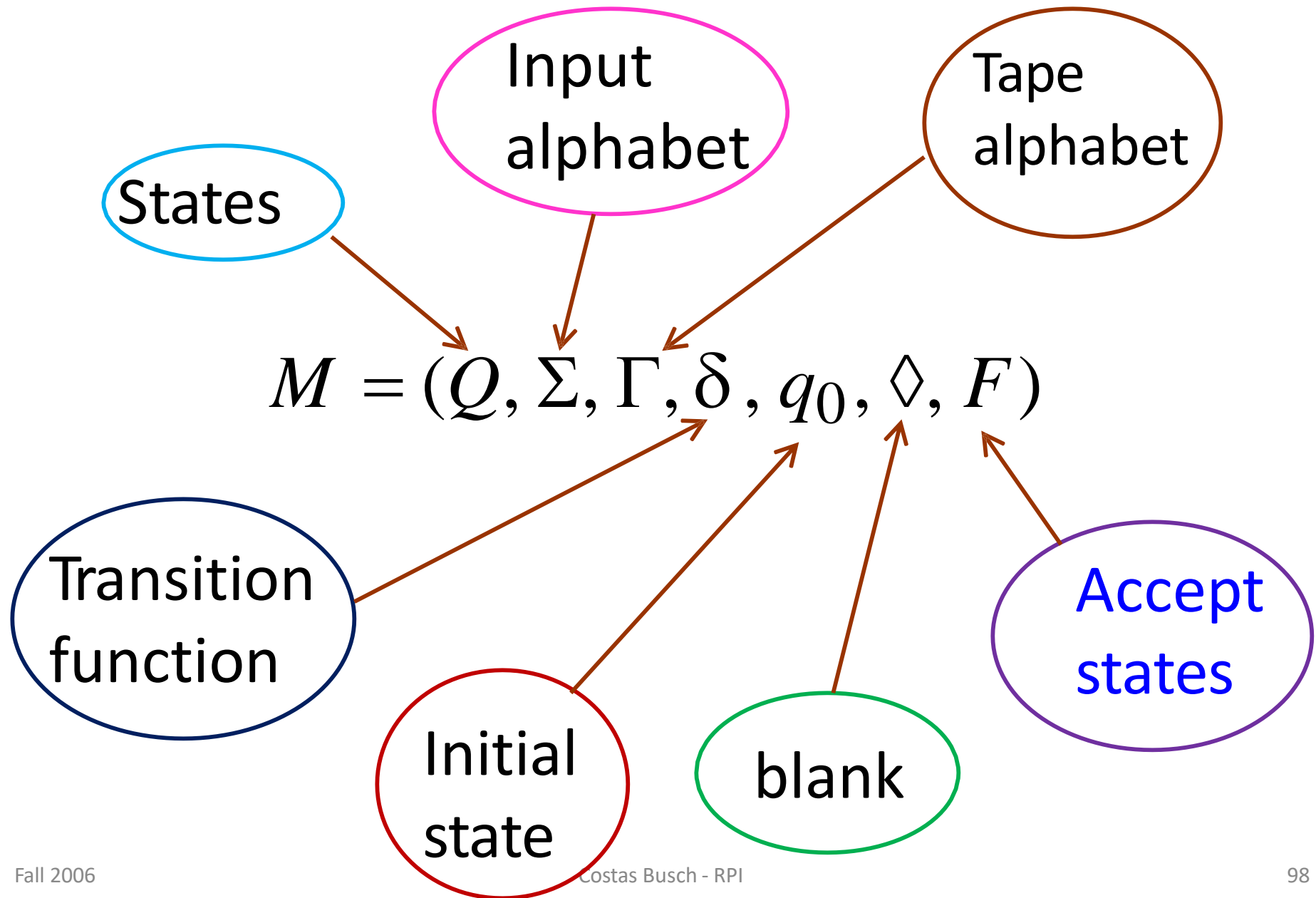
$$\delta(q_1, a) = (q_2, b, R)$$

Transition Function

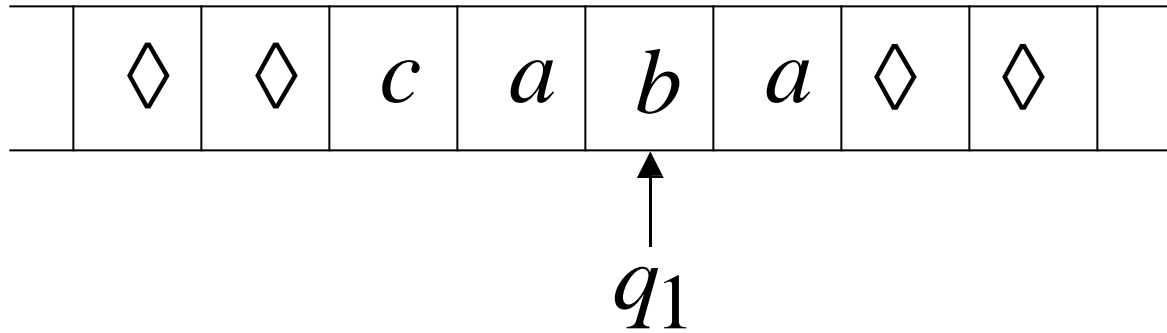


$$\delta(q_1, c) = (q_2, d, L)$$

Turing Machine:



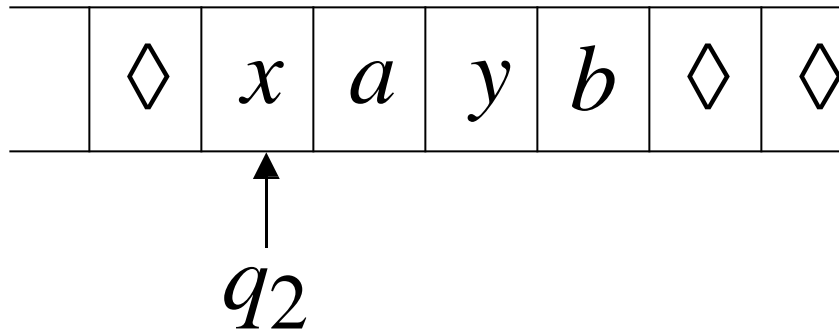
Configuration



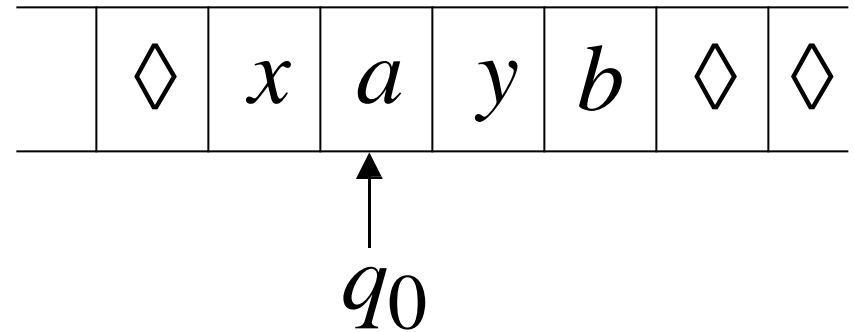
Instantaneous description:

$ca\ q_1\ ba$

Time 4

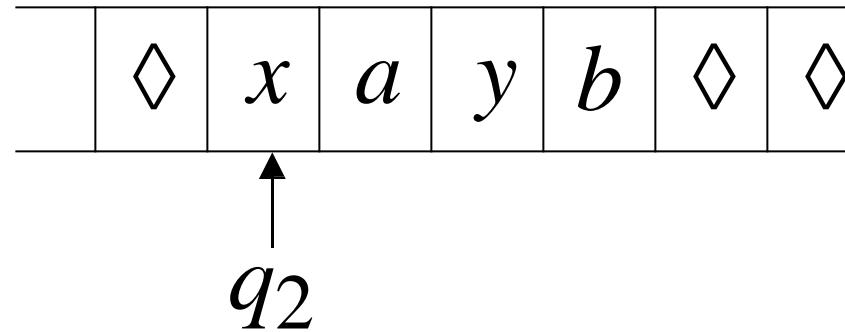


Time 5

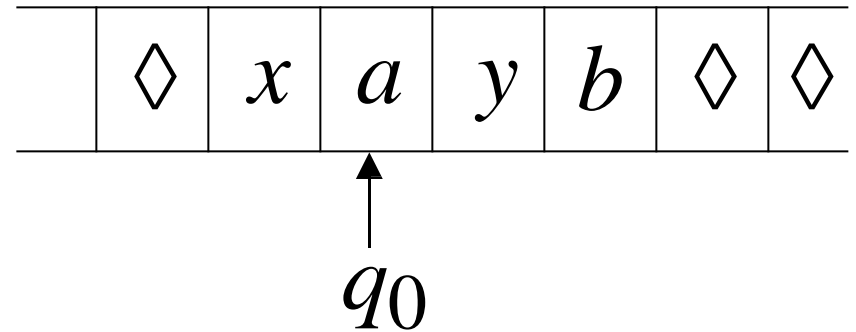


A Move: $q_2 \ x a y b \rightarrow x \ q_0 \ a y b$
(yields in one move)

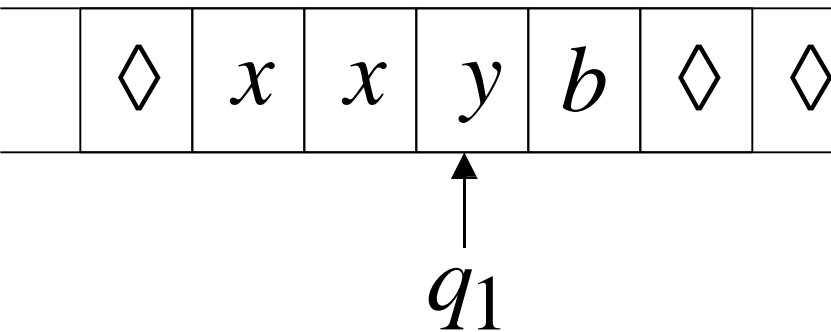
Time 4



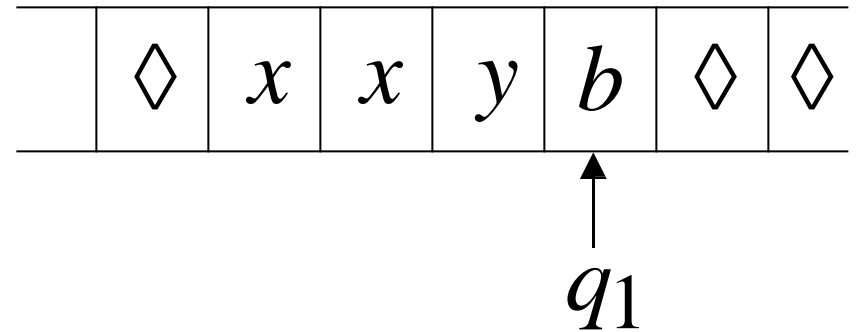
Time 5



Time 6



Time 7



A computation

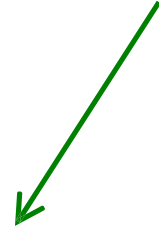
$q_2 \ x a y b$ ' $x \ q_0 \ a y b$ ' $x x \ q_1 \ y b$ ' $x x y \ q_1 \ b$

$$q_2 \ x a y b \quad ' \quad x \ q_0 \ a y b \quad ' \quad x x \ q_1 \ y b \quad ' \quad x x y \ q_1 \ b$$

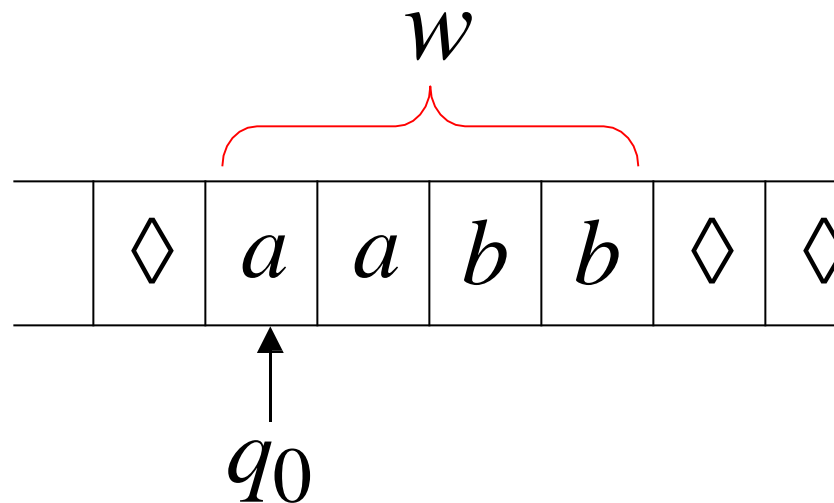
Equivalent notation:

$$q_2 \ x a y b \quad ' \quad x x y \ q_1 \ b^*$$

Initial configuration: $q_0 w$

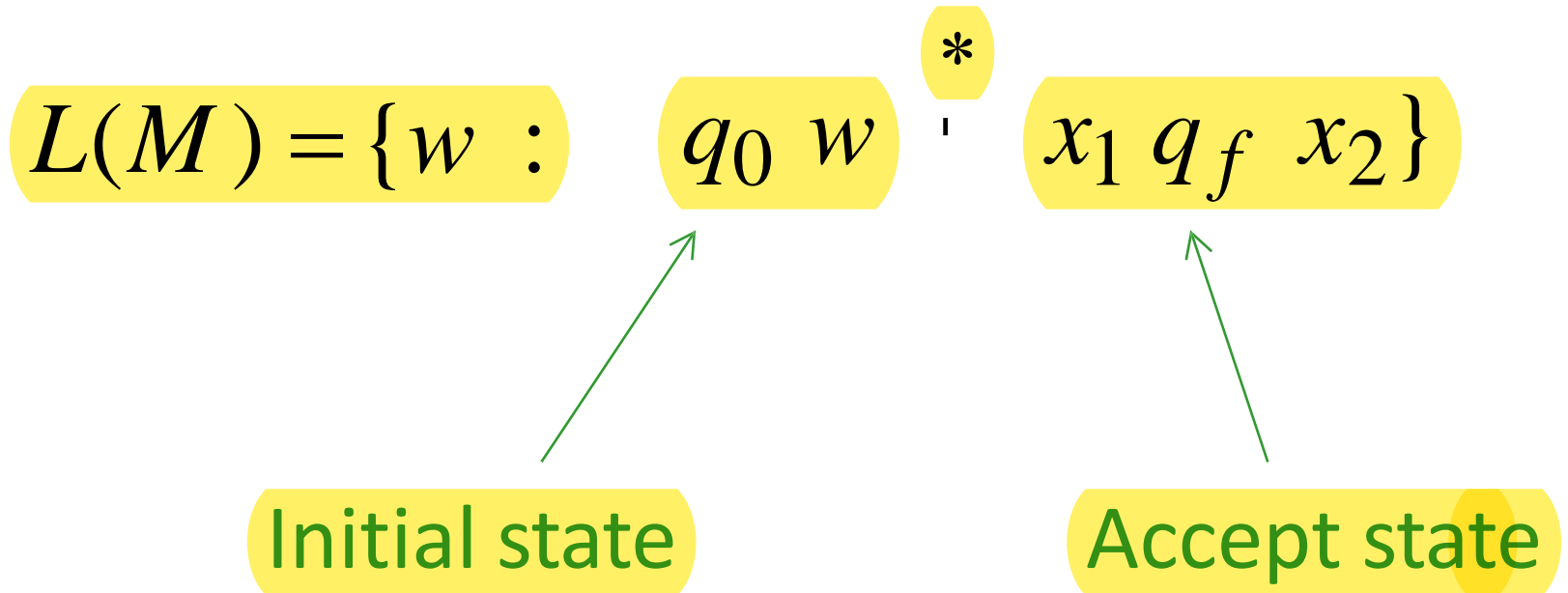


Input string



The Accepted Language

For any Turing Machine M



If a language L is accepted
by a Turing machine M
then we say that L is:

- Turing Recognizable

Other names used:

- Turing Acceptable
- Recursively Enumerable

TM: Computes the proper-subtraction function

$$M = (q_0, q_1, \dots, q_6, \{0, 1\}, \{0, 1, B\}, \delta, q_0, B)$$

State	Symbols		
	0	1	B
q_0	(q_1, B, R)	(q_5, B, R)	-
q_1	$(q_1, 0, R)$	$(q_2, 1, R)$	-
q_2	$(q_3, 1, L)$	$(q_2, 1, R)$	(q_4, B, L)
q_3	$(q_3, 0, L)$	$(q_3, 1, L)$	(q_0, B, R)
q_4	$(q_4, 0, L)$	(q_4, B, L)	$(q_6, 0, R)$
q_5	(q_5, B, R)	(q_5, B, R)	(q_6, B, R)
q_6	-	-	-

□ This TM is not used to accept inputs, we have omitted the 7th component (accepting states)

Transition Diagram

- See page 325, Figure 8.12

Programming Techniques for TM

- Storage in the state
- Multiple Tracks
- Subroutines

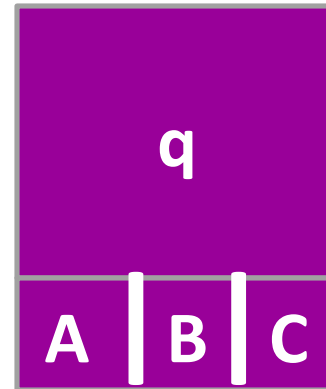
Storage in the state

- Finite control can use
 - ❑ not only to represent a position in the “program” of the TM,
 - ❑ but to hold a finite amount of data.

Techniques

- ❑ Storage
- ❑ Multiple track

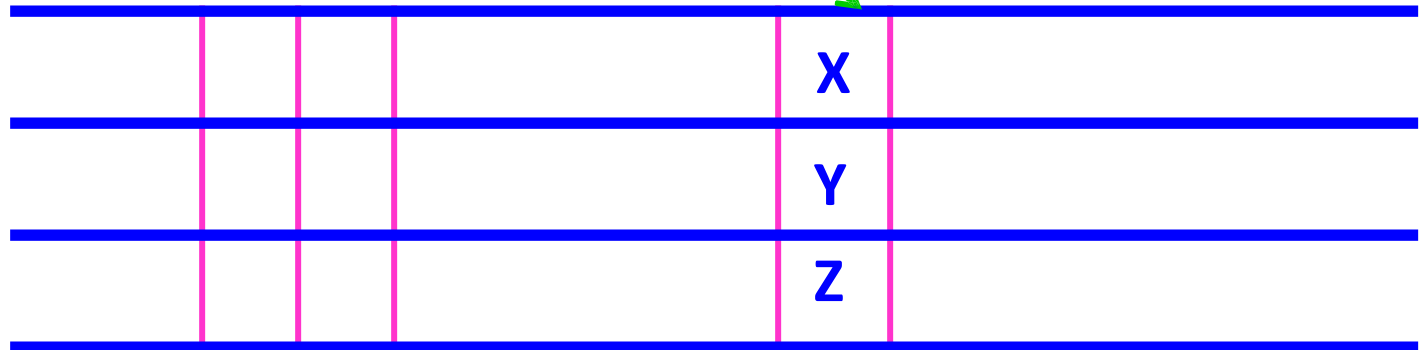
State
Storage



Track 1

Track 2

Track 3



- **Finite control**: state q ++ data elements (A, B, C)
- The technique requires no extension to the TM Model;
- Think state as a **tuple** = $[q, A, B, C]$

Multiple Tracks

- Tape composed of several tracks
- Each track can hold
 - one symbol
 - Tape alphabet of the TM consists of tuples,
With one component for each track
- The cell scanned by the tape head contains the symbol **[X, Y, Z]**

Subroutines

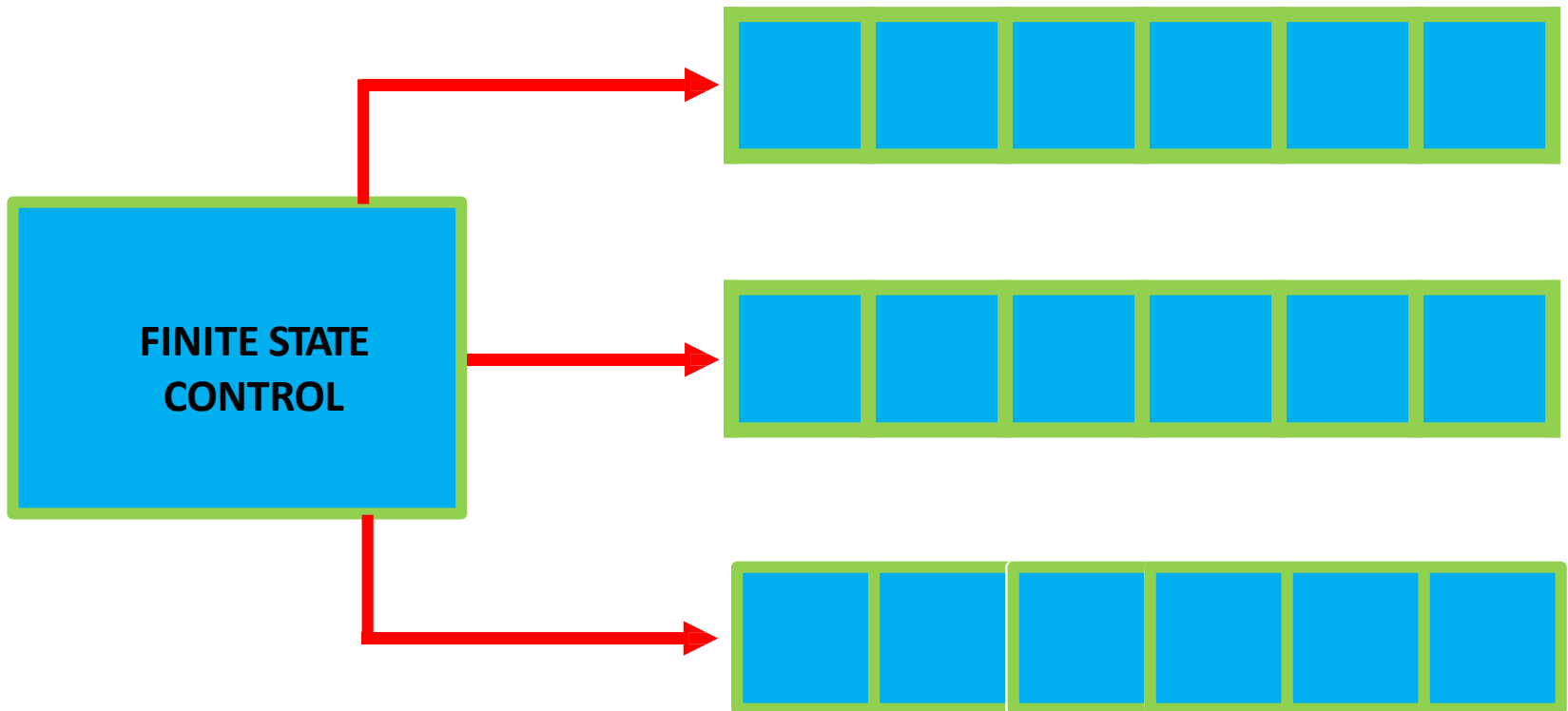
- TM build from a collection of interacting components/subroutines
- A TM subroutine is a set of states that perform some useful process
- This set of states includes 02 states:
 - a start state
 - another state that temporarily has no moves, and that serves as the return state to pass control to whatever other set of states called the subroutine
- The call of a subroutine occurs whenever there is a transition to its initial state
- Since the TM has no mechanisms for remembering a return address,
 - that is, a state to go to after it finishes,
- a TM call for one subroutine to be called from several states,
- we can make copies of the subroutine, using a new set of states for each copy
- The calls are made to the start states of different copies of the subroutine, and each copy returns to a different state.

Extensions to the Basic TM

- **Multi-tape TM**: simulate real computers
- **Nondeterministic TM**: allow to make any of a finite set of choices of move in a given situation

Multi-tape TM

- A finite control
- Some finite number of tapes
- Each tapes is divided into cells & each cell can hold any symbol of the finite tape alphabet
- The set of tape symbols includes
 - a blank
 - has a subset called the input symbols,
- The set of states includes: initial state and some accepting states.



Simulating a TM by Computer

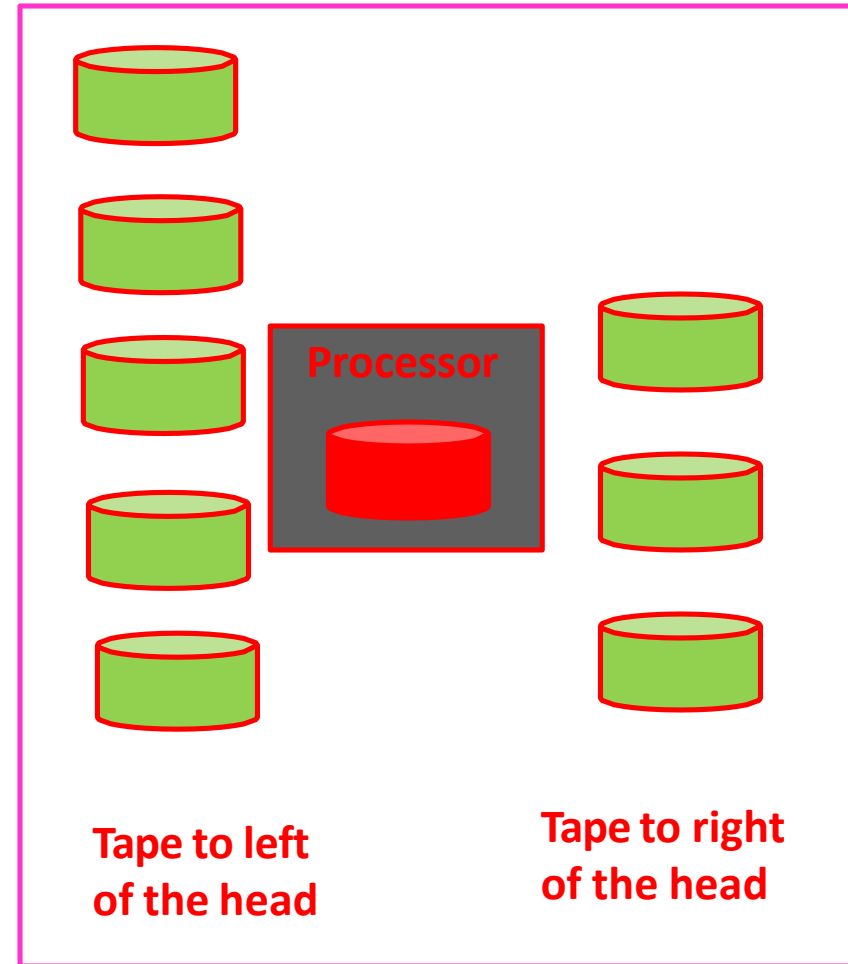
- Given a TM M , we write a program that acts like M
- Since there are only a finite number of states & a finite number of transitions rules, **program can encode states as character strings & use a table of transitions, which it looks up to determine each move.**
- Likewise, the **tape symbols can be encoded as character strings of a fixed length**, since there are only finite number of tape symbols.

How program is to simulate the TM tape?

- Tape grow infinitely long, but the computer's memory-main memory, disk & other storage devices---are finite
- Can we simulate an infinite tape with a fixed amount of memory?
- ❑ If there are no opportunity to replace storage devices, then in fact we cannot; a computer would then be a finite automation & only languages it could accept would be regular

- However, common computers have swappable storage devices, perhaps a “Zip” disk.
- In fact, the typical hard disk is removable & can be replaced by an empty, but otherwise identical disk
- **Assume:** as many disks as the computer needs is available
- Arrange that the disks are placed in 02 stacks

- One stack holds the data in cells of the TM tape that are located significantly to the left of the tape head
- The other stack holds data significantly to the right of the tape head
- The further down the stacks, the further away from the tape head the data is



- If the tape head of TM move sufficiently far to the left that it reaches cells that are not represented by the disk currently mounted in the computer, then it prints a message, “swap left”
- The currently mounted disk is removed by a human operator & placed on the top of the right stack
- The disk on top of the left stack is mounted in the computer & computation resumes

- Similarly, if TM's tape head reaches cells so far to the right that these cells are not represented by the mounted disk, then a “swap right” message is printed
- The human operator moves the currently mounted disk to the top of the left stack, & mounts the disk on top of the right stack in the computer
- If either stack is empty when the computer asks, that a disk from that stack be mounted, then the TM has entered an all-blank region of the tape
- In that case, the human operator must go to the store & buy a fresh disk to mount.

Comparing the running times of computers & Turing machines

- See textbook: P#361