18CSC301T – FORMAL LANGUAGE AND AUTOMATA

3rd Year CSE

SRM IST, Ramapuram

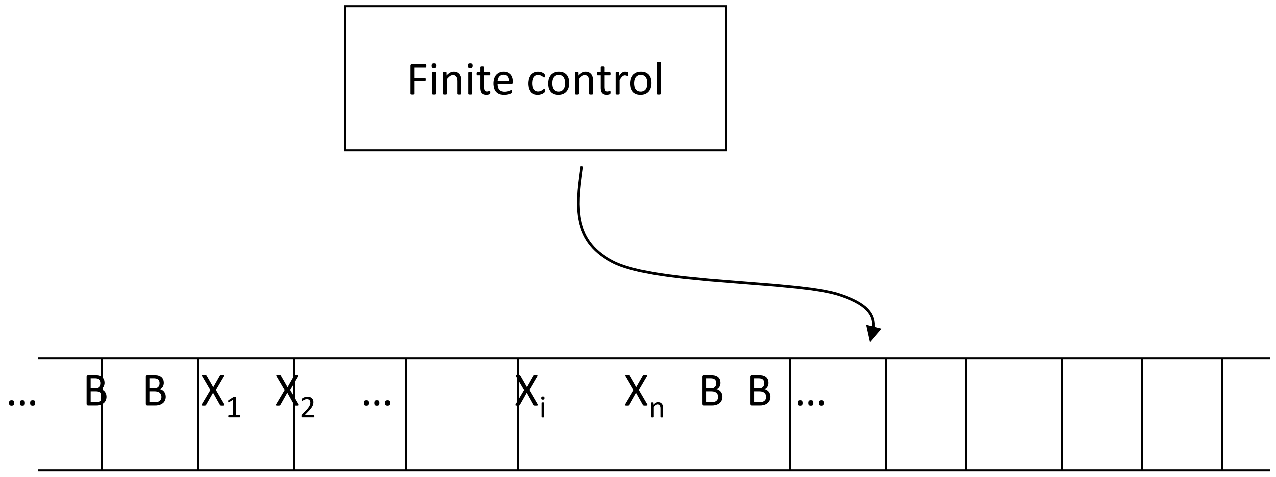
Turing Machines

Introduction

- TM’s described in 1936
  - Well before the days of modern computers but remains a popular model for what is possible to compute on today’s systems
  - Advances in computing still fall under the TM model, so even if they may run faster, they are still subject to the same limitations
- A TM consists of a finite control (i.e. a finite state automaton) that is connected to an infinite tape.

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.



Turing Machine Details

- 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
- Formally, the Turing Machine is denoted by the 8-tuple:
  - $M = (Q, \sum, \Gamma, \delta, q_0, B, q_a, q_r)$

Turing Machine Description

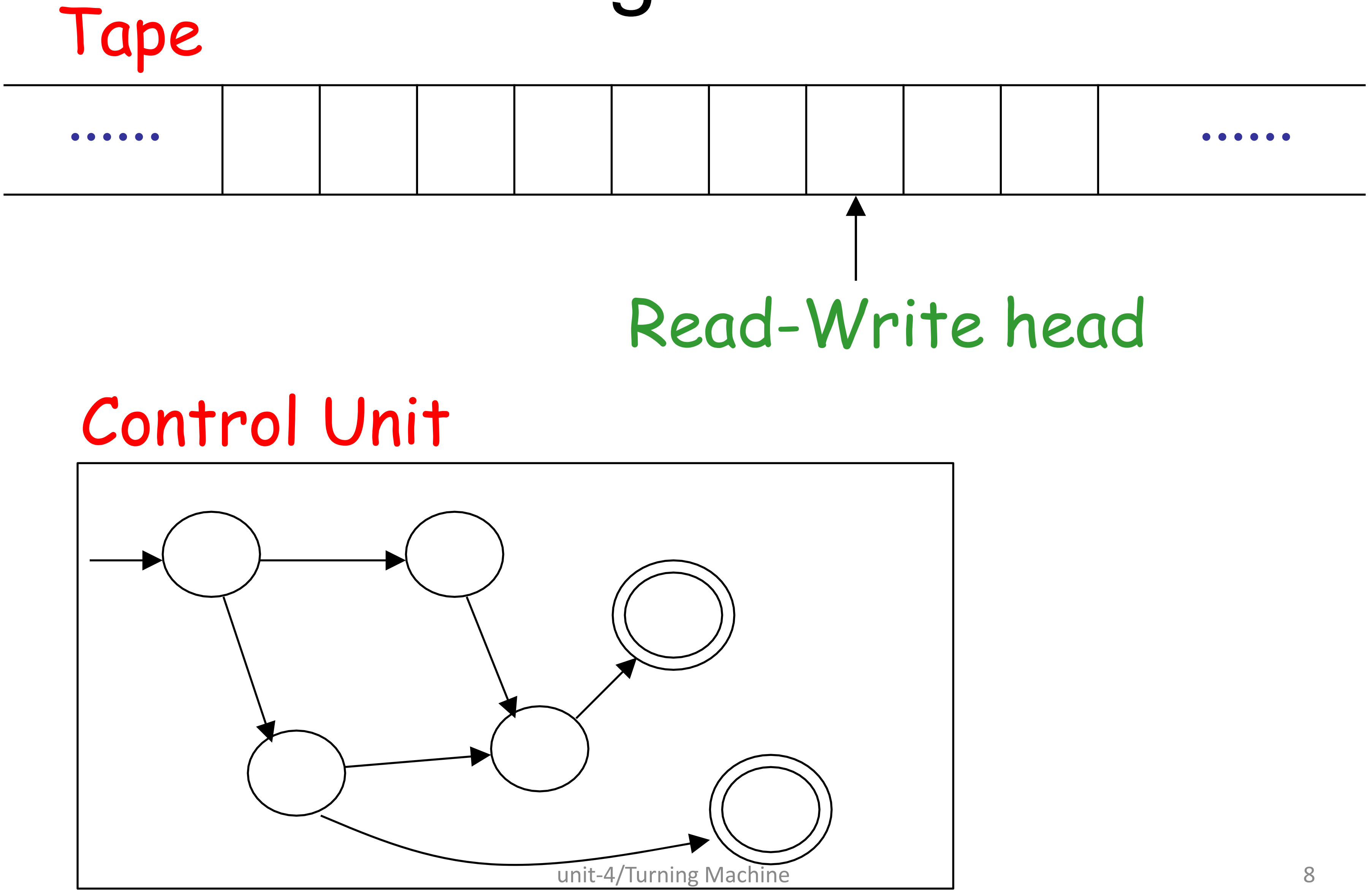
- Q = finite states of the control
- $\sum$ = finite set of input symbols, which is a subset of Γ below
- Γ = finite set of tape symbols
- δ = transition function. δ(q,X) are a state and tape symbol X.
  - The output is the triple, (p, Y, D)
  - Where p = next state, Y = new symbol written on the tape, D = direction to move the tape head
- q_0= start state for finite control
- B = blank symbol. This symbol is in Γ but not in $\sum$.
- q_{accept} = set of final or accepting states of Q.
- q_{reject} = set of rejecting states of Q

Instantaneous Description

- Sometimes it is useful to describe what a TM does in terms of its ID (instantaneous description), just as we did with the PDA.
- The ID shows all non-blank cells in the tape, pointer to the cell the head is over with the name of the current state
  - use the turnstile symbol ├ to denote the move.
  - As before, to denote zero or many moves, we can use ├*.
- For example, for the above TM on the input 10#10 we can describe our processing as follows:

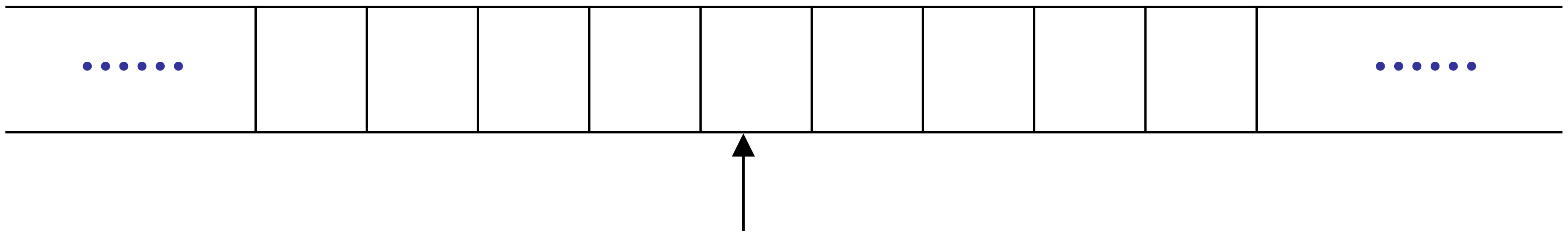
  Ba10#10B ├ B1a0#10B ├ B10a#10B ├ B10#b10B ├ B10#b10B ├ B10#1b0B ├ B10#10bB ├ B10#1c0B ├* cB10#10B ├ Bf10#10B ├* BXX#XXBl
- In this example the blanks that border the input symbols are shown since they are used in the Turing machine to define the borders of our input.

A Turing Machine



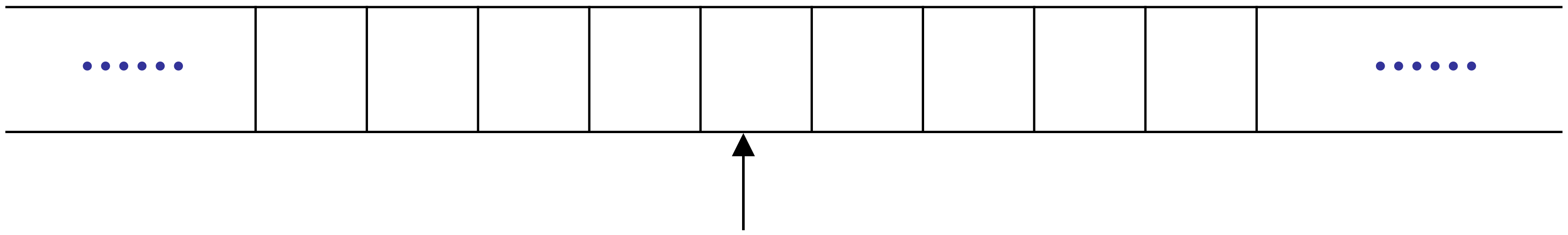
The Tape

No boundaries -- infinite length



Read-Write head

The head moves Left or Right

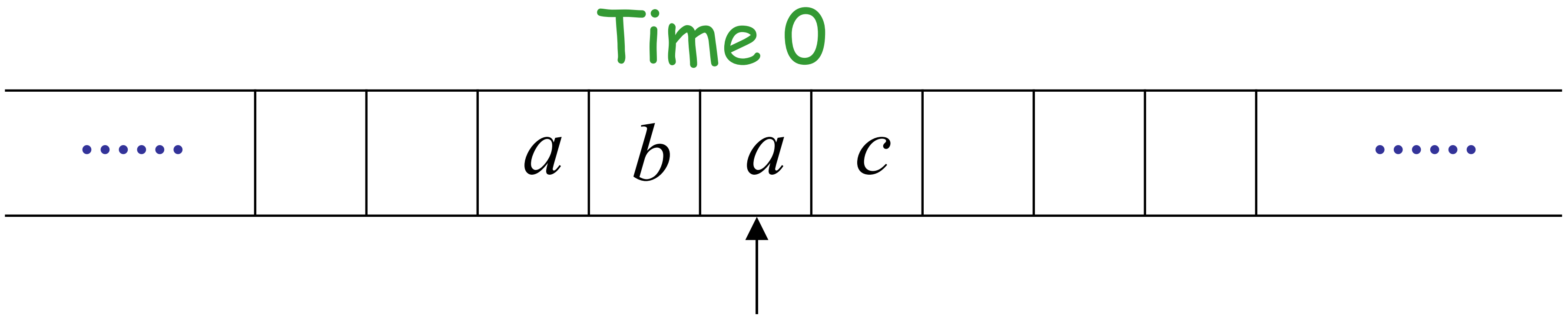


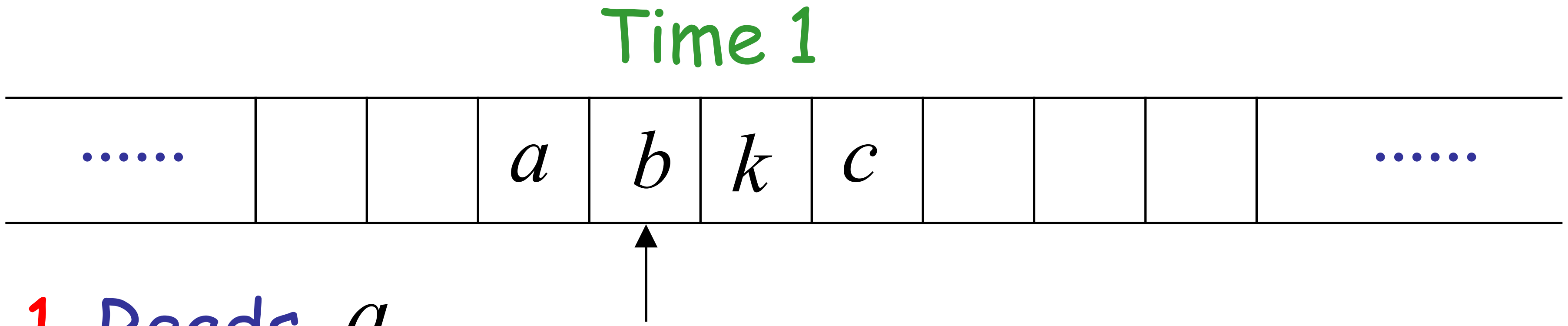
Read-Write head

The head at each time step:

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

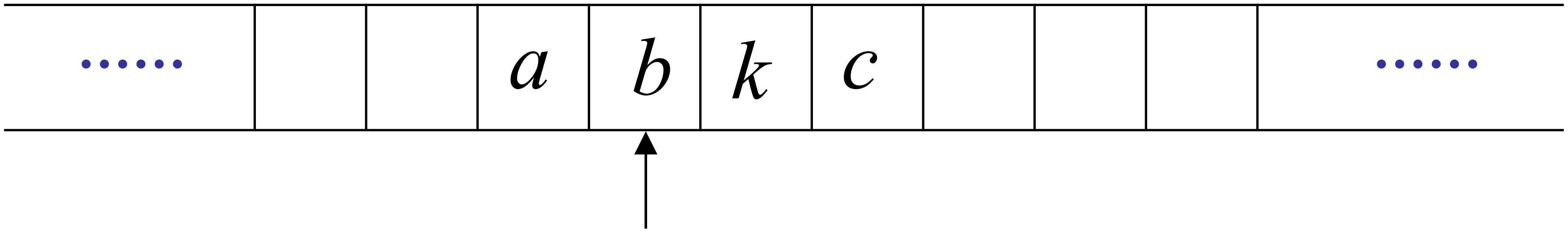
Example:



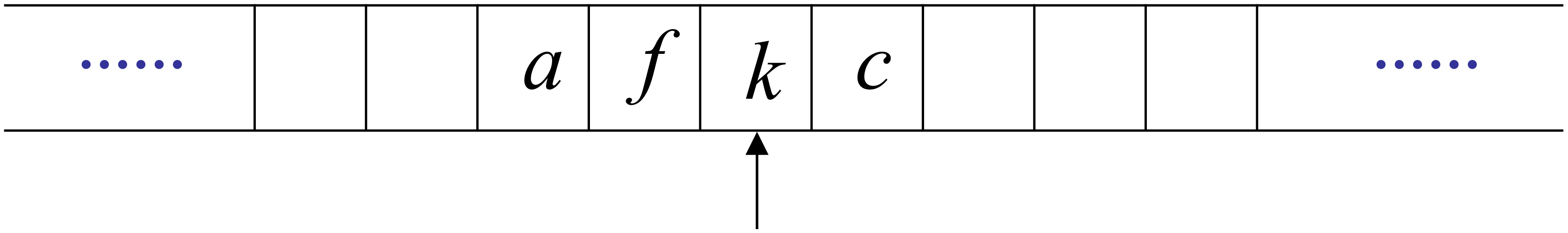


1. Reads a
2. Writes k
3. Moves Left

Time 1

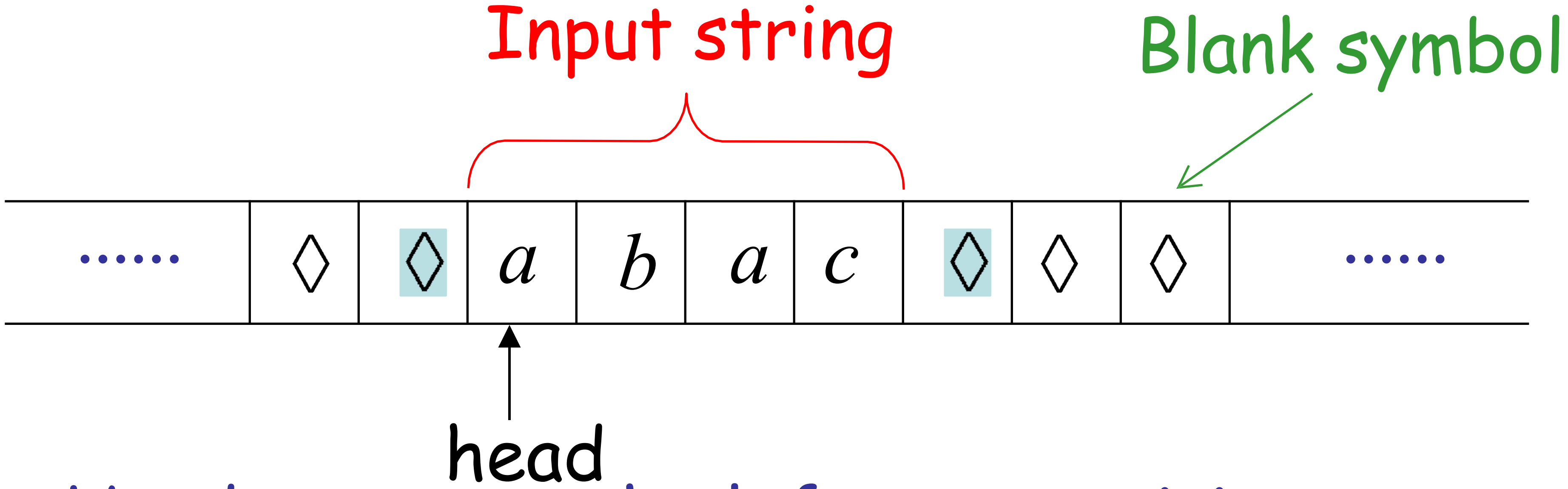


Time 2



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

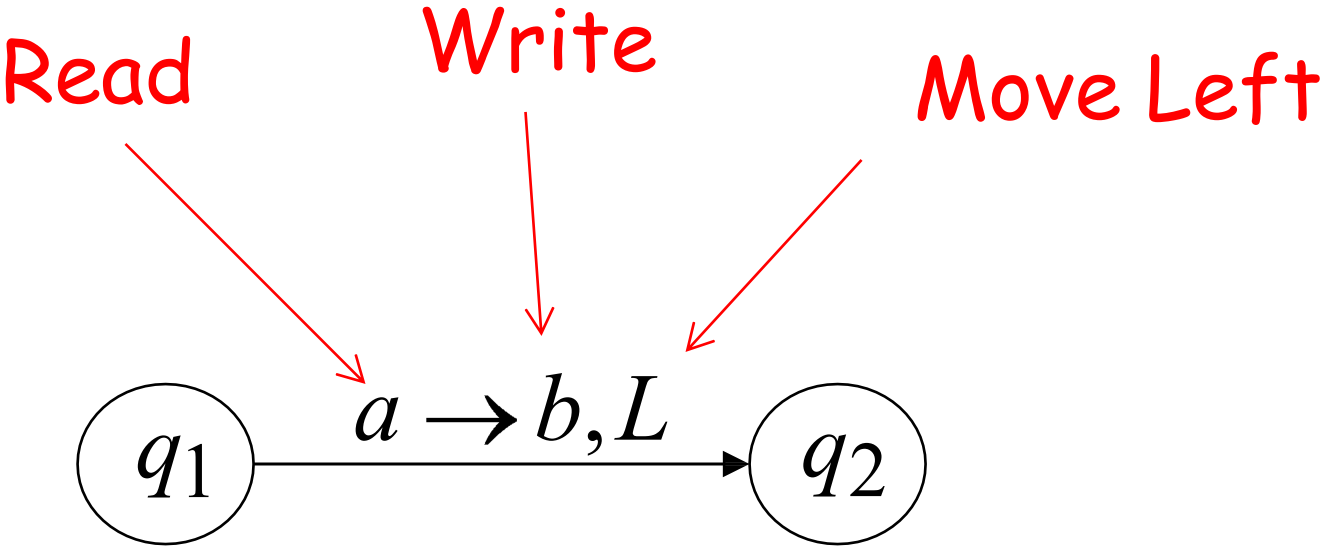
The Input String

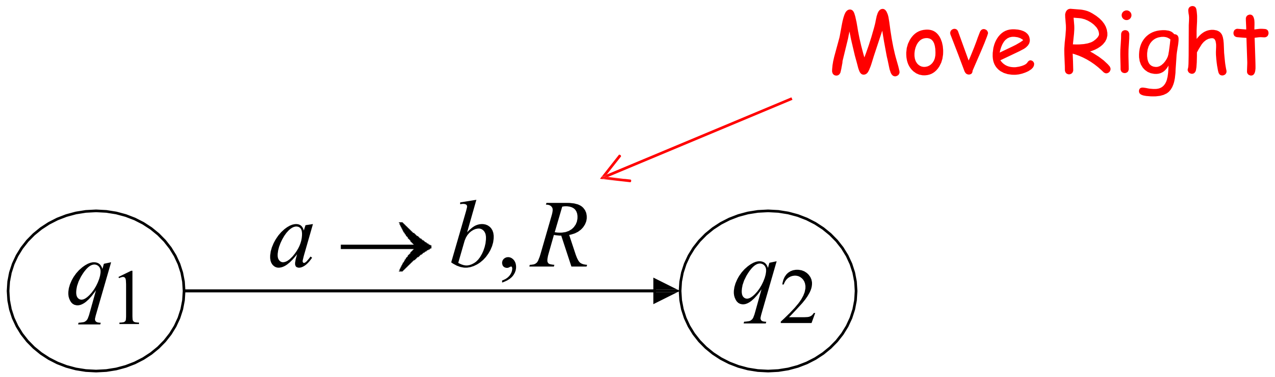


Head starts at the leftmost position of the input string

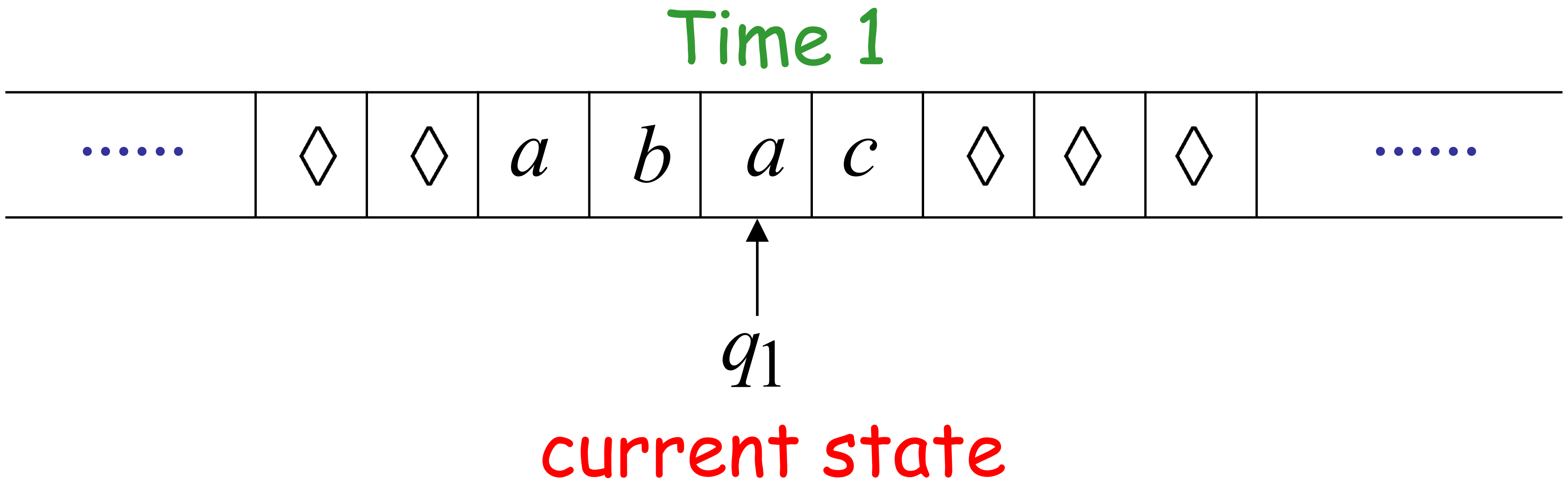
◊ **Are treated as left and right brackets for the input written on the tape.**

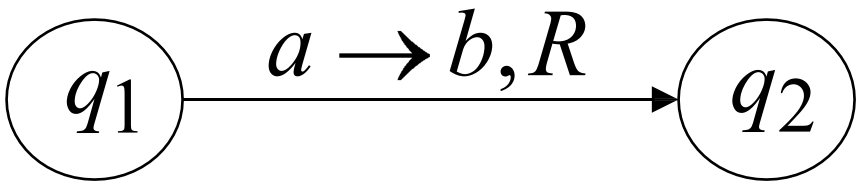
States & Transitions



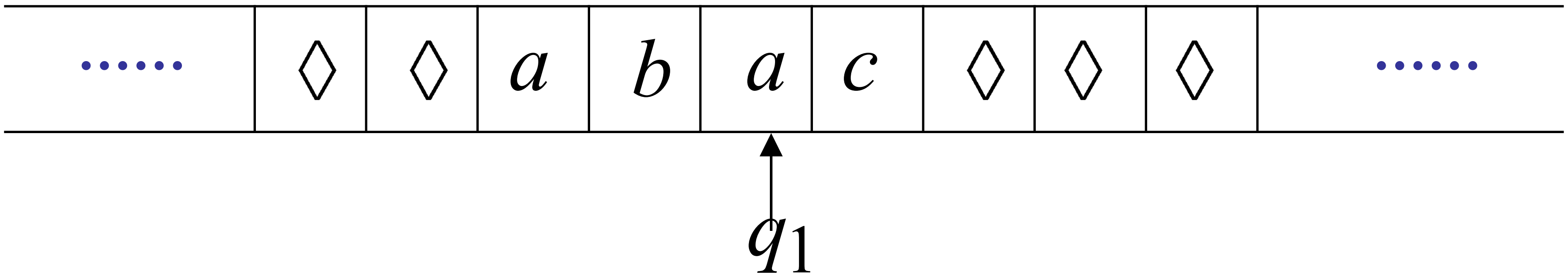


Example:

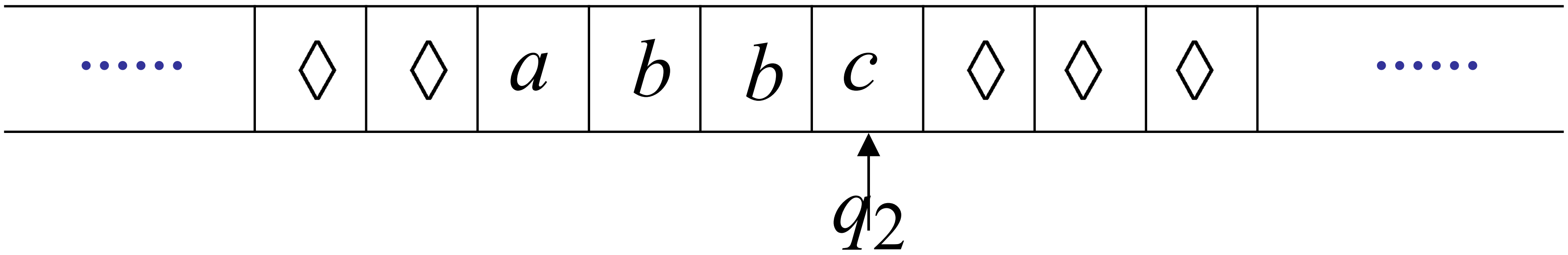


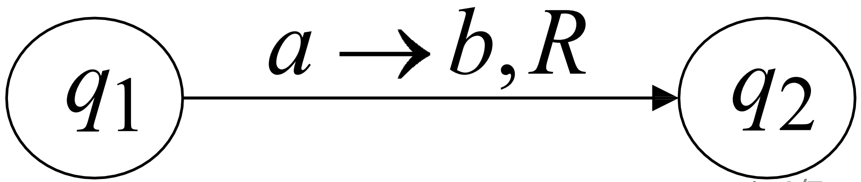


Time 1

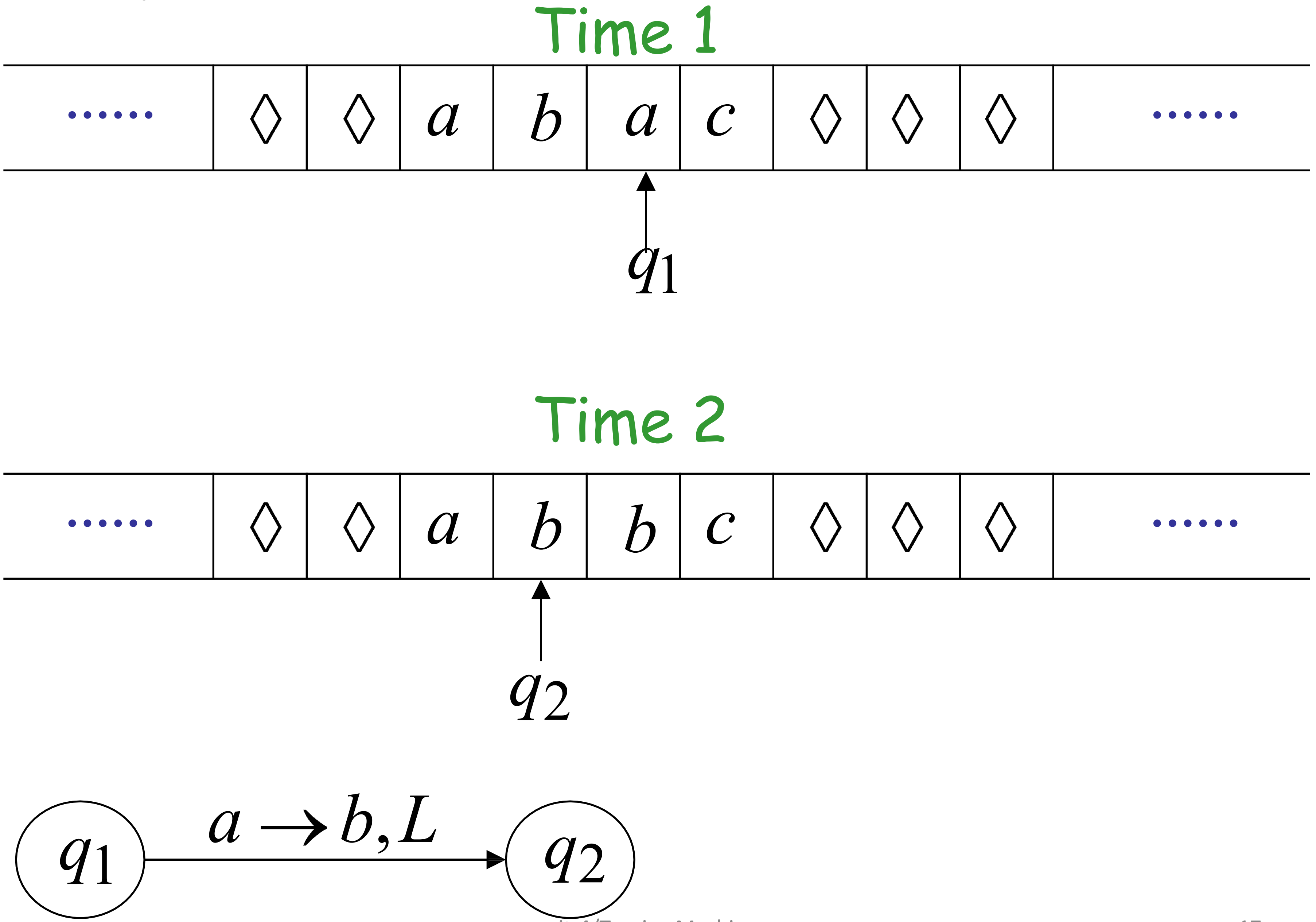


Time 2



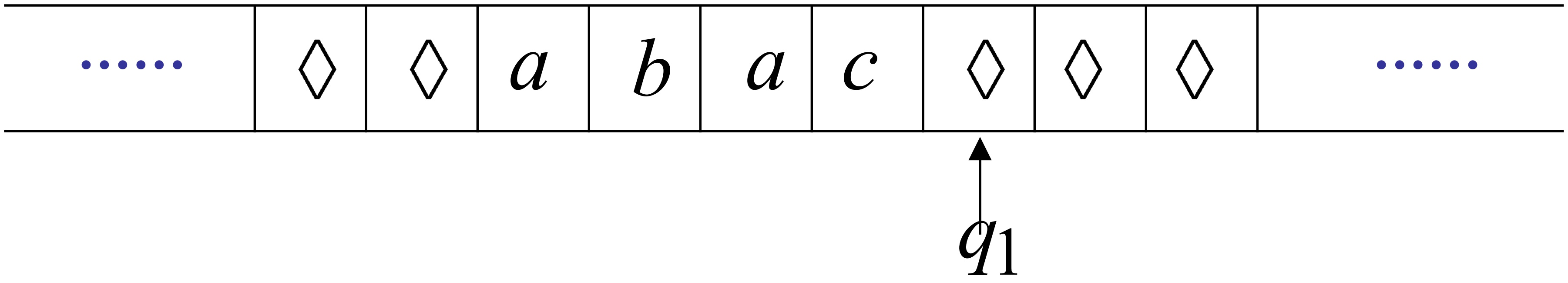


Example:

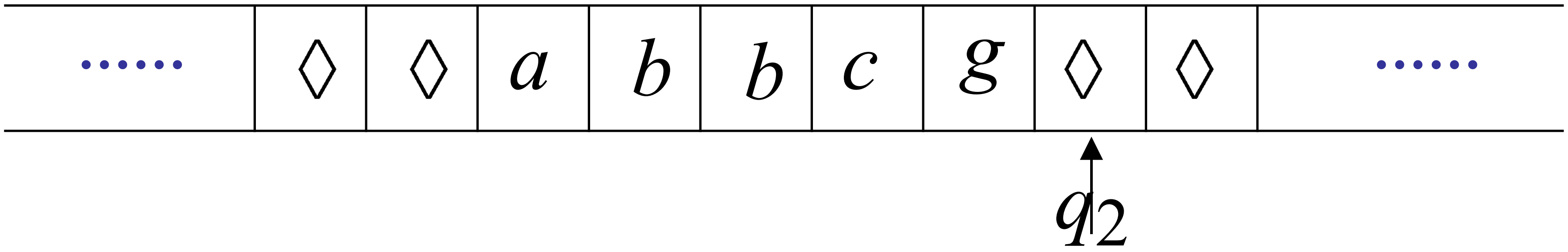


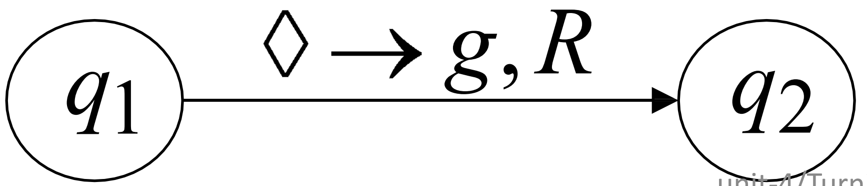
Example:

Time 1



Time 2

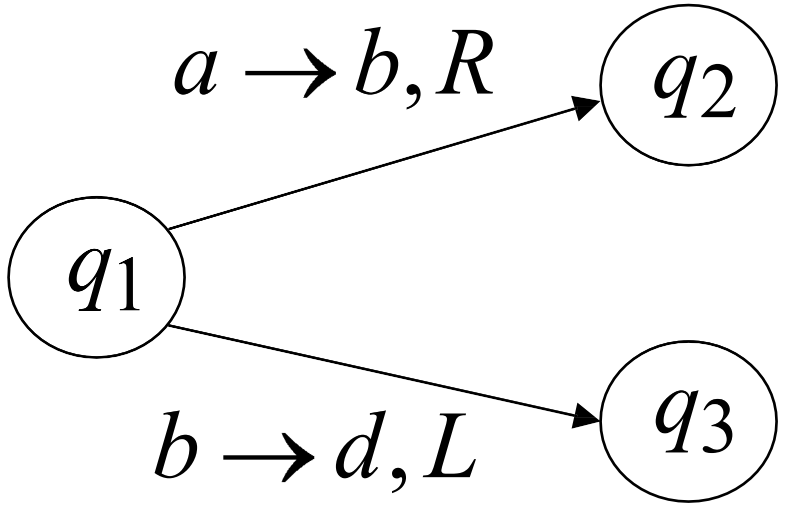




Determinism

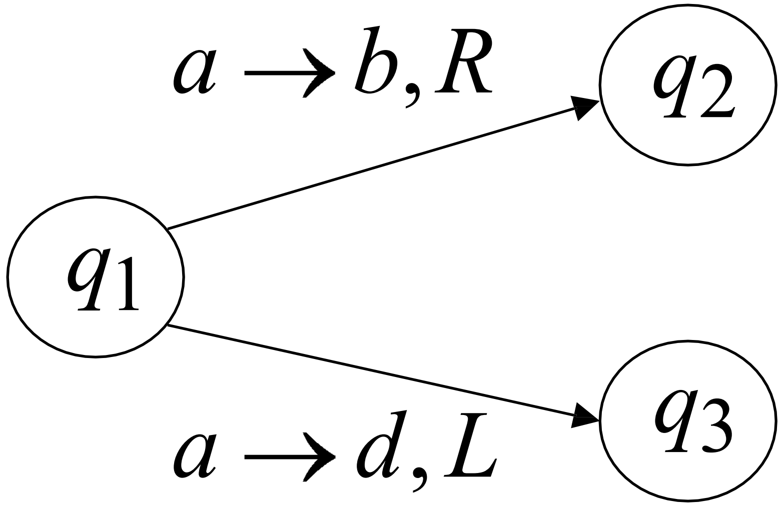
Turing Machines are deterministic

Allowed



$$\begin{tikzcd} & q_2 \\ q_1 \arrow[ur, "a \rightarrow b, R"] \arrow[dr, "b \rightarrow d, L"'] & \\ & q_3 \end{tikzcd}$$

Not Allowed

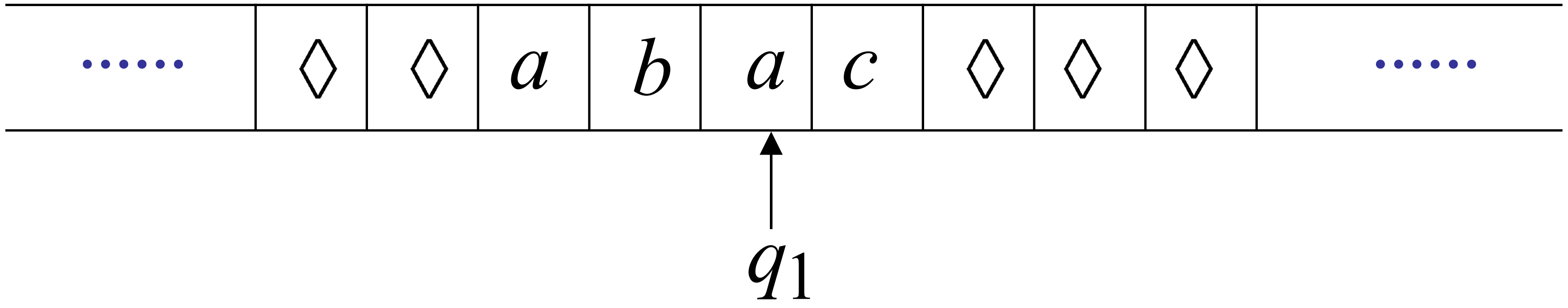


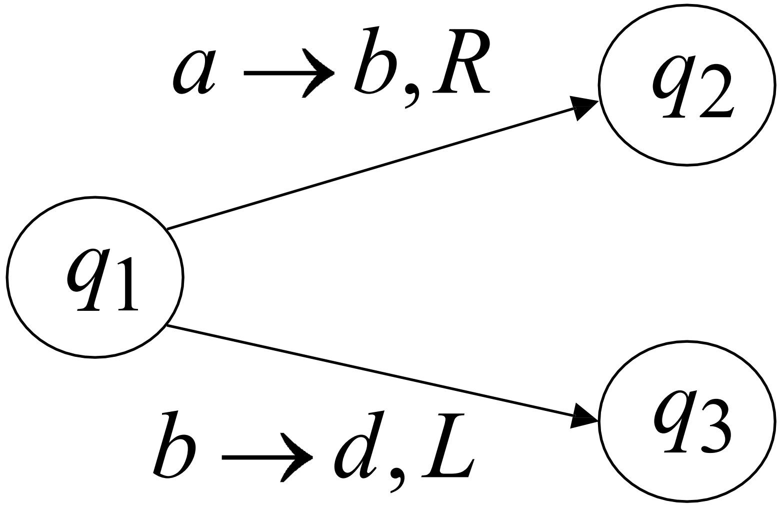
$$\begin{tikzcd} & q_2 \\ q_1 \arrow[ur, "a \rightarrow b, R"] \arrow[dr, "a \rightarrow d, L"'] & \\ & q_3 \end{tikzcd}$$

No lambda transitions allowed

Partial Transition Function

Example:





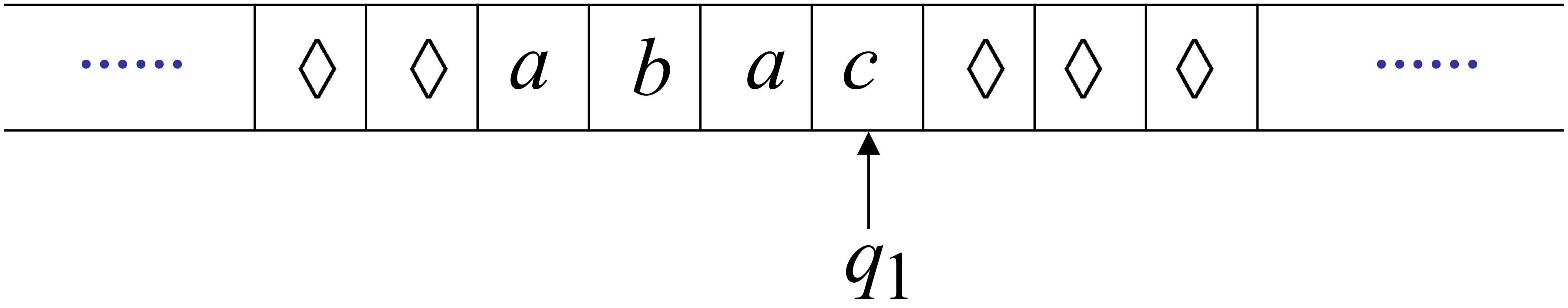
Allowed:

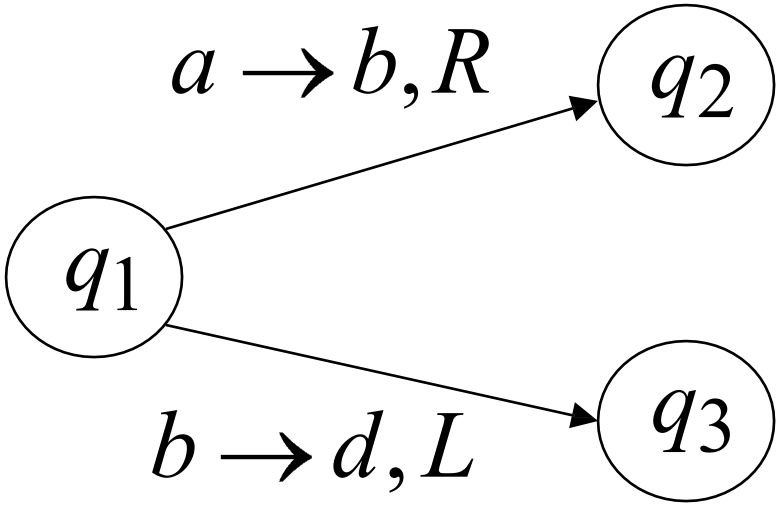
No transition for input symbol c

Halting

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

Example:

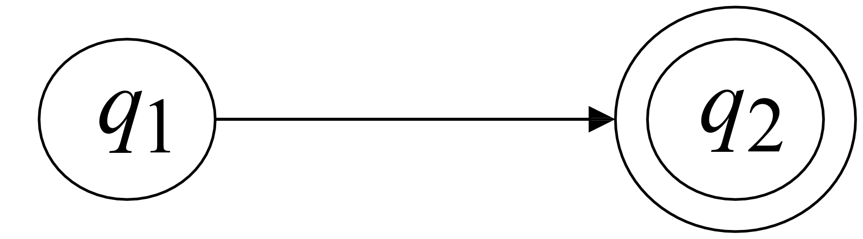




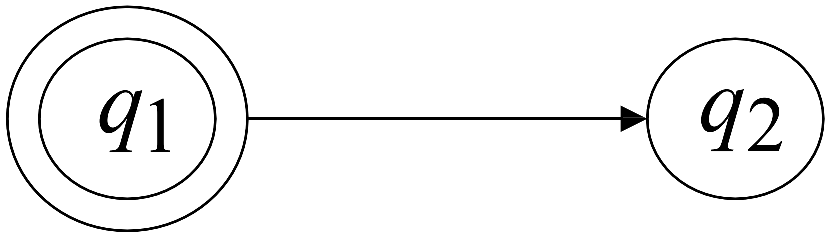
No possible transition

HALT!!!

Final States



Allowed



Not Allowed

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

Turing Machines as Acceptors

- A Turing machine *halts* when it no longer has any available moves. If it halts in a final state, it accepts its input; otherwise, it rejects its input.This is too easy, so let's repeat it in symbols:
- A Turing Machine T = (Q, , , , q_0, #, F) accepts a language L(M), where L(M) = (w +: q_0w $x_i q_f x_j$ for some q_f F, x_i, x_j *}(Notice that this definition assumes that the Turing machine starts with its tape head positioned on the leftmost symbol.)

Turing Machines as Acceptors

- We said a Turing machine accepts its input if it halts in a final state. There are two ways this could fail to happen:
- The Turing machine could halt in a nonfinal state, or The Turing machine could never stop (in which case we say it is in an *infinite loop.*)
- If a Turing machine halts, the sequence of configurations leading to the halt state is called a *computation.*

Acceptance

Accept Input $\Rightarrow$ If machine halts in a final state

Reject Input $\Rightarrow$ If machine halts in a non-final state

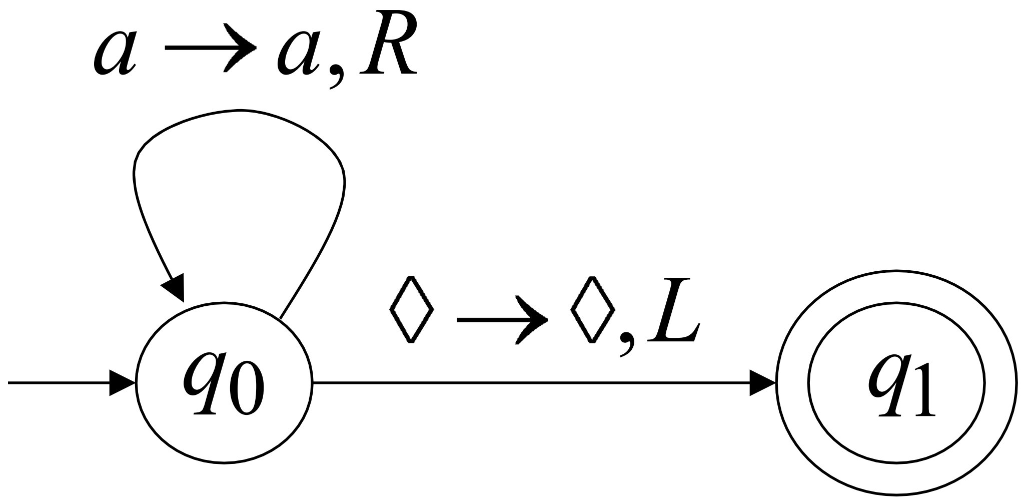
or

If machine enters an *infinite loop*

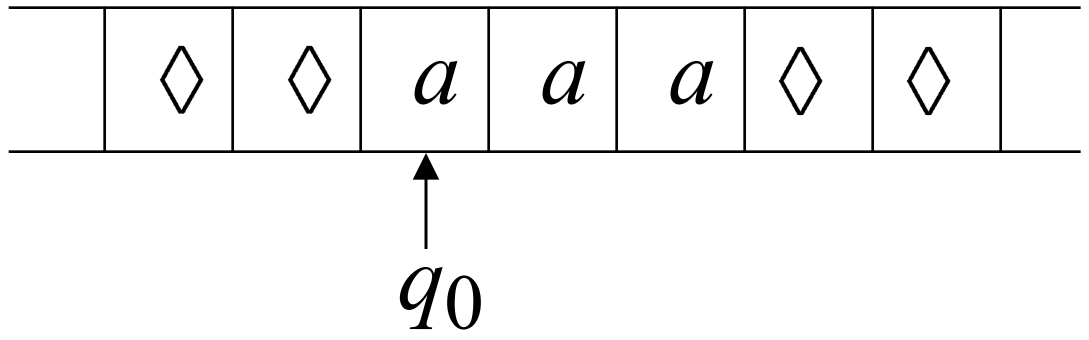
Turing Machine Example

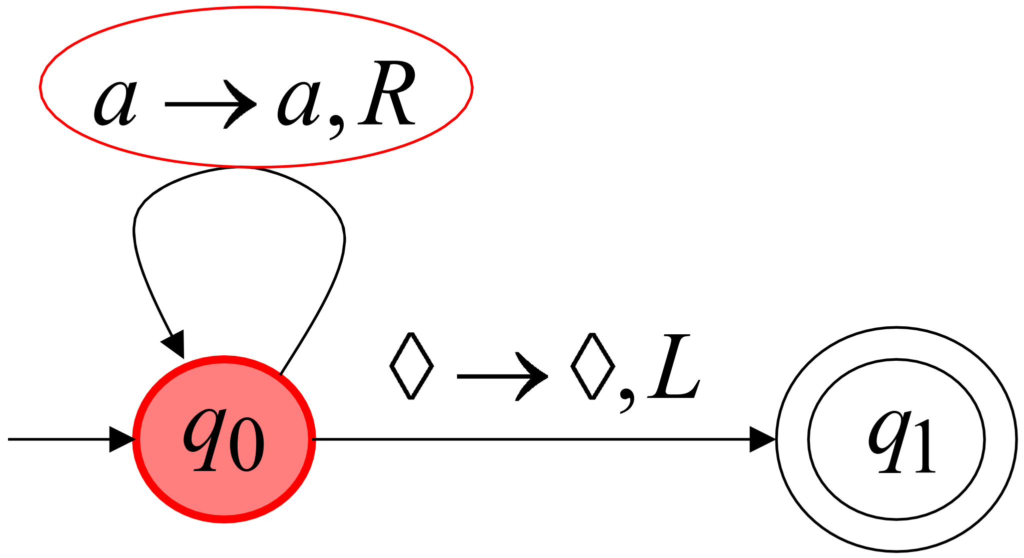
A Turing machine that accepts the language:

$$aa^*$$

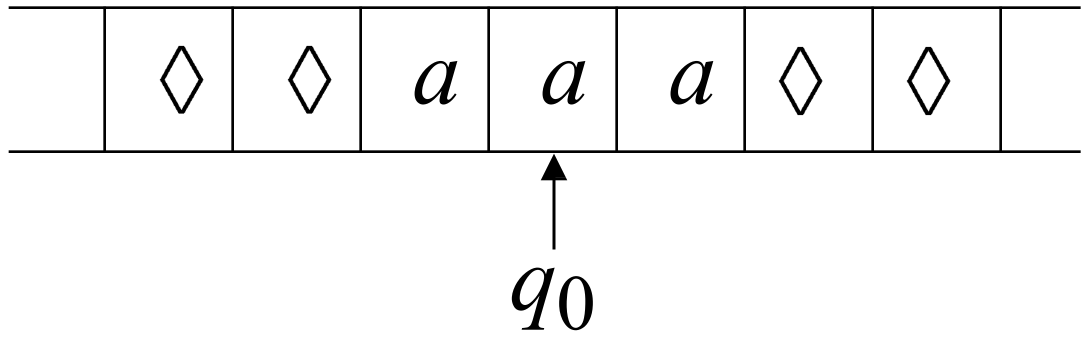


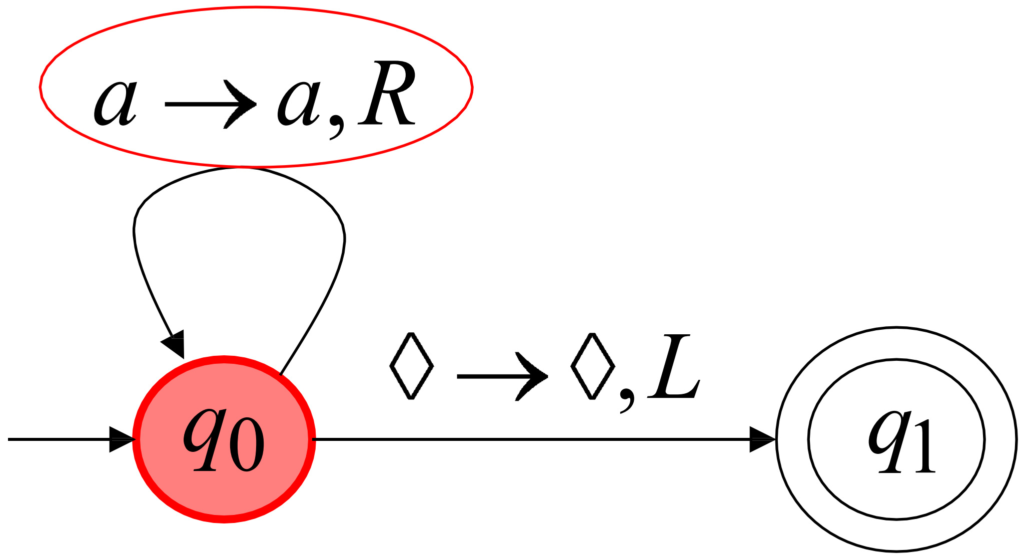
Time 0



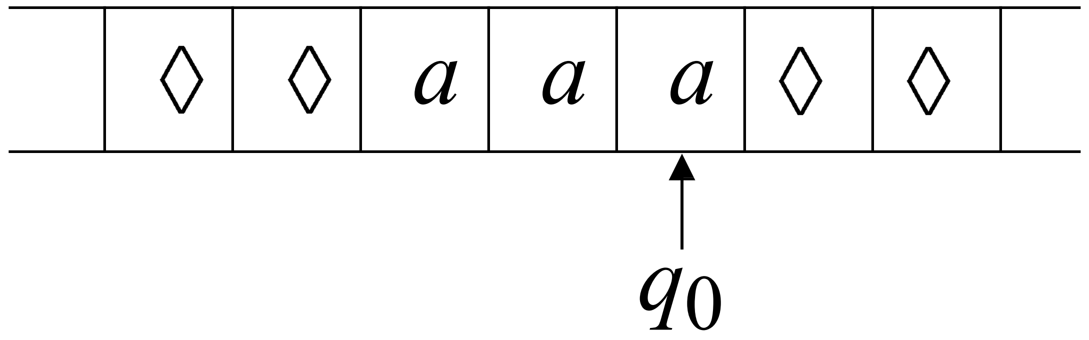


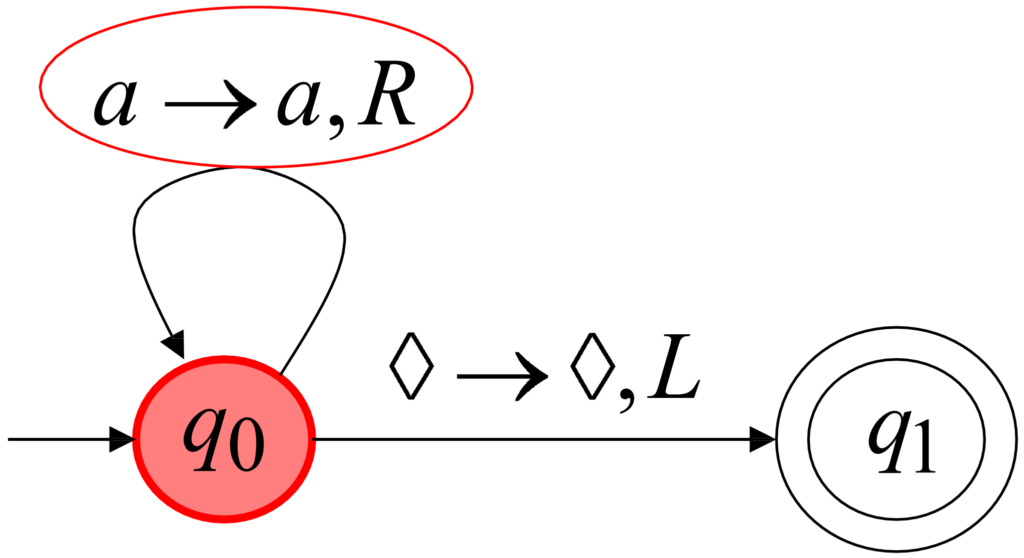
Time 1



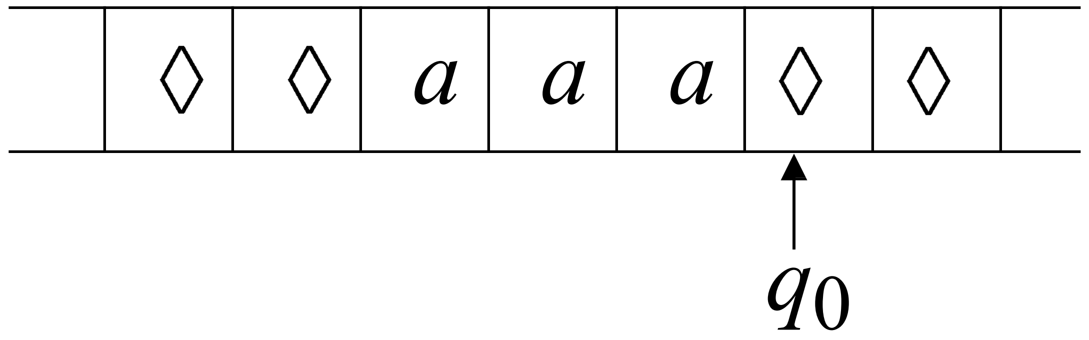


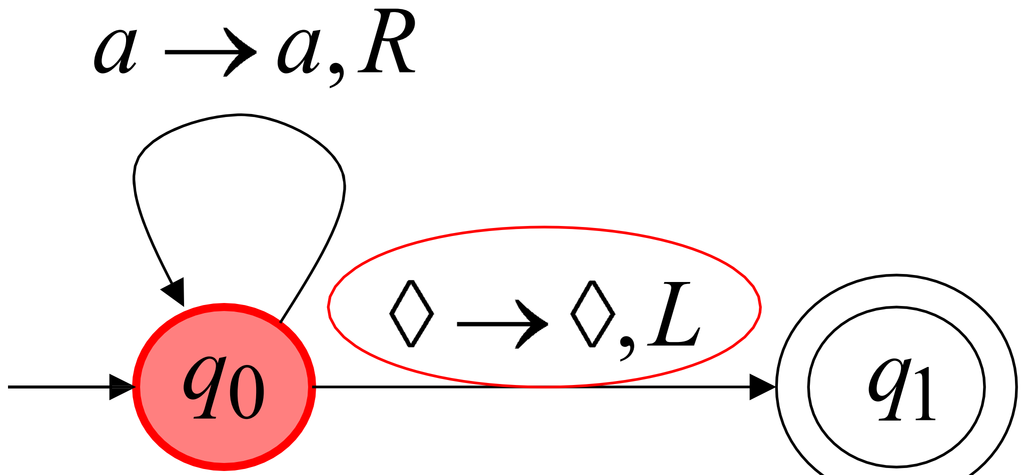
Time 2



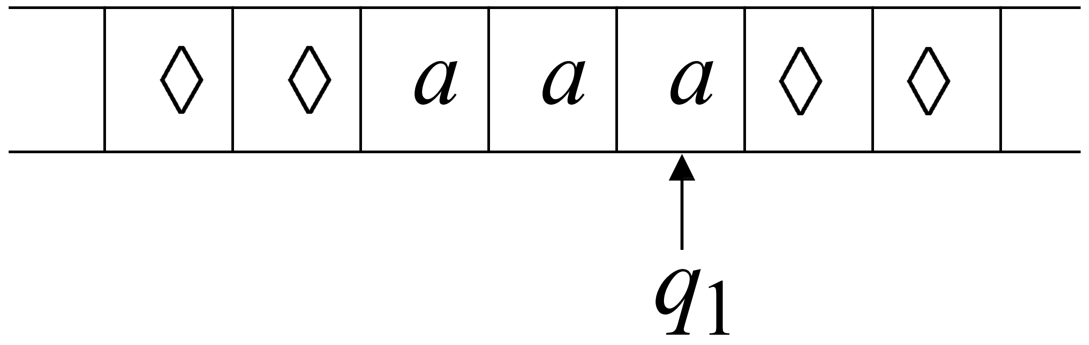


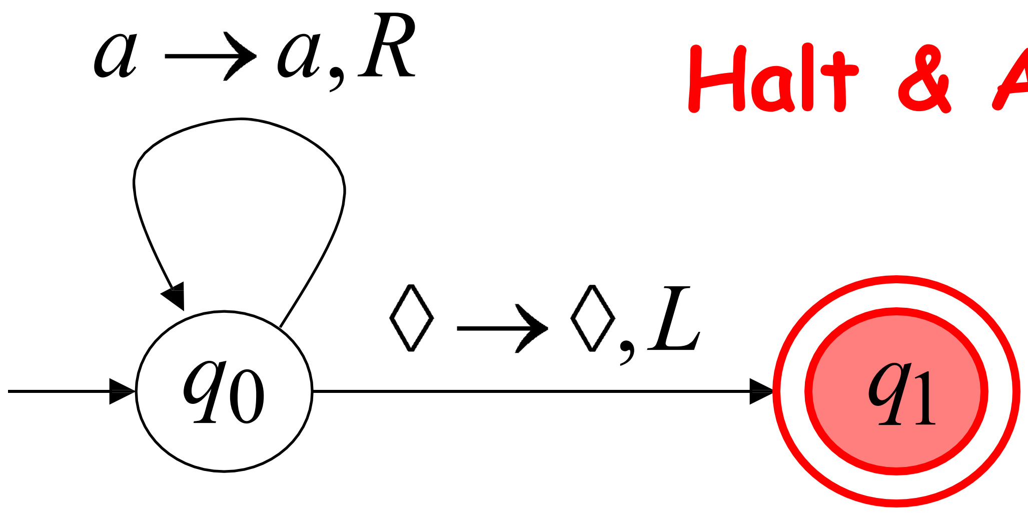
Time 3





Time 4

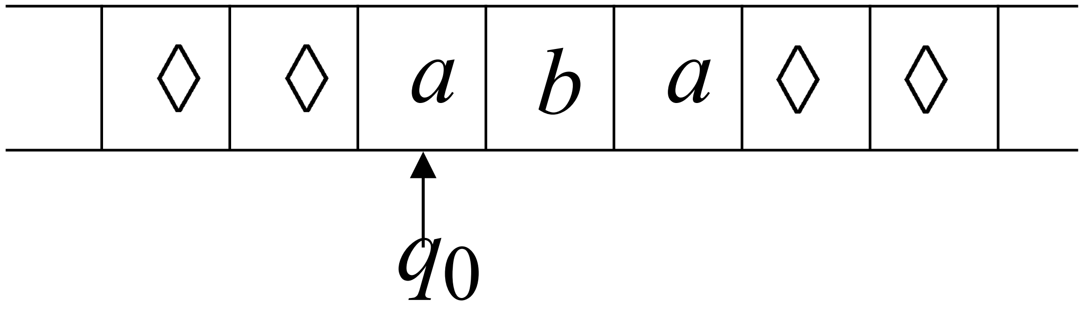


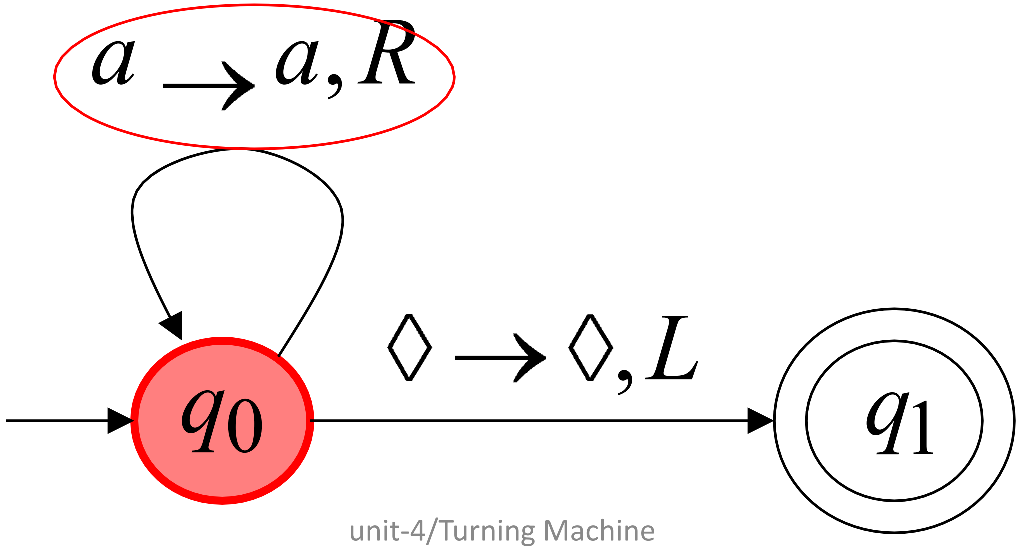


Halt & Accept

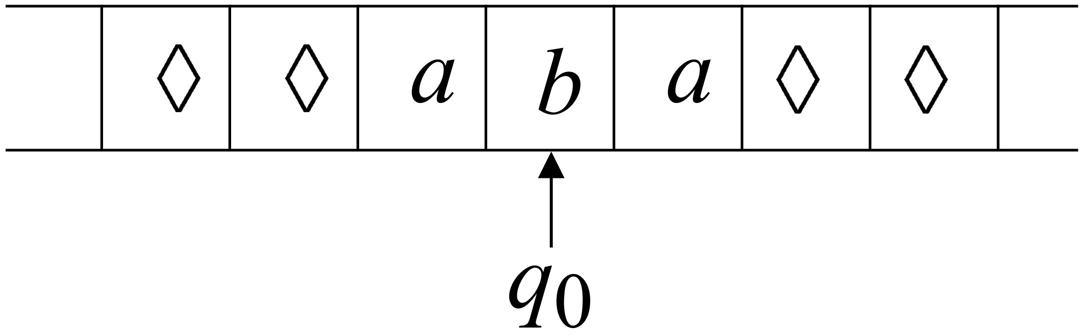
Rejection Example

Time 0



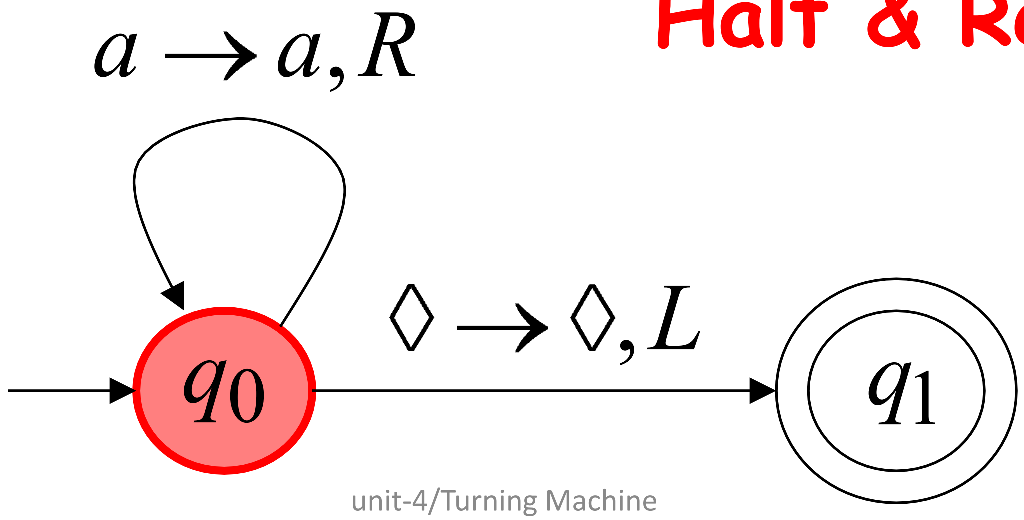


Time 1

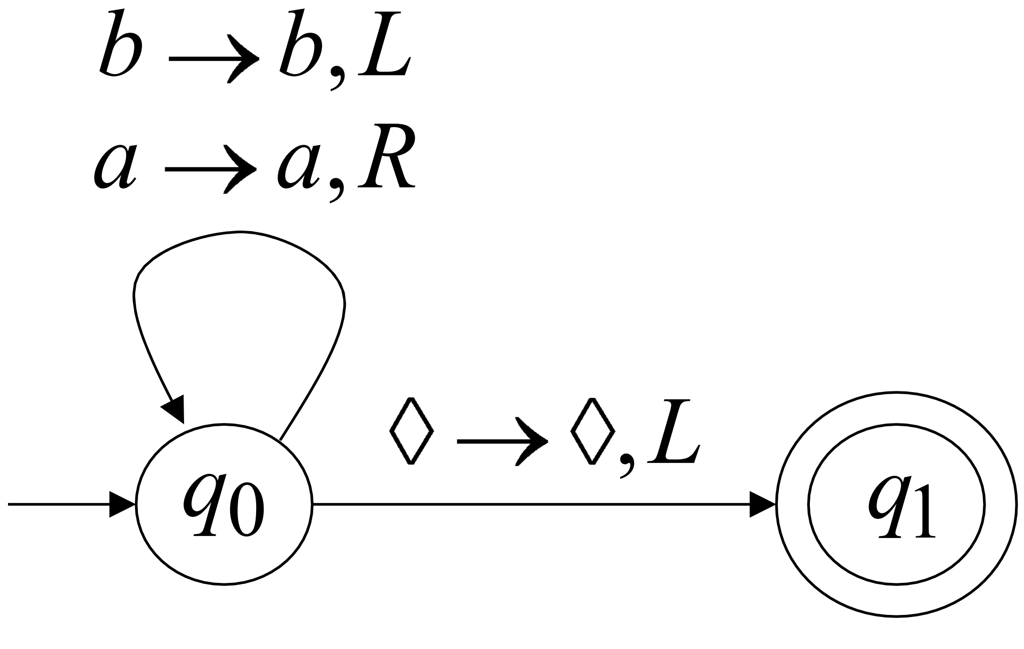


No possible Transition

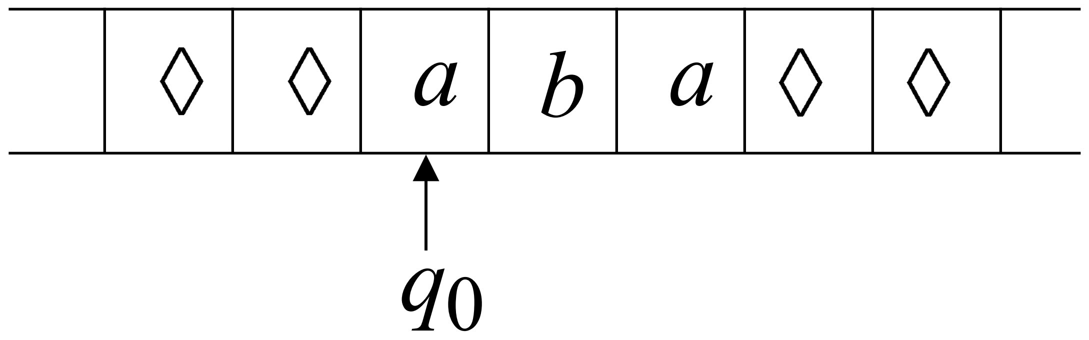
Halt & Reject

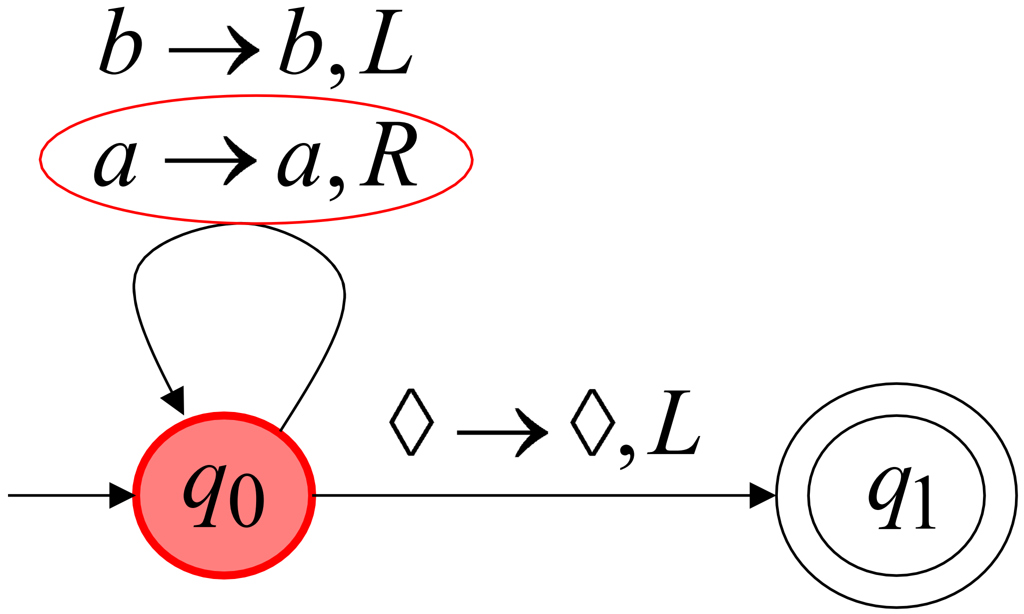


Infinite Loop Example

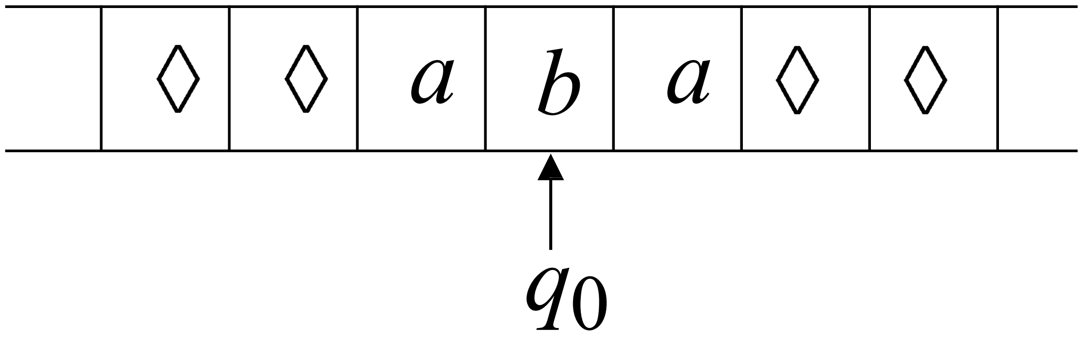

$$\begin{tikzcd} \arrow[r] & q_0 \arrow[loop above, "\begin{array}{c} b \rightarrow b, L \\ a \rightarrow a, R \end{array}"] \arrow[r, "\Diamond \rightarrow \Diamond, L"] & q_1 \end{tikzcd}$$

Time 0

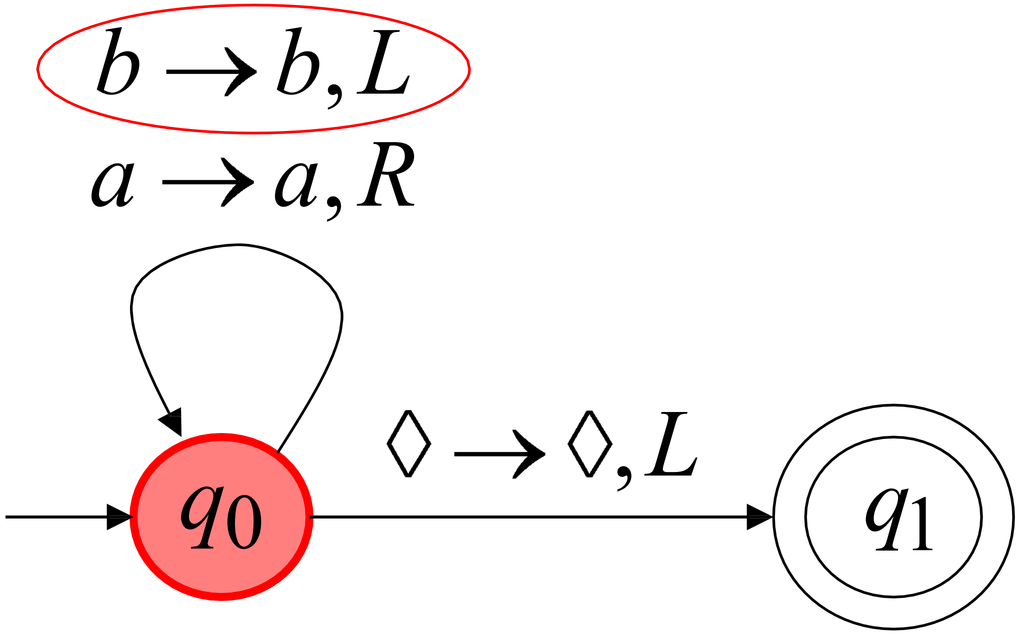




Time 1

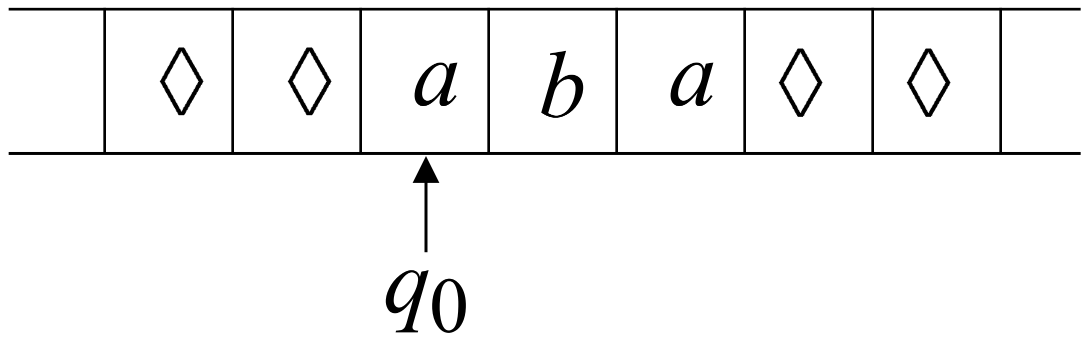


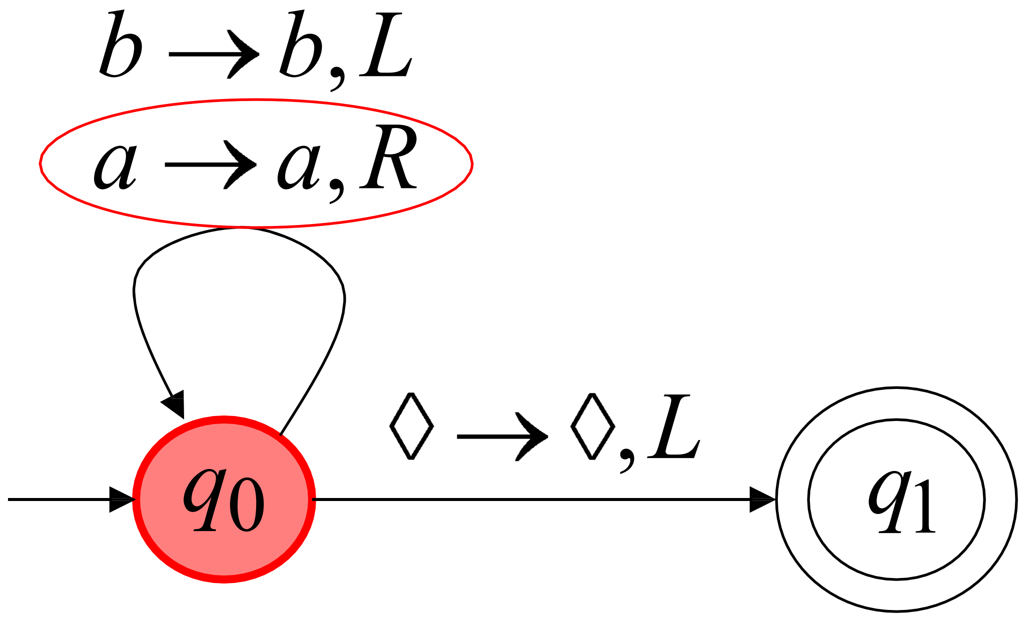
◊
◊
a
b
a
◊
◊
q_0



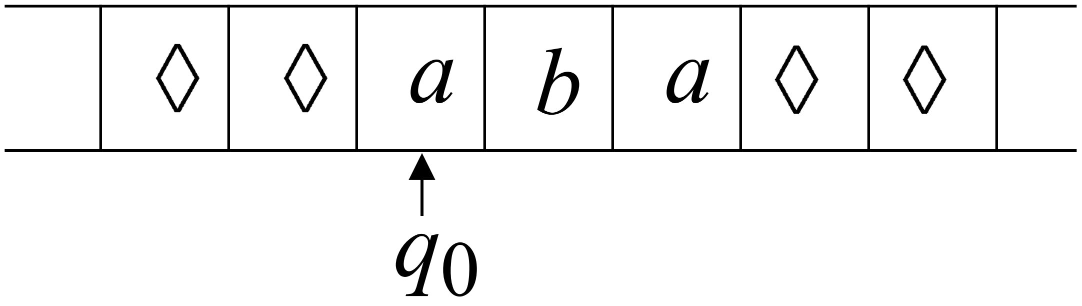
$b \rightarrow b, L$
$a \rightarrow a, R$
$\lozenge \rightarrow \lozenge, L$
q_0
q_1

Time 2

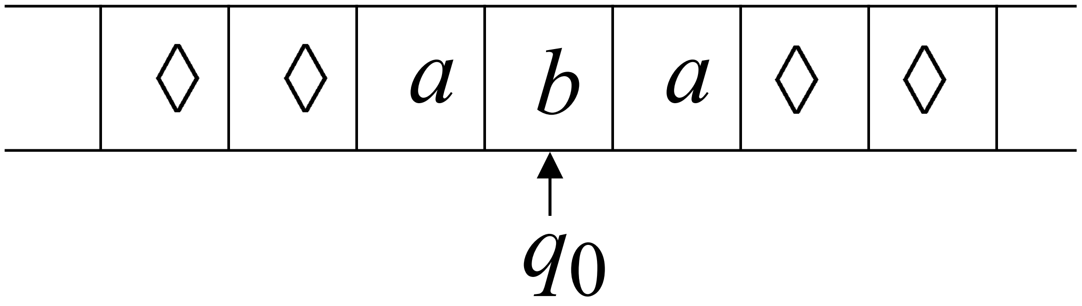




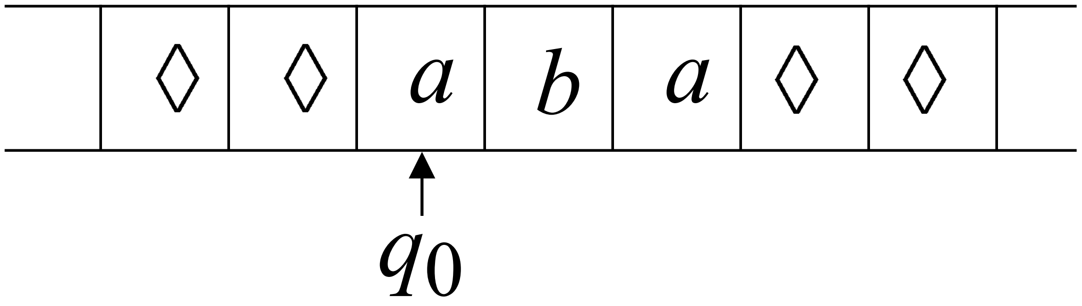
Time 2



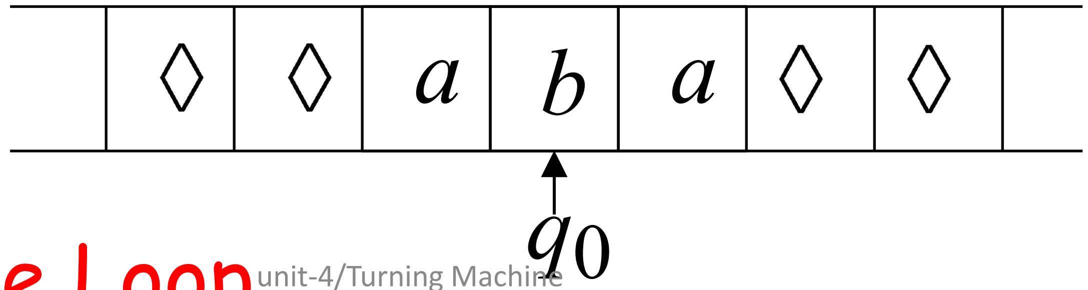
Time 3



Time 4



Time 5



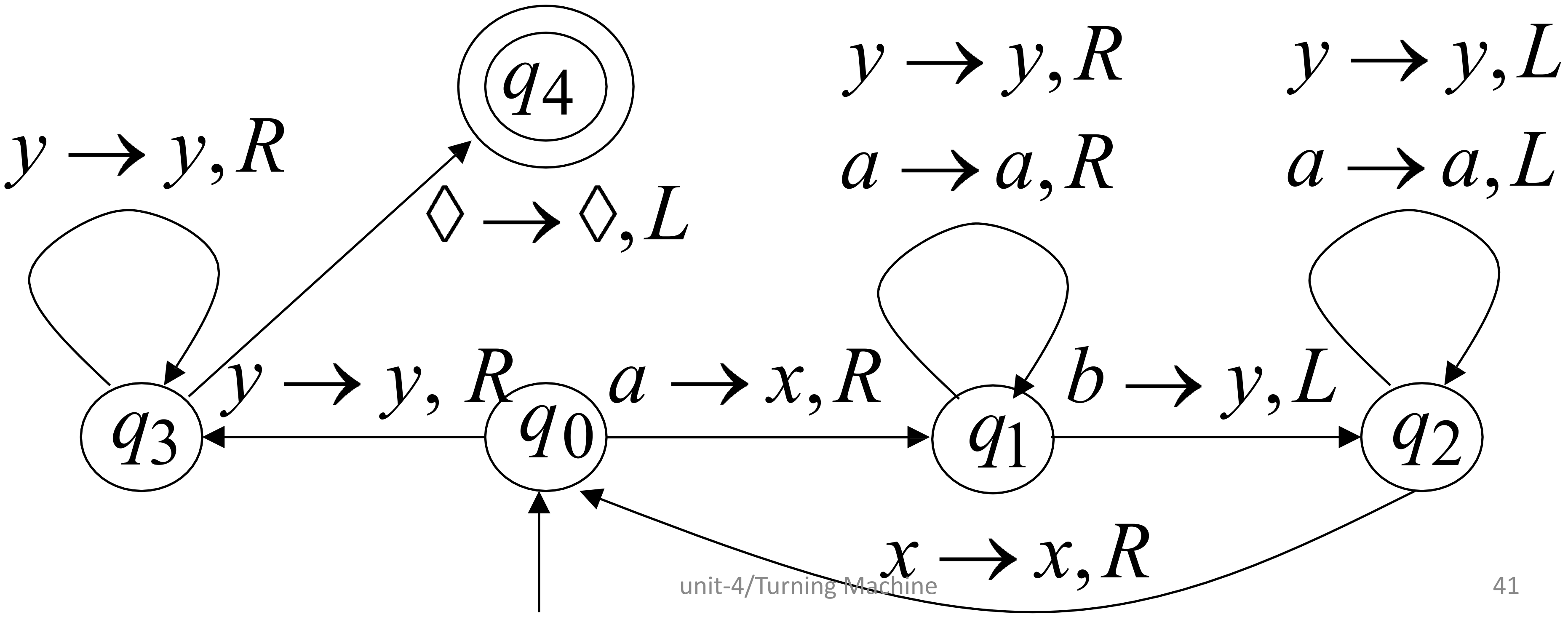
… Infinite Loop

Because of the infinite loop:

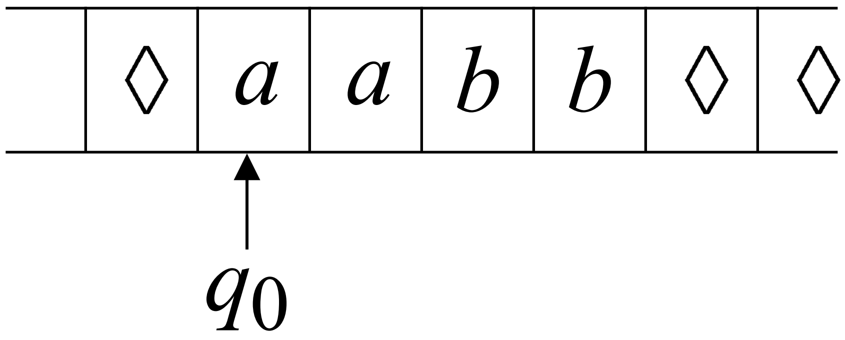
- The final state cannot be reached
- The machine never halts
- The input is not accepted

Another Turing Machine Example

Turing machine for the language $\{a^n b^n\}$

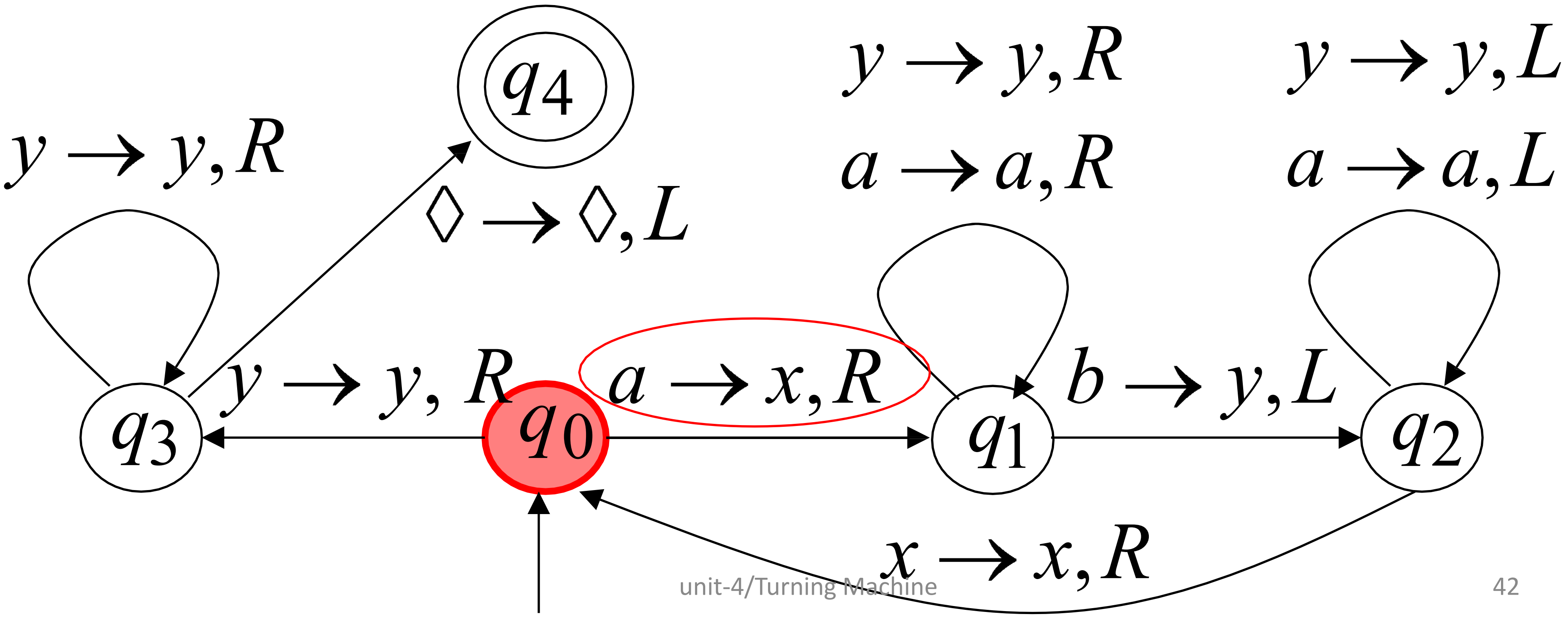


Time 0



$\Diamond$ a a b b $\Diamond$ $\Diamond$

q_0



q_4

$y \rightarrow y, R$

$\Diamond \rightarrow \Diamond, L$

$y \rightarrow y, R$

$a \rightarrow a, R$

$y \rightarrow y, L$

$a \rightarrow a, L$

$y \rightarrow y, R$

$a \rightarrow x, R$

$b \rightarrow y, L$

q_3

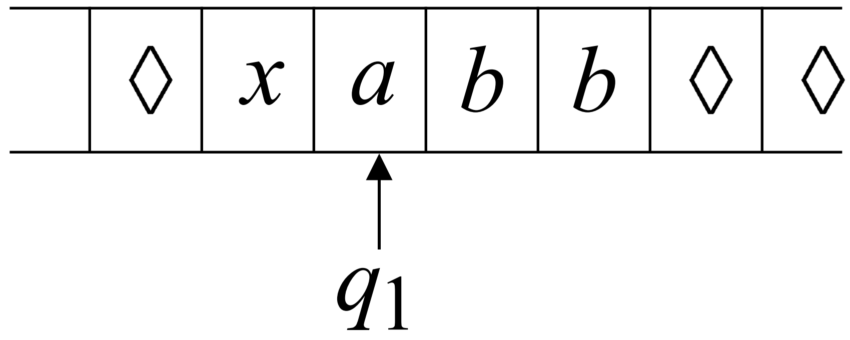
q_0

q_1

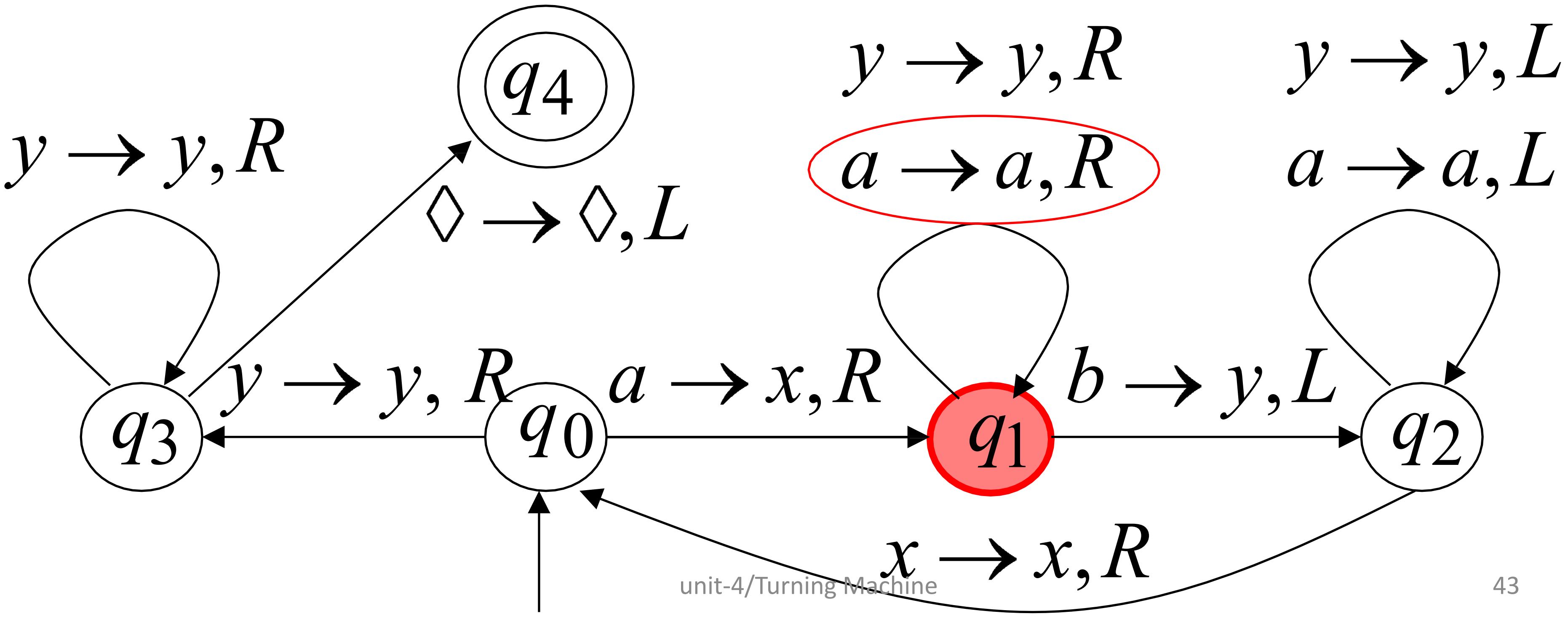
q_2

$x \rightarrow x, R$

Time 1

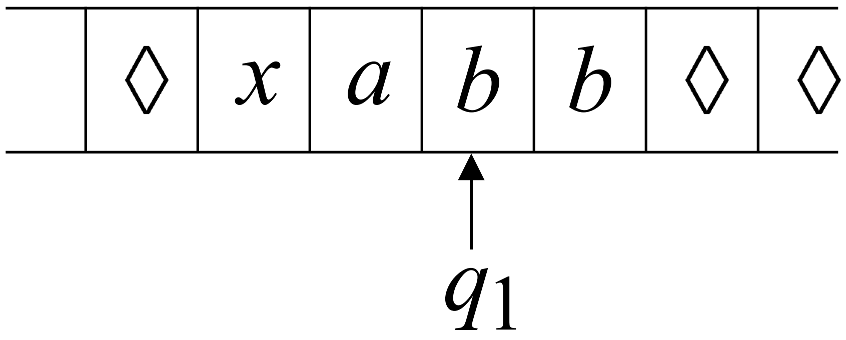


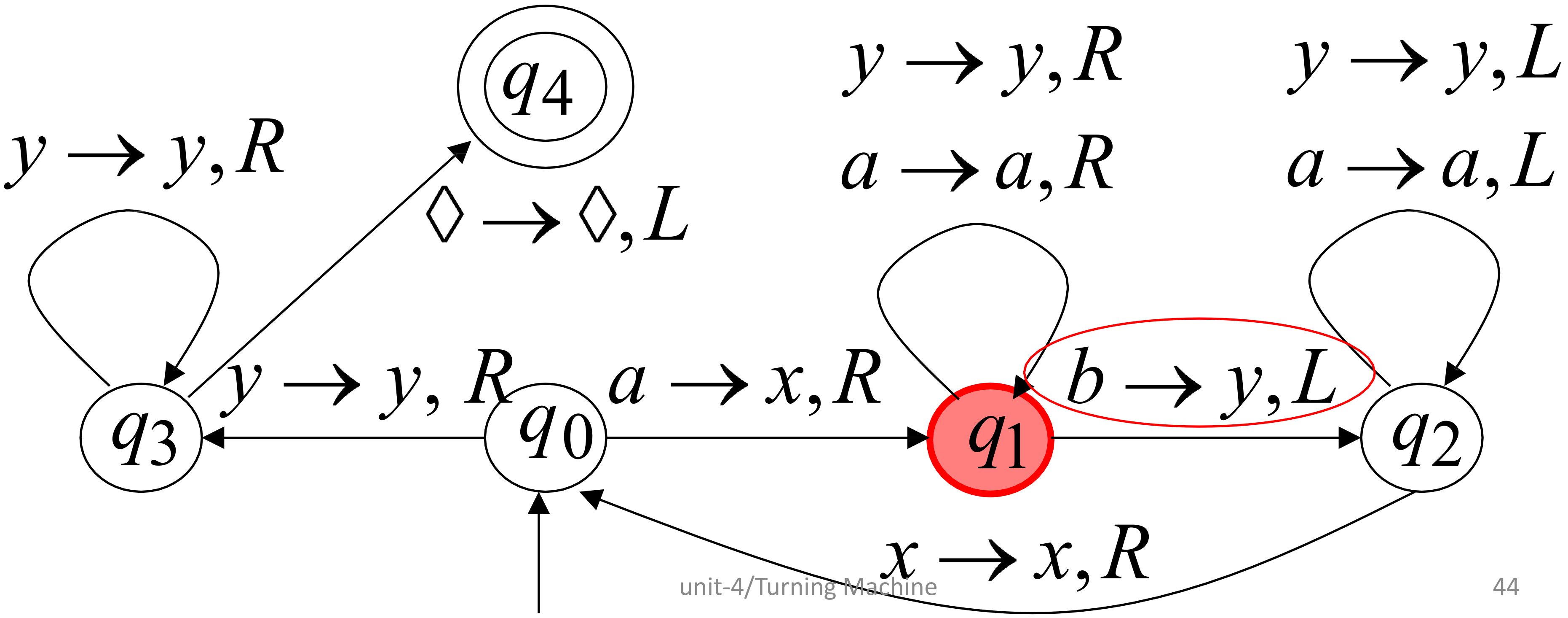
◊
x
a
b
b
◊
◊
q_1



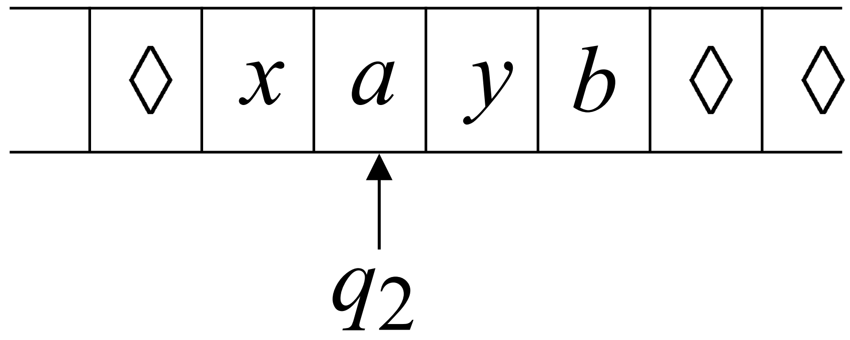
q_4
q_3
q_0
q_1
q_2
$y \rightarrow y, R$
$\lozenge \rightarrow \lozenge, L$
$y \rightarrow y, R$
$a \rightarrow x, R$
$y \rightarrow y, R$
$a \rightarrow a, R$
$b \rightarrow y, L$
$y \rightarrow y, L$
$a \rightarrow a, L$
$x \rightarrow x, R$

Time 2

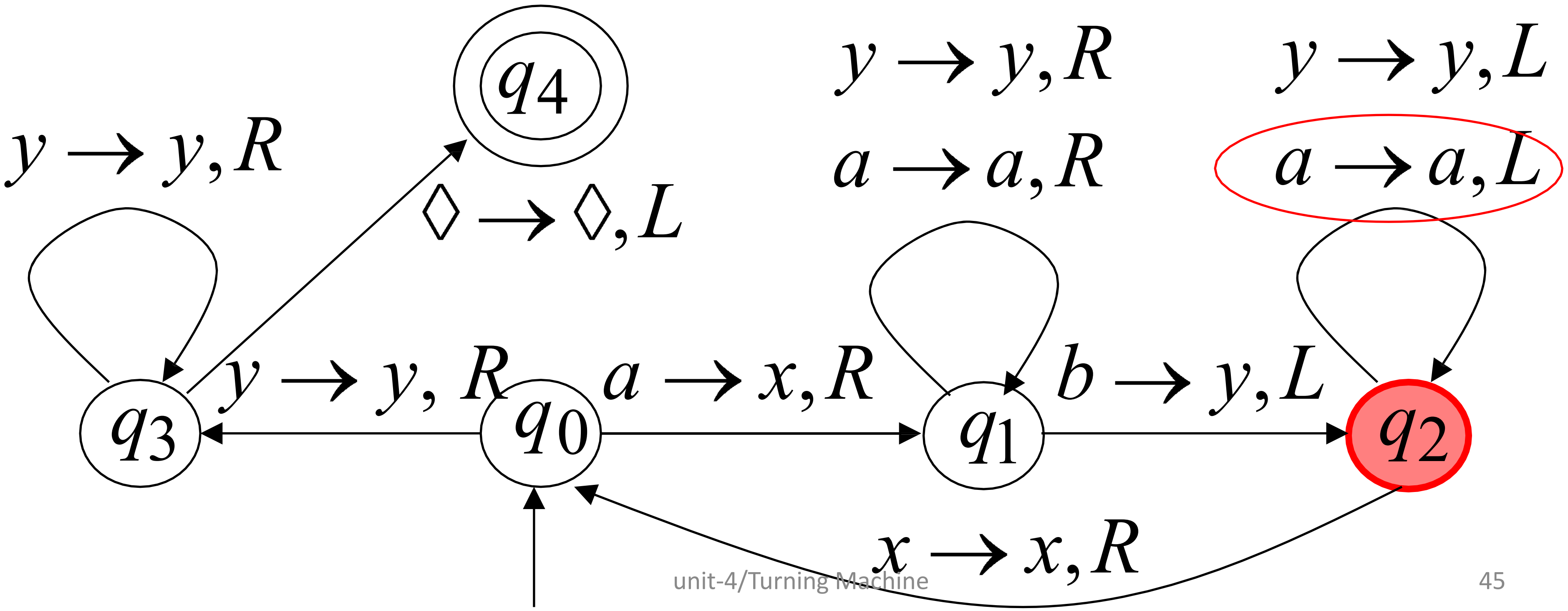

◊
x
a
b
b
◊
◊
q_1


q_4
$y \to y, R$
$y \to y, L$
$a \to a, R$
$a \to a, L$
$y \to y, R$
$◊ \to ◊, L$
$y \to y, R$
$a \to x, R$
$b \to y, L$
q_3
q_0
q_1
q_2
$x \to x, R$

Time 3

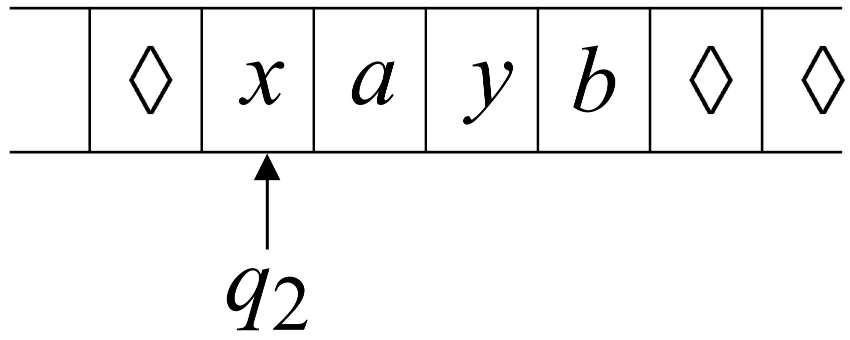


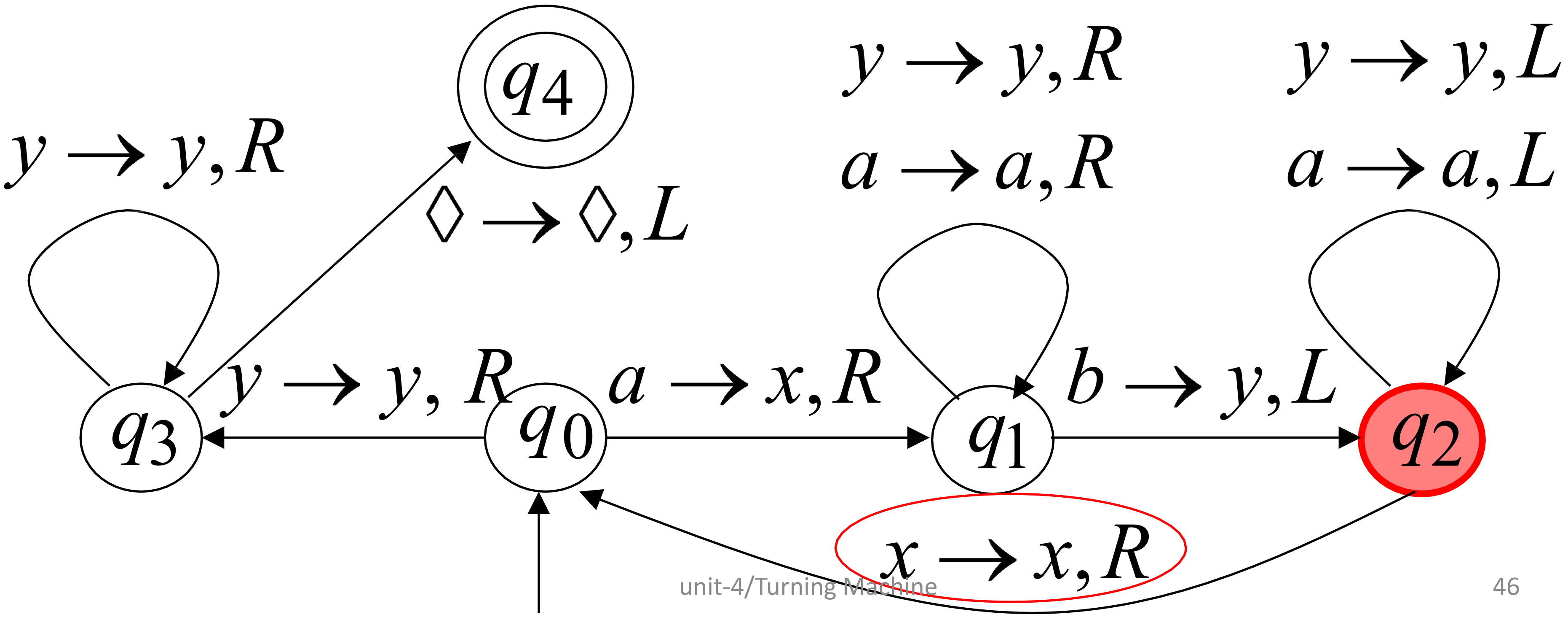
◊
x
a
y
b
◊
◊
q_2



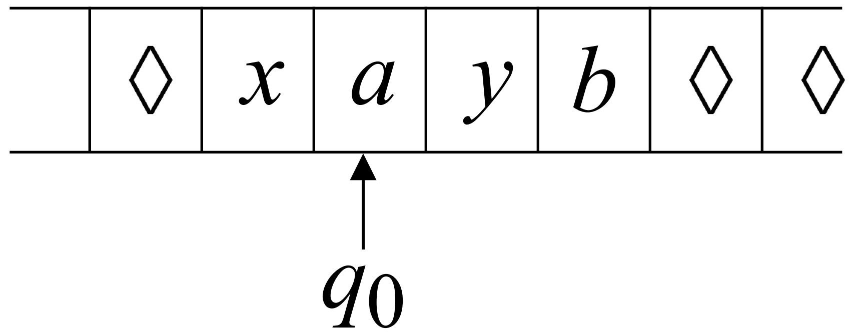
q_4
$y \rightarrow y, R$
$\lozenge \rightarrow \lozenge, L$
$y \rightarrow y, R$
$a \rightarrow a, R$
$y \rightarrow y, L$
$a \rightarrow a, L$
$y \rightarrow y, R$
$a \rightarrow x, R$
$b \rightarrow y, L$
q_3
q_0
q_1
q_2
$x \rightarrow x, R$

Time 4

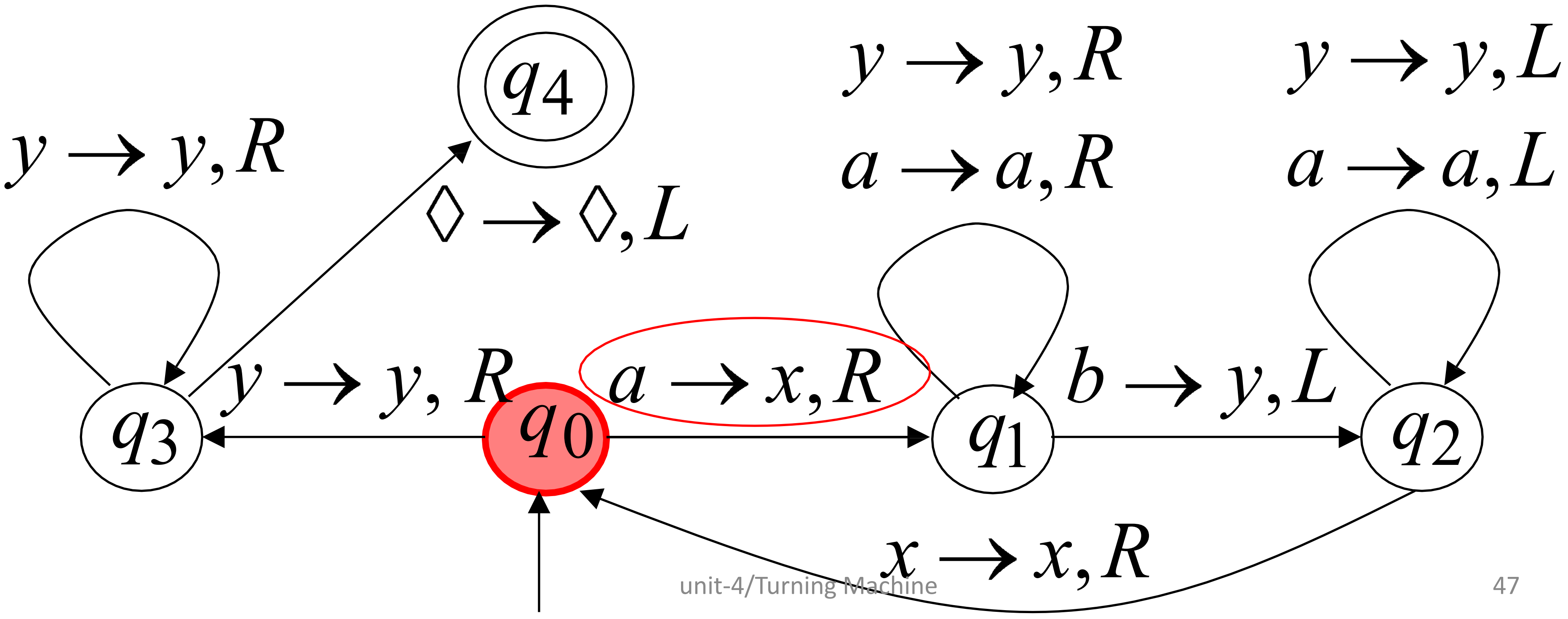

◊
x
a
y
b
◊
◊
q_2


q_4
$y \rightarrow y, R$
$\lozenge \rightarrow \lozenge, L$
$y \rightarrow y, R$
$a \rightarrow a, R$
$y \rightarrow y, L$
$a \rightarrow a, L$
$y \rightarrow y, R$
$a \rightarrow x, R$
$b \rightarrow y, L$
q_3
q_0
q_1
q_2
$x \rightarrow x, R$

Time 5

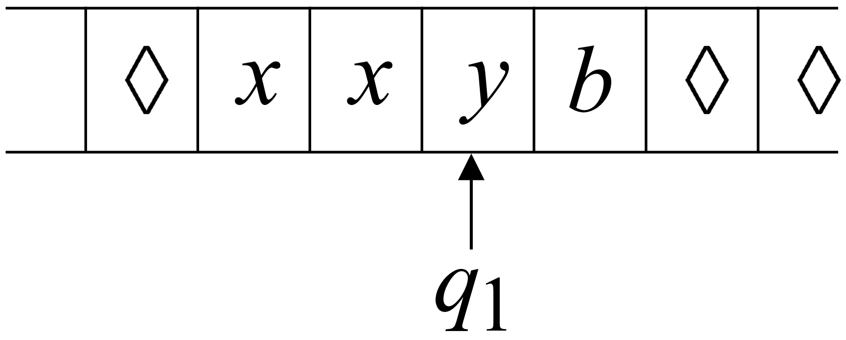


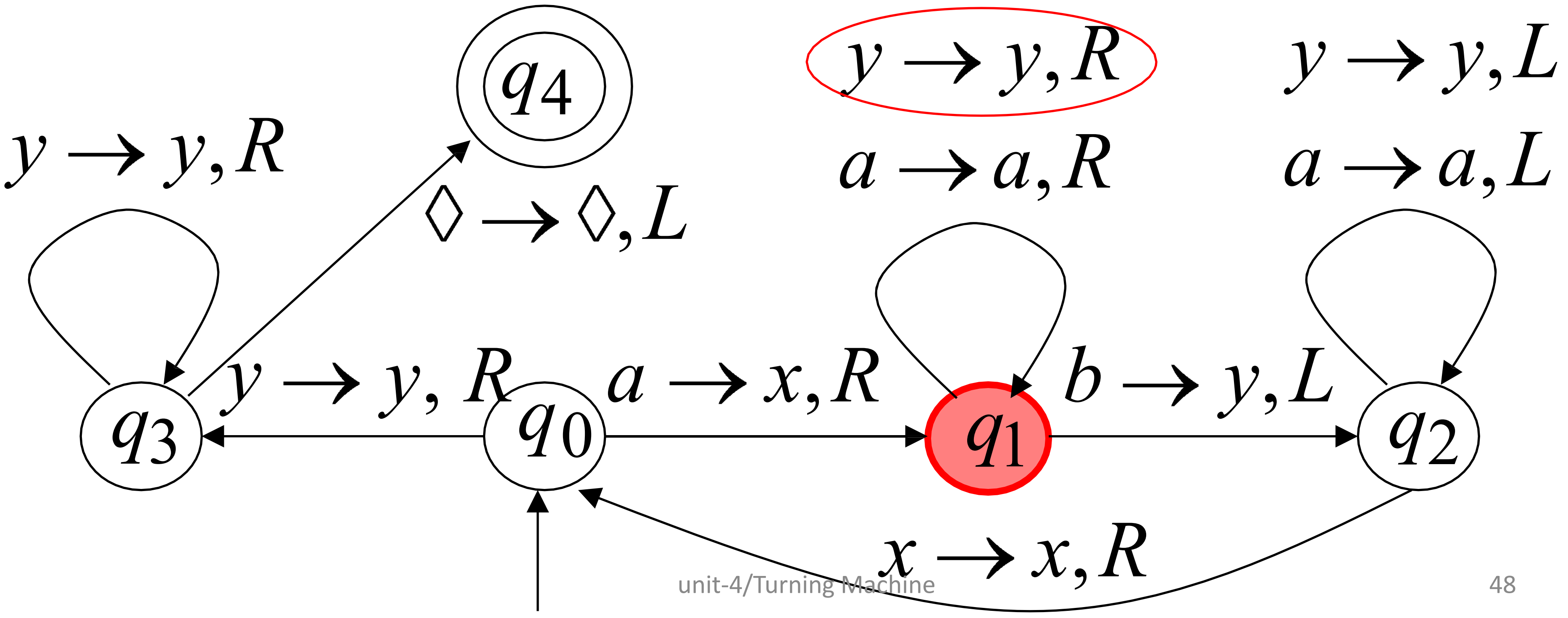
$\Diamond$
x
a
y
b
$\Diamond$
$\Diamond$
q_0



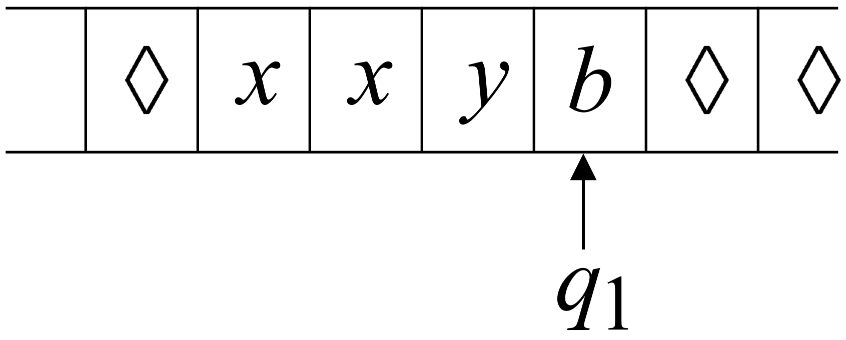
q_4
$\Diamond \rightarrow \Diamond, L$
$y \rightarrow y, R$
q_3
$y \rightarrow y, R$
q_0
$a \rightarrow x, R$
$y \rightarrow y, R$
$a \rightarrow a, R$
q_1
$b \rightarrow y, L$
$y \rightarrow y, L$
$a \rightarrow a, L$
q_2
$x \rightarrow x, R$

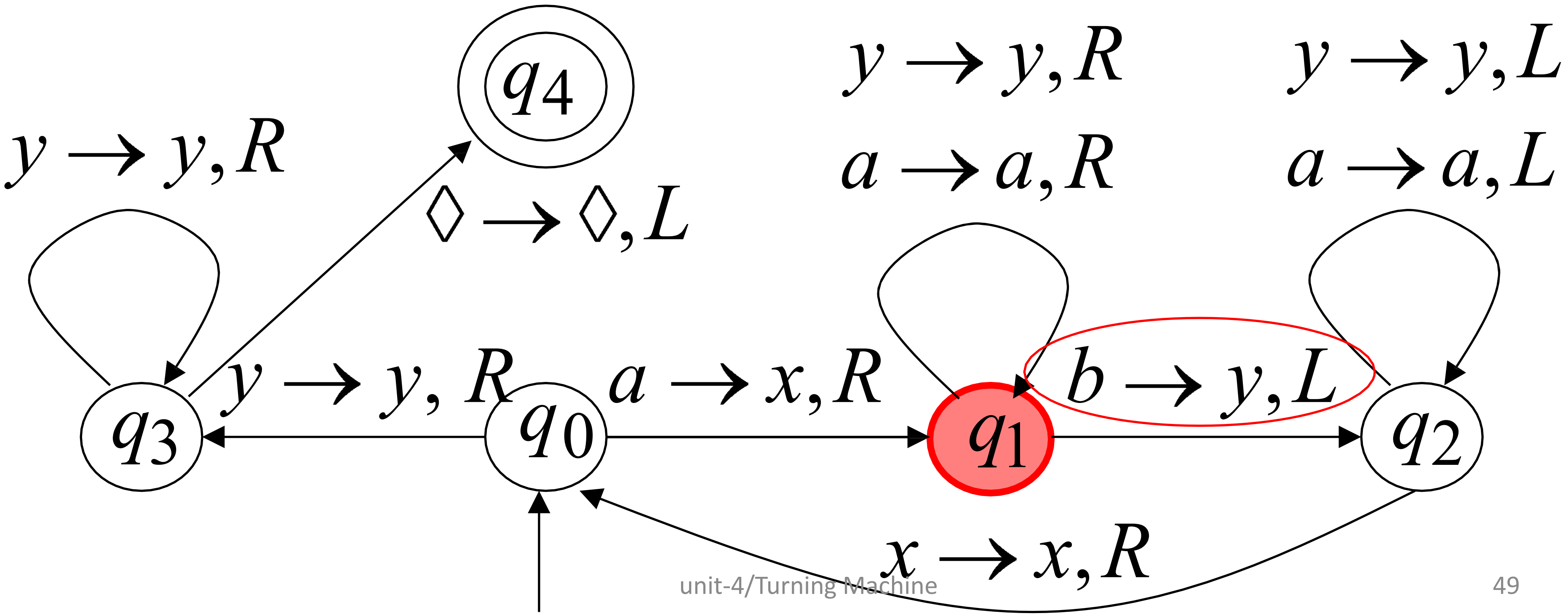
Time 6


◊
x
x
y
b
◊
◊
q_1

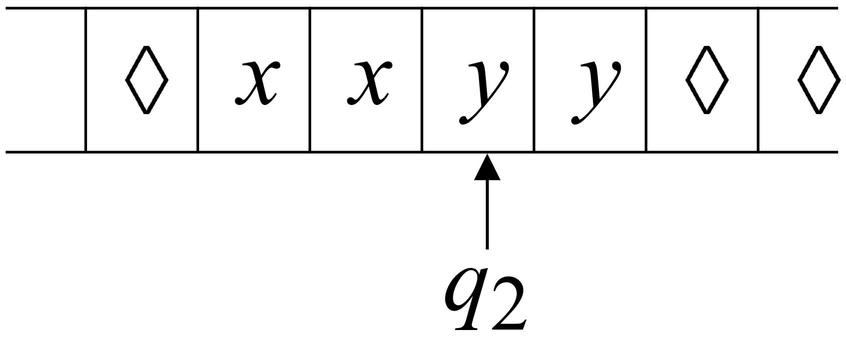

q_4
$y \rightarrow y, R$
$\Diamond \rightarrow \Diamond, L$
$y \rightarrow y, R$
$a \rightarrow a, R$
$y \rightarrow y, L$
$a \rightarrow a, L$
q_3
$y \rightarrow y, R$
q_0
$a \rightarrow x, R$
q_1
$b \rightarrow y, L$
q_2
$x \rightarrow x, R$

Time 7

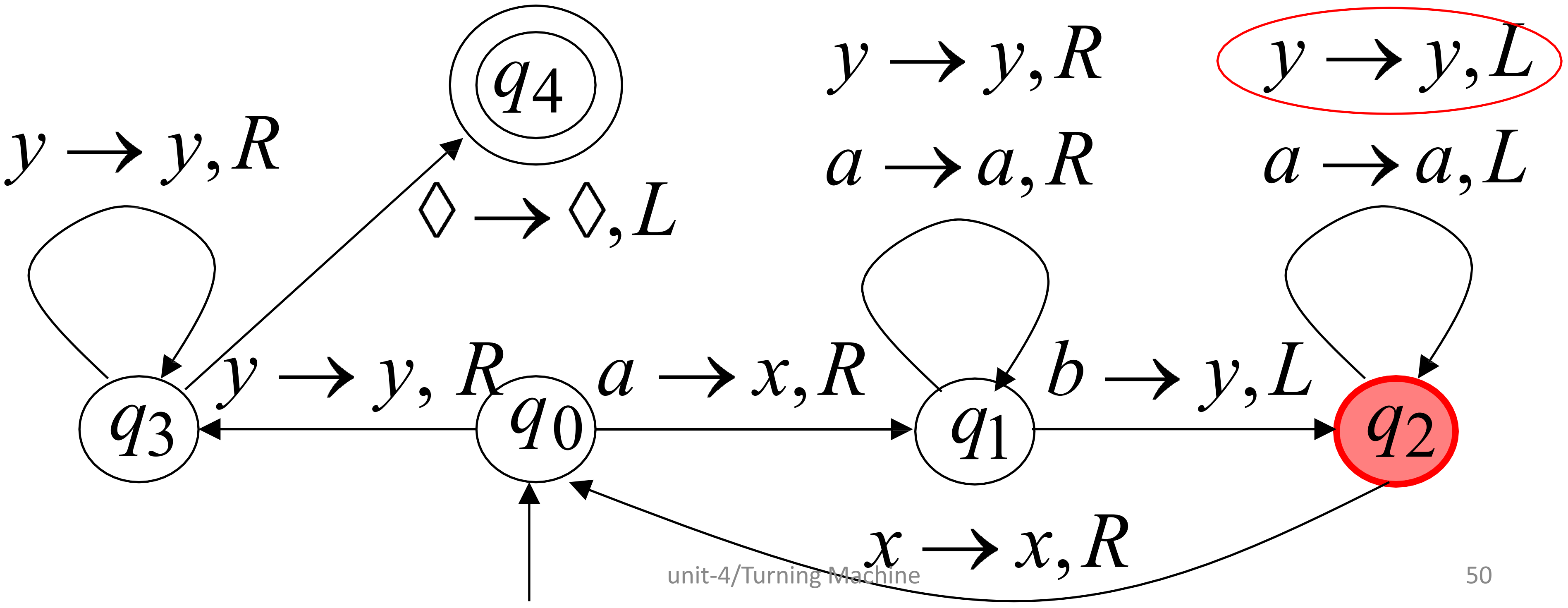

◊
x
x
y
b
◊
◊
q_1


q_4
q_3
q_0
q_1
q_2
$y \rightarrow y, R$
$\lozenge \rightarrow \lozenge, L$
$y \rightarrow y, R$
$a \rightarrow x, R$
$y \rightarrow y, R$
$a \rightarrow a, R$
$b \rightarrow y, L$
$y \rightarrow y, L$
$a \rightarrow a, L$
$x \rightarrow x, R$

Time 8

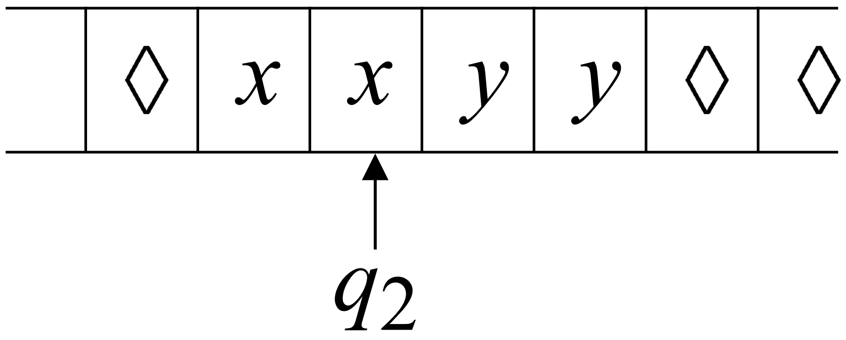


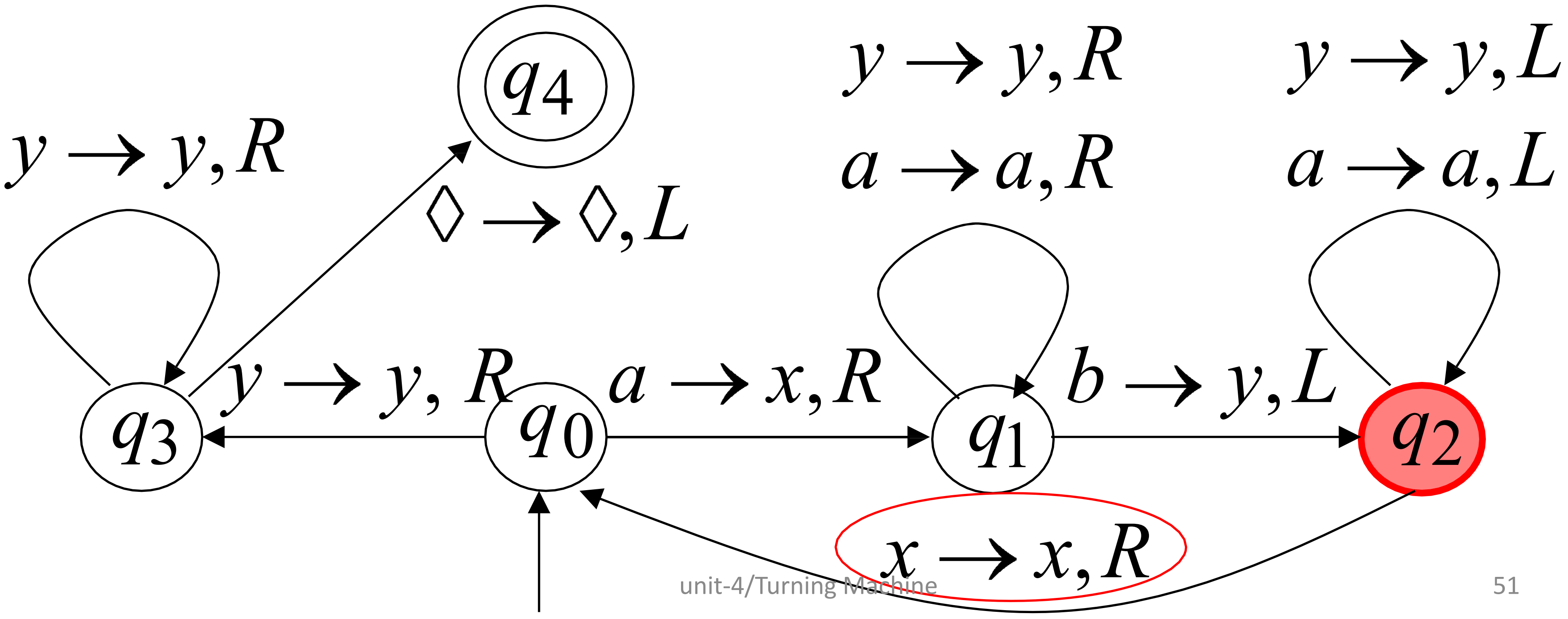
◊ x x y y ◊ ◊
q_2



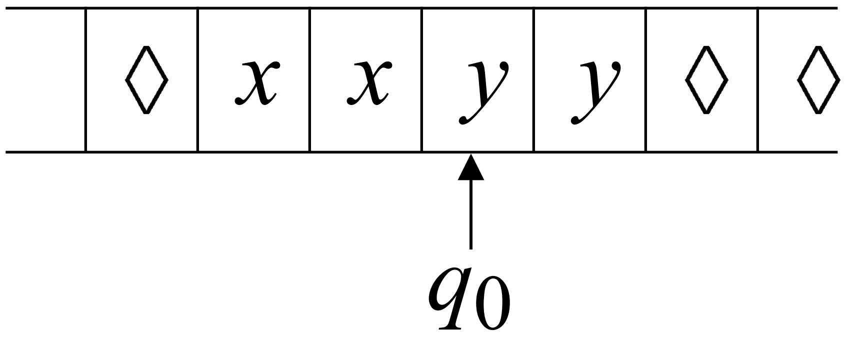
q_4
q_3
q_0
q_1
q_2
$y \rightarrow y, R$
$\Diamond \rightarrow \Diamond, L$
$y \rightarrow y, R$
$a \rightarrow a, R$
$y \rightarrow y, L$
$a \rightarrow a, L$
$y \rightarrow y, R$
$a \rightarrow x, R$
$b \rightarrow y, L$
$x \rightarrow x, R$

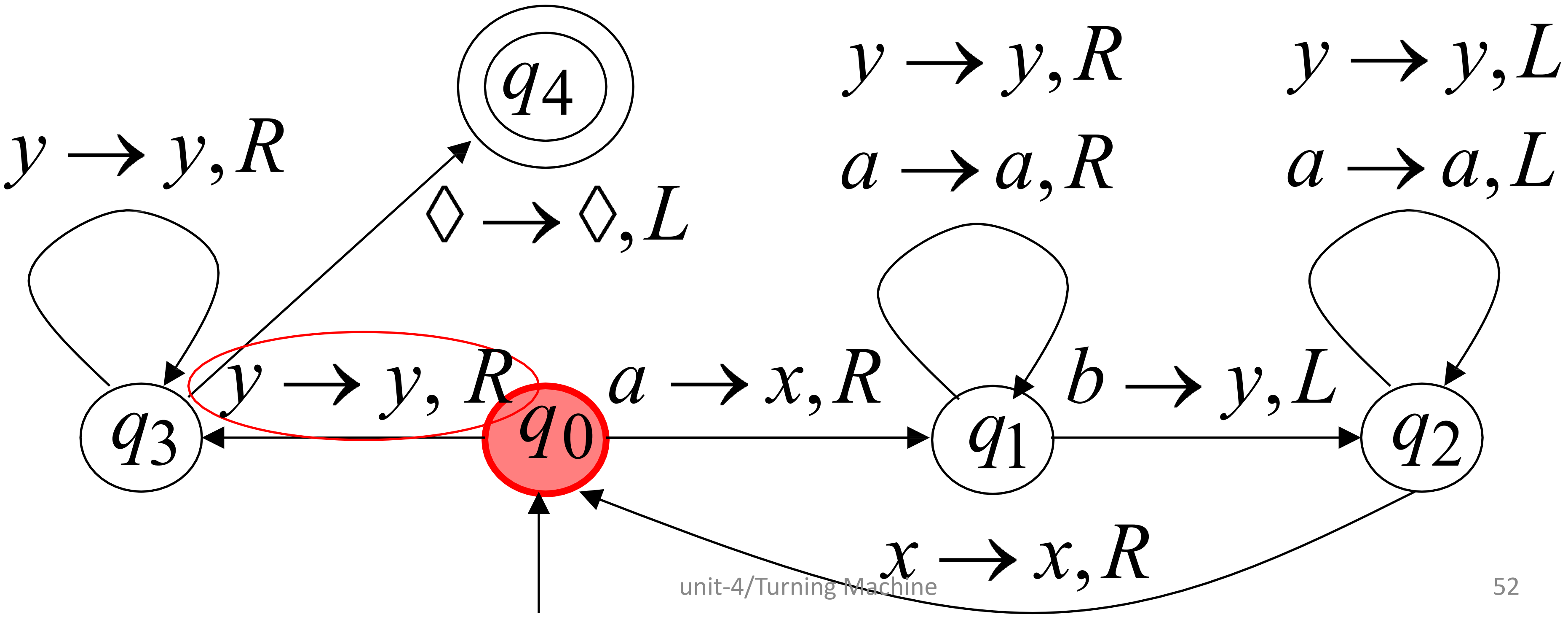
Time 9


◊
x
x
y
y
◊
◊
q_2

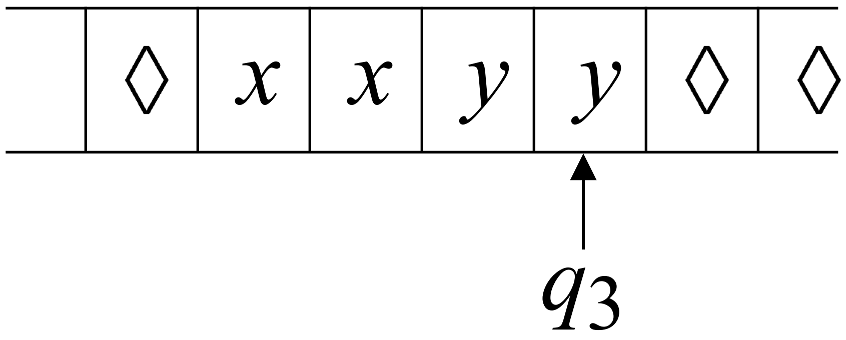

q_4
$\Diamond \rightarrow \Diamond, L$
$y \rightarrow y, R$
$y \rightarrow y, R$
$a \rightarrow a, R$
$y \rightarrow y, L$
$a \rightarrow a, L$
$y \rightarrow y, R$
$a \rightarrow x, R$
$b \rightarrow y, L$
q_3
q_0
q_1
q_2
$x \rightarrow x, R$

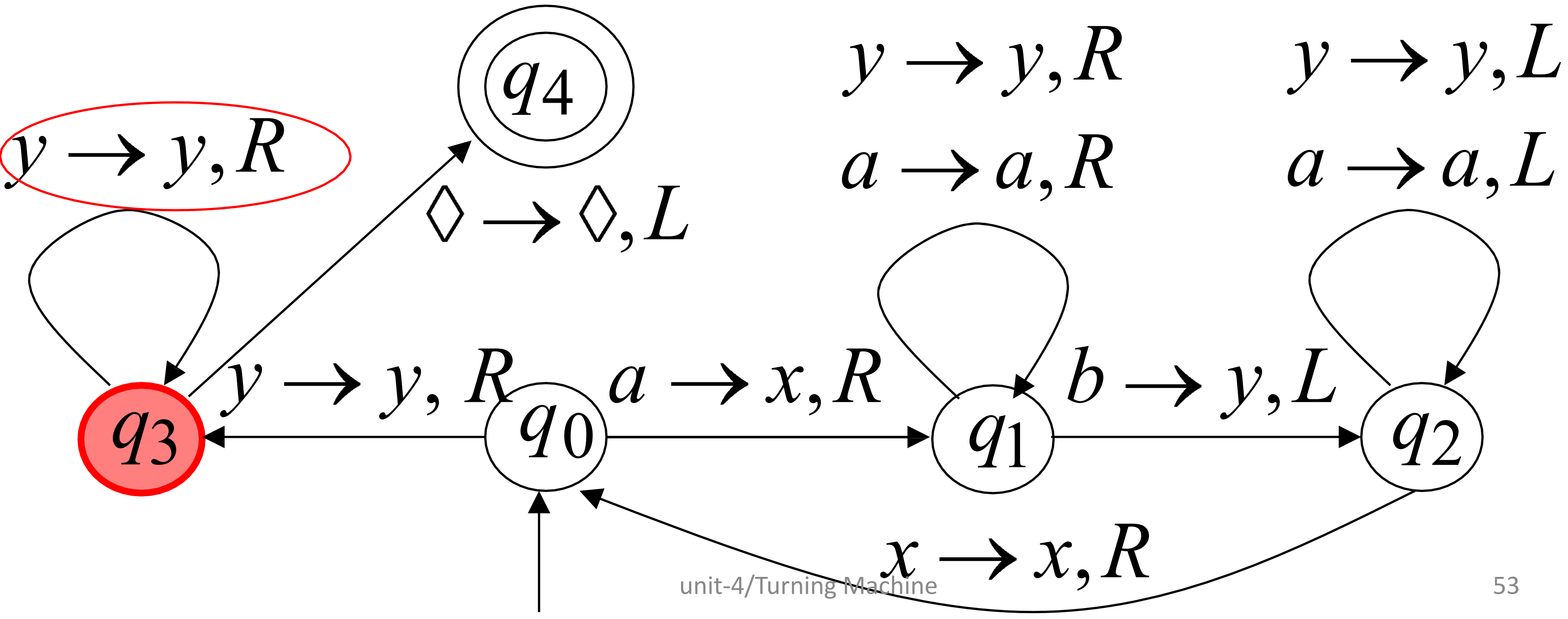
Time 10


◊
x
x
y
y
◊
◊
q_0

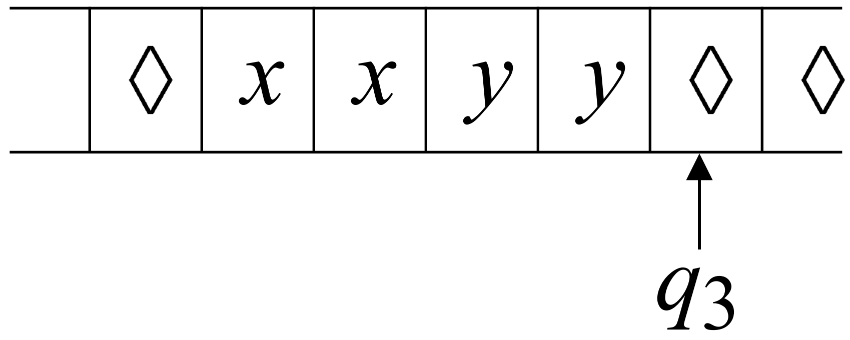

q_4
$y \rightarrow y, R$
$\lozenge \rightarrow \lozenge, L$
$y \rightarrow y, R$
$a \rightarrow a, R$
$y \rightarrow y, L$
$a \rightarrow a, L$
q_3
$y \rightarrow y, R$
q_0
$a \rightarrow x, R$
q_1
$b \rightarrow y, L$
q_2
$x \rightarrow x, R$

Time 11

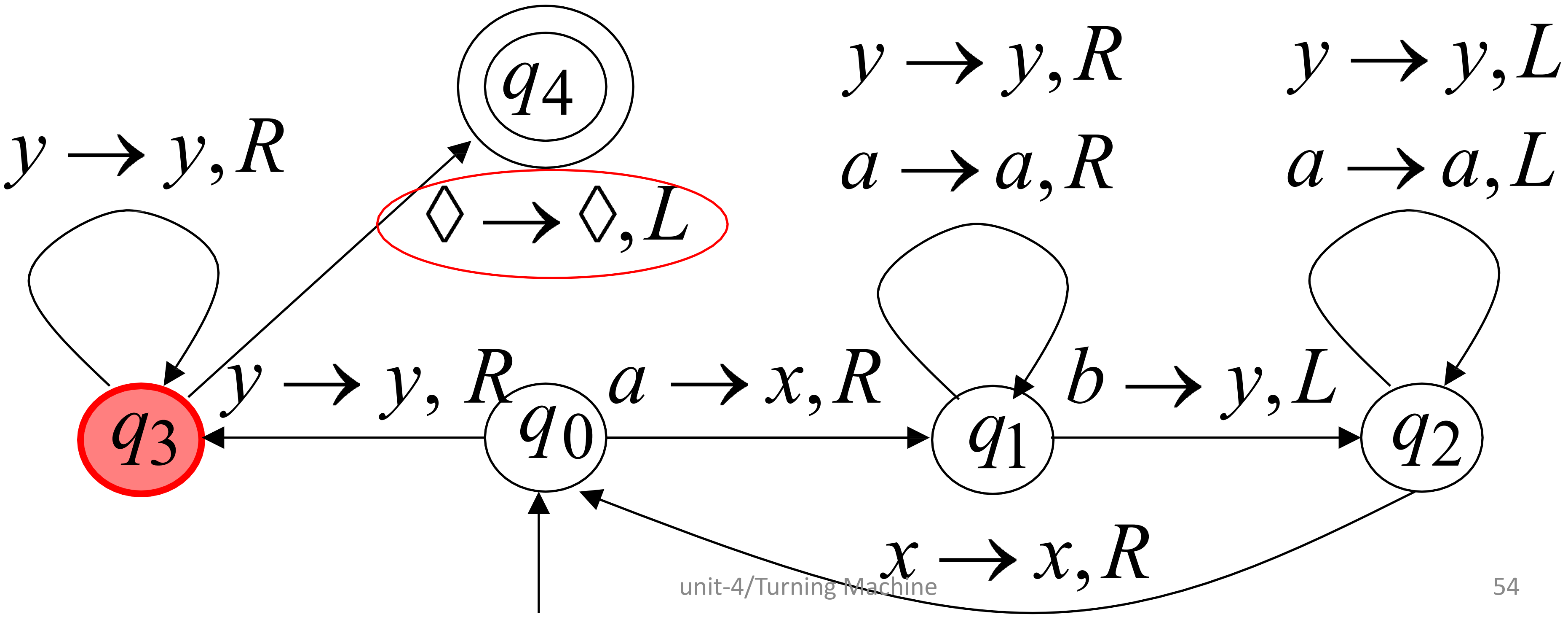

◊
x
x
y
y
◊
◊
q_3


q_4
$y \rightarrow y, R$
$\Diamond \rightarrow \Diamond, L$
$y \rightarrow y, R$
$a \rightarrow a, R$
$y \rightarrow y, L$
$a \rightarrow a, L$
q_3
$y \rightarrow y, R$
q_0
$a \rightarrow x, R$
q_1
$b \rightarrow y, L$
q_2
$x \rightarrow x, R$

Time 12

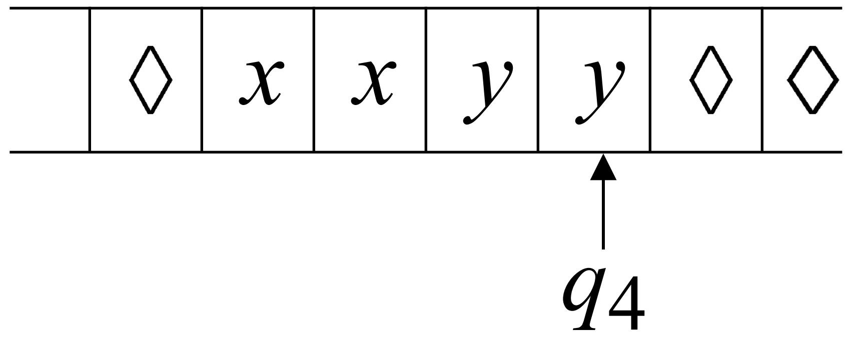


◊
x
x
y
y
◊
◊
q_3

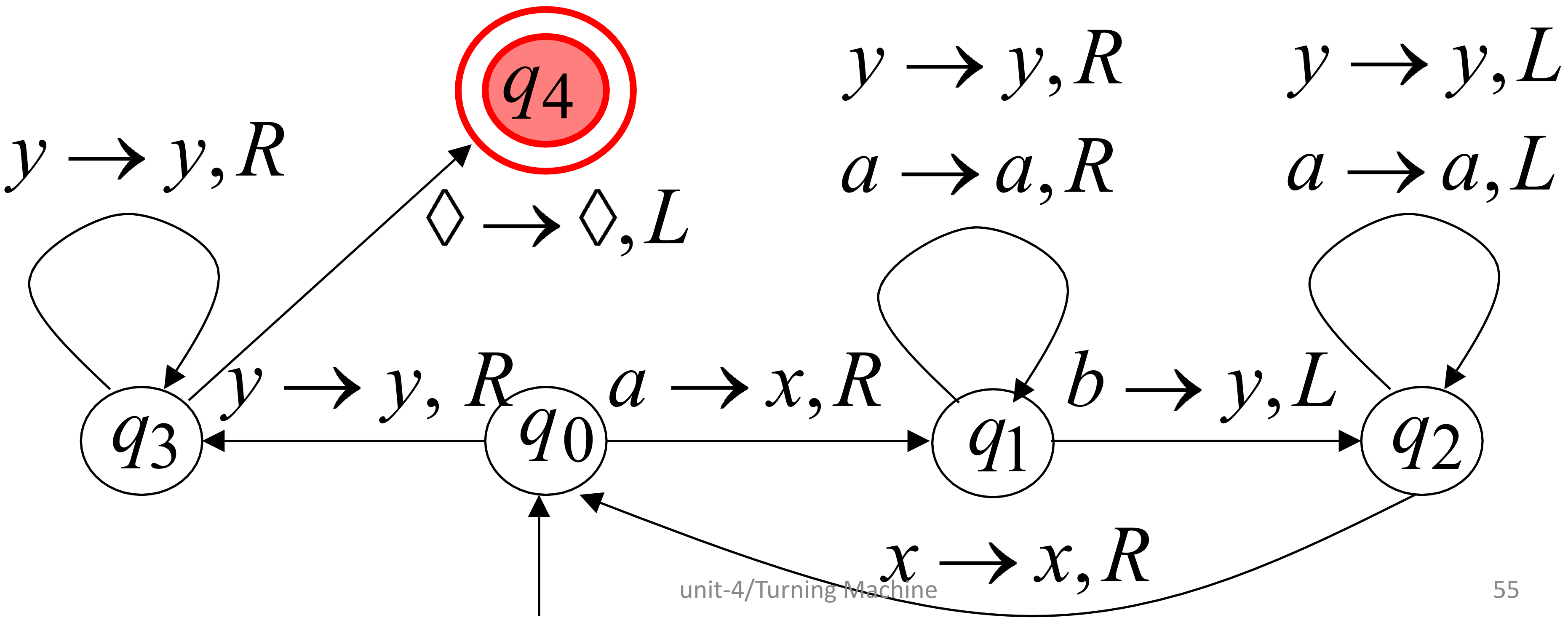


q_4
$\lozenge \rightarrow \lozenge, L$
$y \rightarrow y, R$
q_3
$y \rightarrow y, R$
q_0
$a \rightarrow x, R$
$y \rightarrow y, R$
$a \rightarrow a, R$
q_1
$b \rightarrow y, L$
$y \rightarrow y, L$
$a \rightarrow a, L$
q_2
$x \rightarrow x, R$

Time 13



Halt & Accept



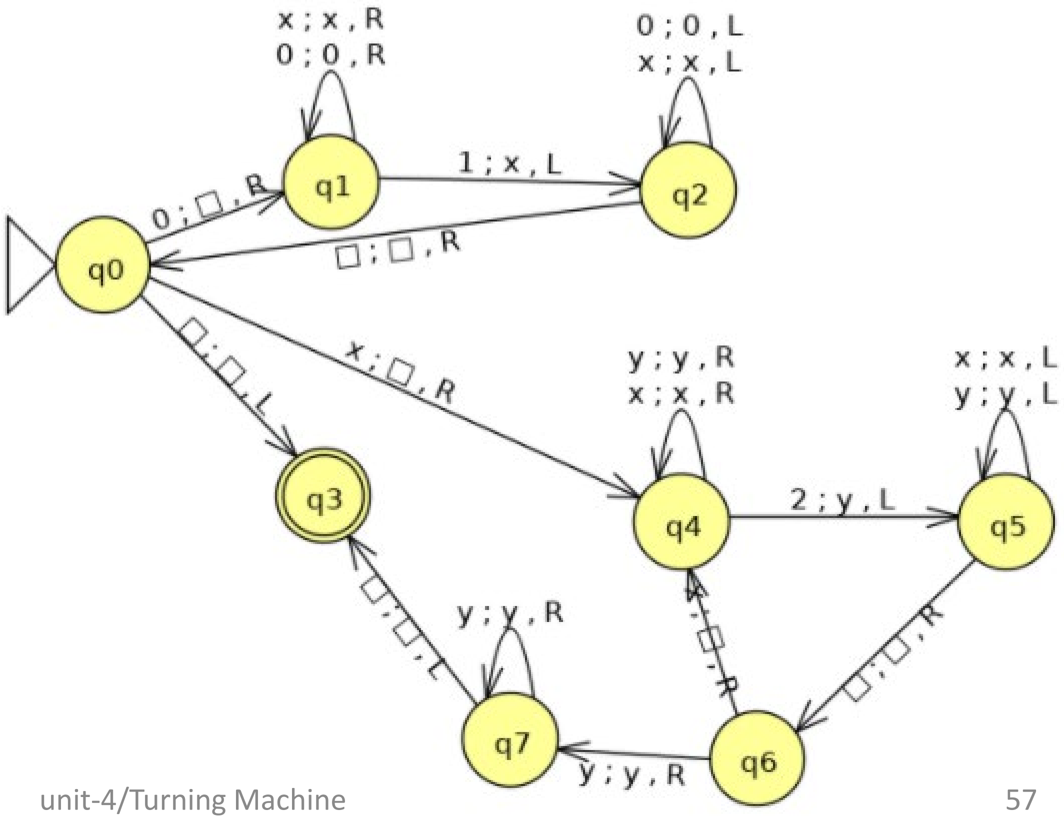
Observation:

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

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

Turing Machines as Language Acceptors- Example 2

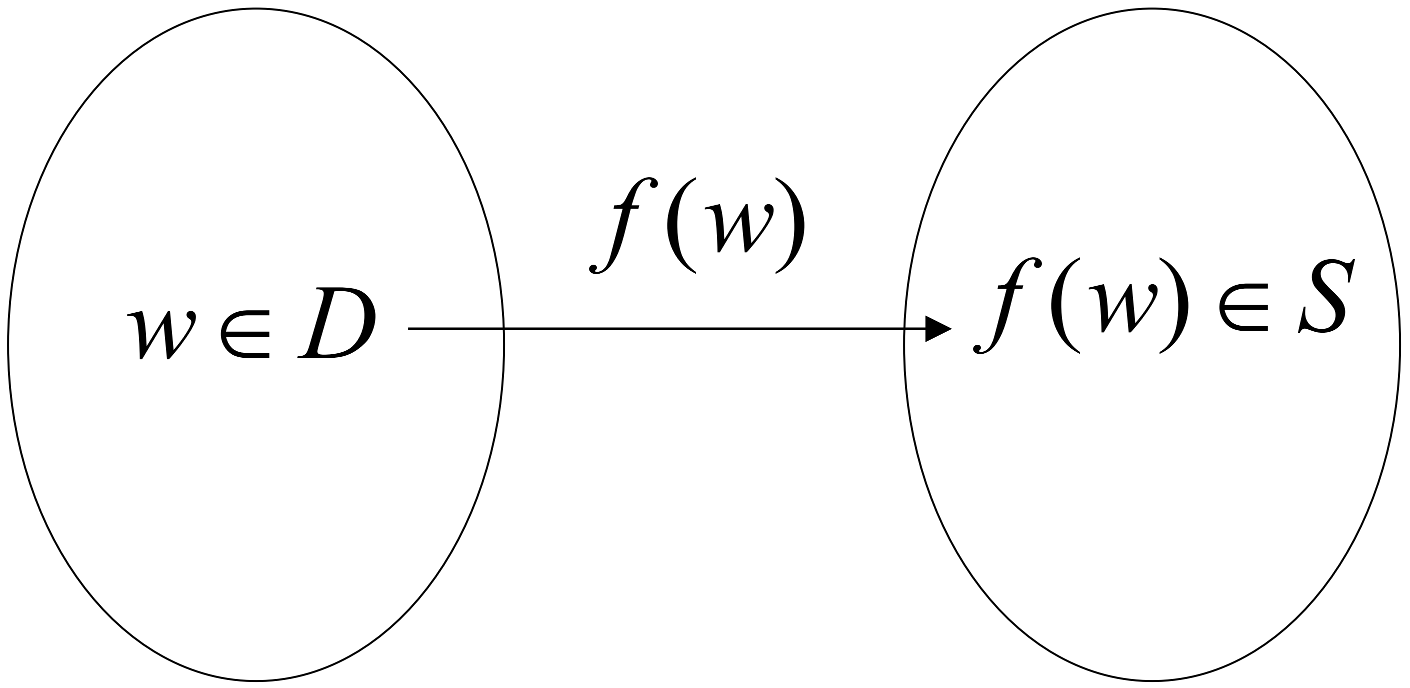
- Design a TM to recognize the language of strings of the form $0^n1^n2^n$



Computing Functions with Turing Machines

A function $f(w)$ has:

Domain: D Result Region: S



A function may have many parameters:

Example: Addition function

$$f(x,y) = x + y$$

Integer Domain

Decimal: 5

Binary: 101

Unary: 11111

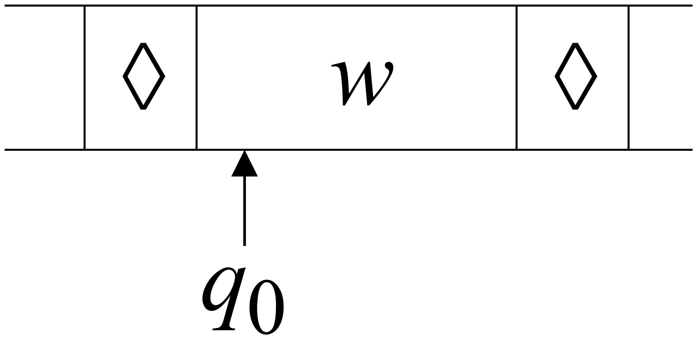
We prefer **unary** representation:

easier to manipulate with Turing machines

Definition:

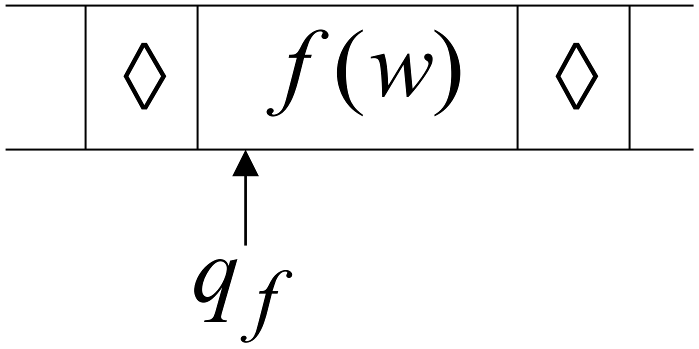
A function f is computable if there is a Turing Machine M such that:

Initial configuration



initial state

Final configuration



accept state

For all $w \in D$ Domain

In other words:

A function f is computable if there is a Turing Machine M such that:

$$q_0\, w \overset{*}{\succ} q_f\, f(w)$$

Initial Configuration

Halts at

Final Configuration

For all $w \in D$ Domain

Example

The function $f(x, y) = x + y$ is computable

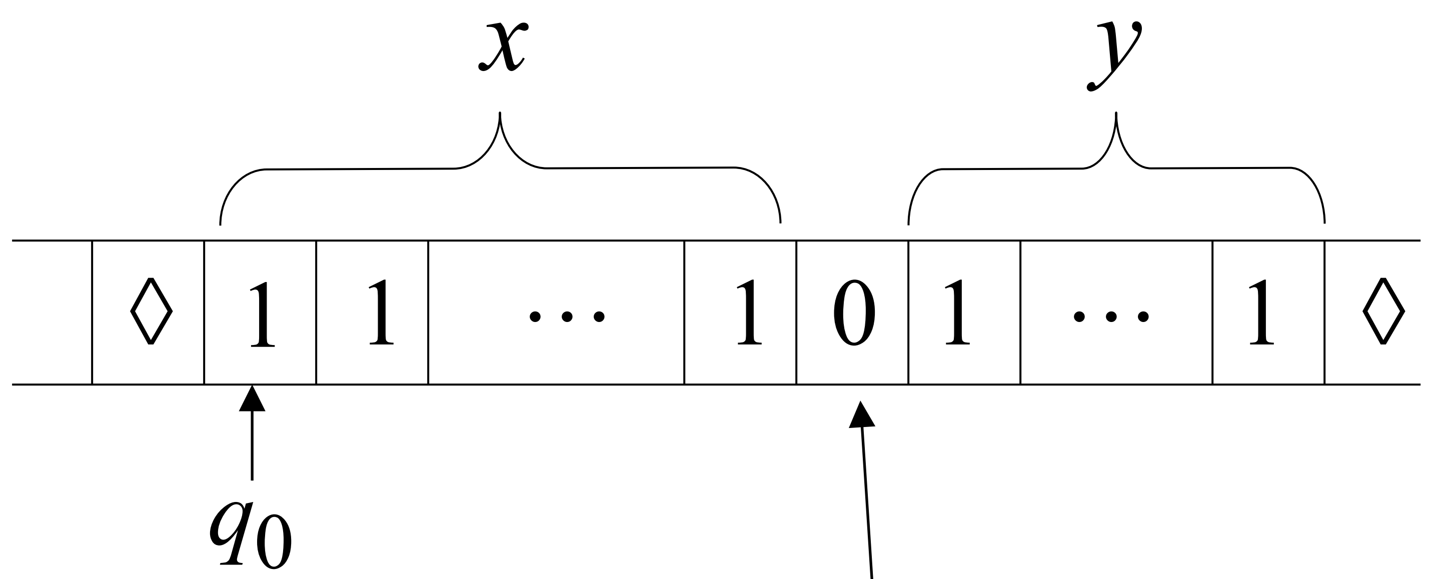
x, y are integers

Turing Machine:

Input string: $x0y$ unary

Output string: $xy0$ unary

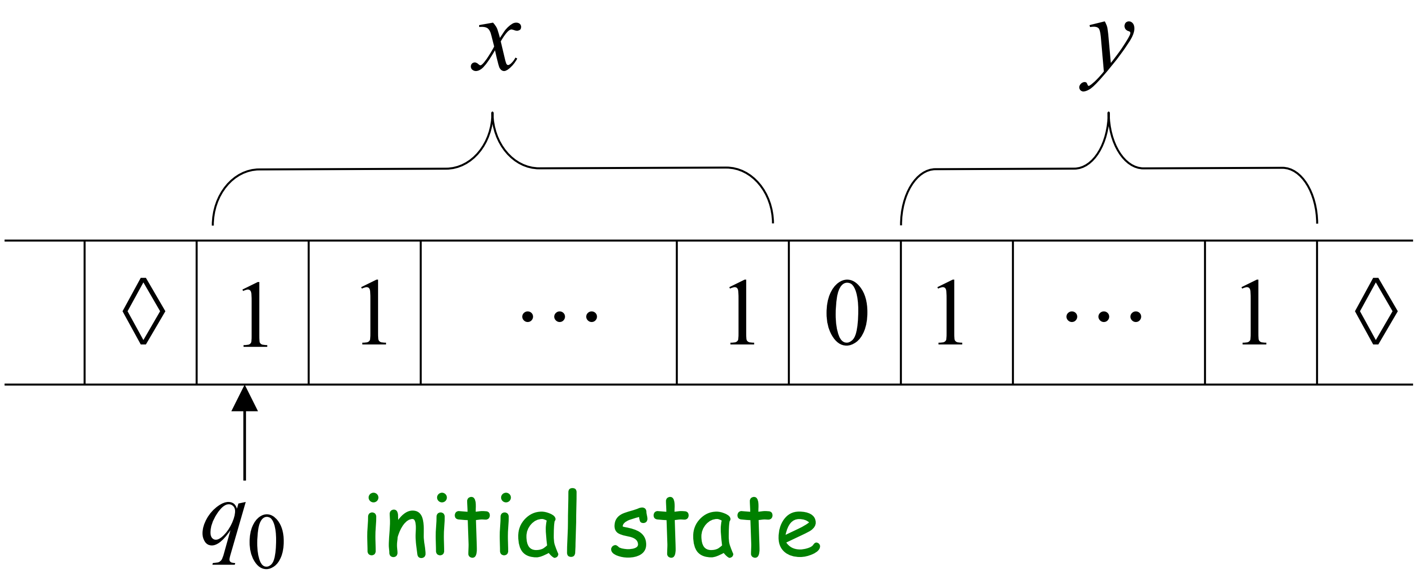
Start



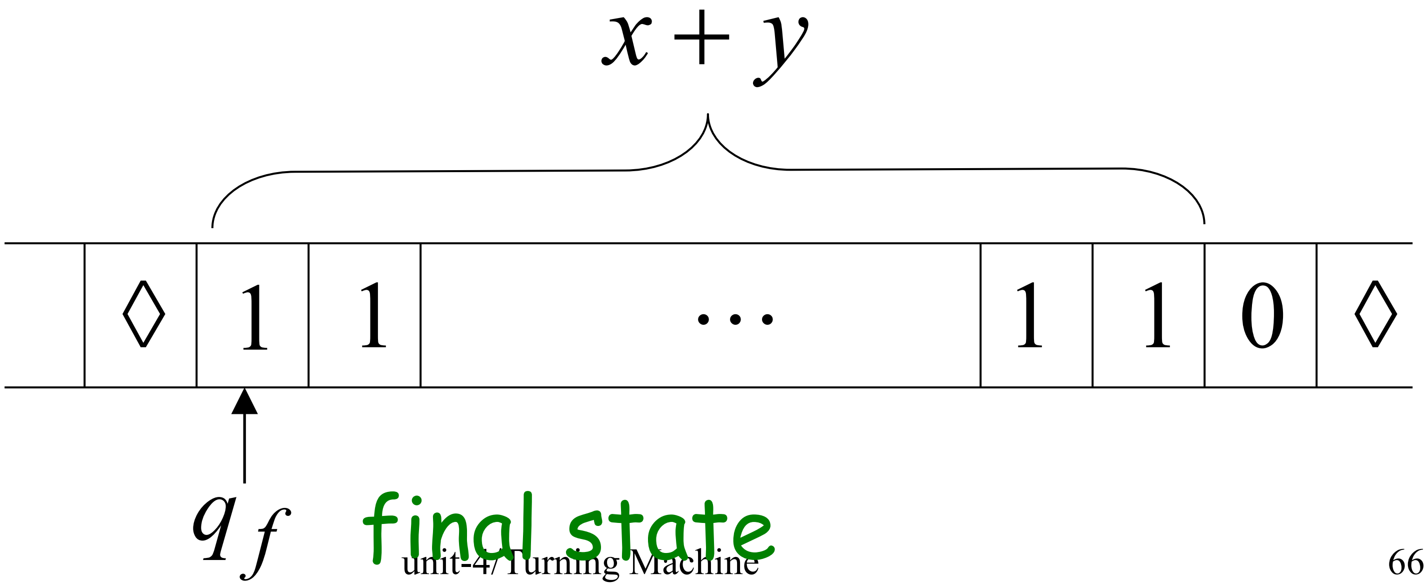
initial state

The 0 is the delimiter that separates the two numbers

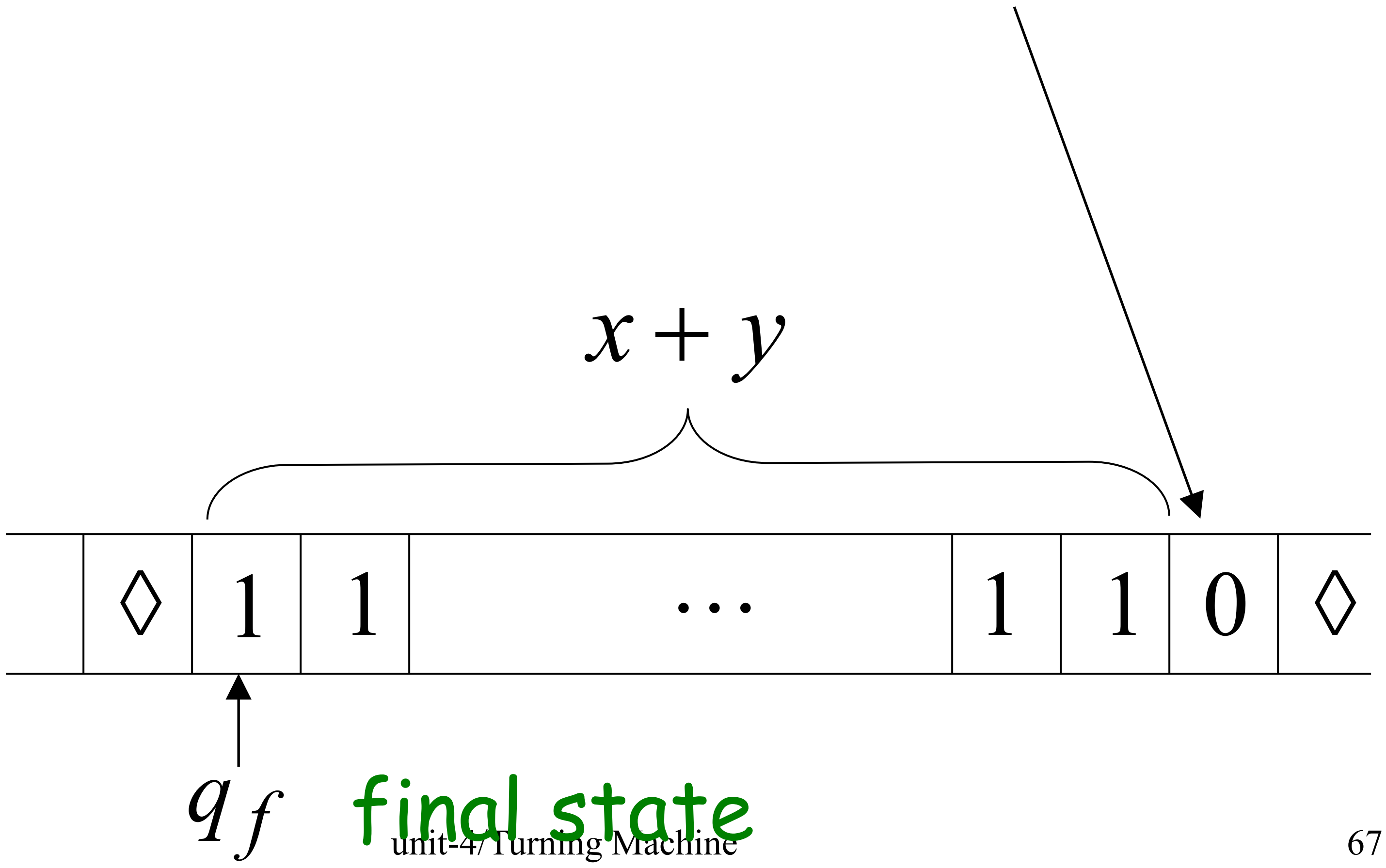
Start



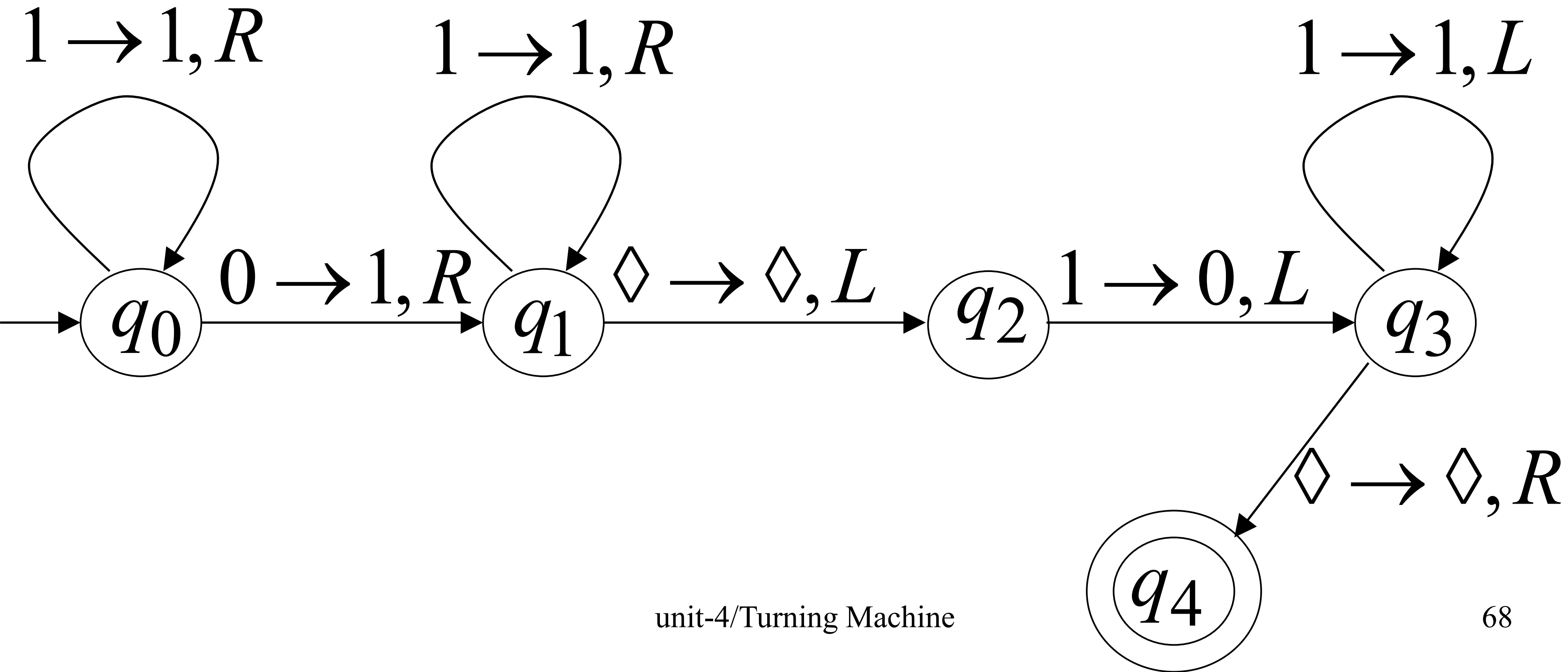
Finish



The 0 here helps when we use the result for other operations



Turing machine for function $f(x, y) = x + y$

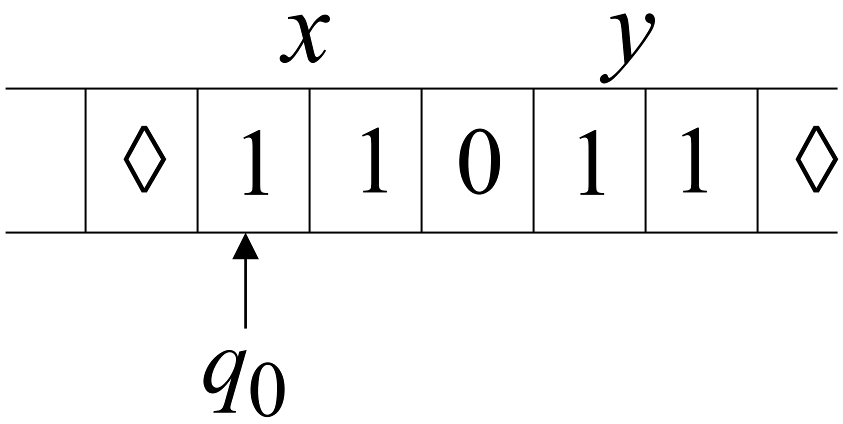


Execution Example:

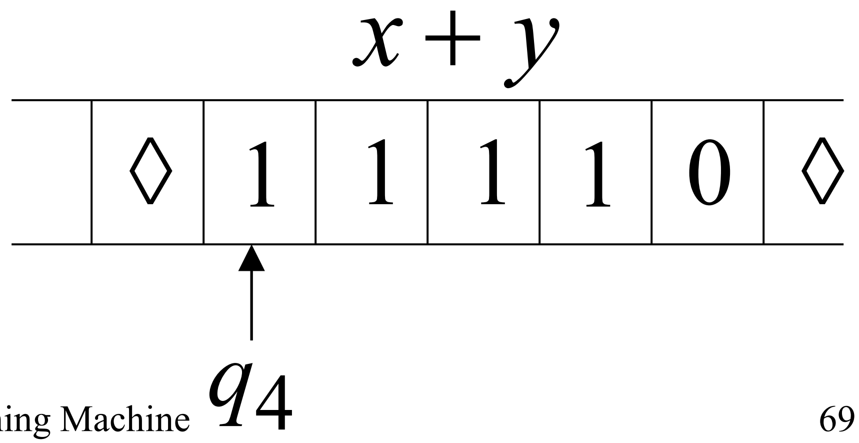
$x = 11$ (=2)

$y = 11$ (=2)

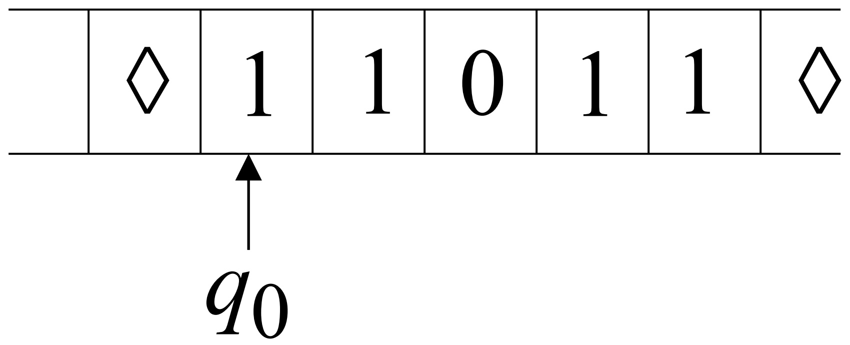
Time 0

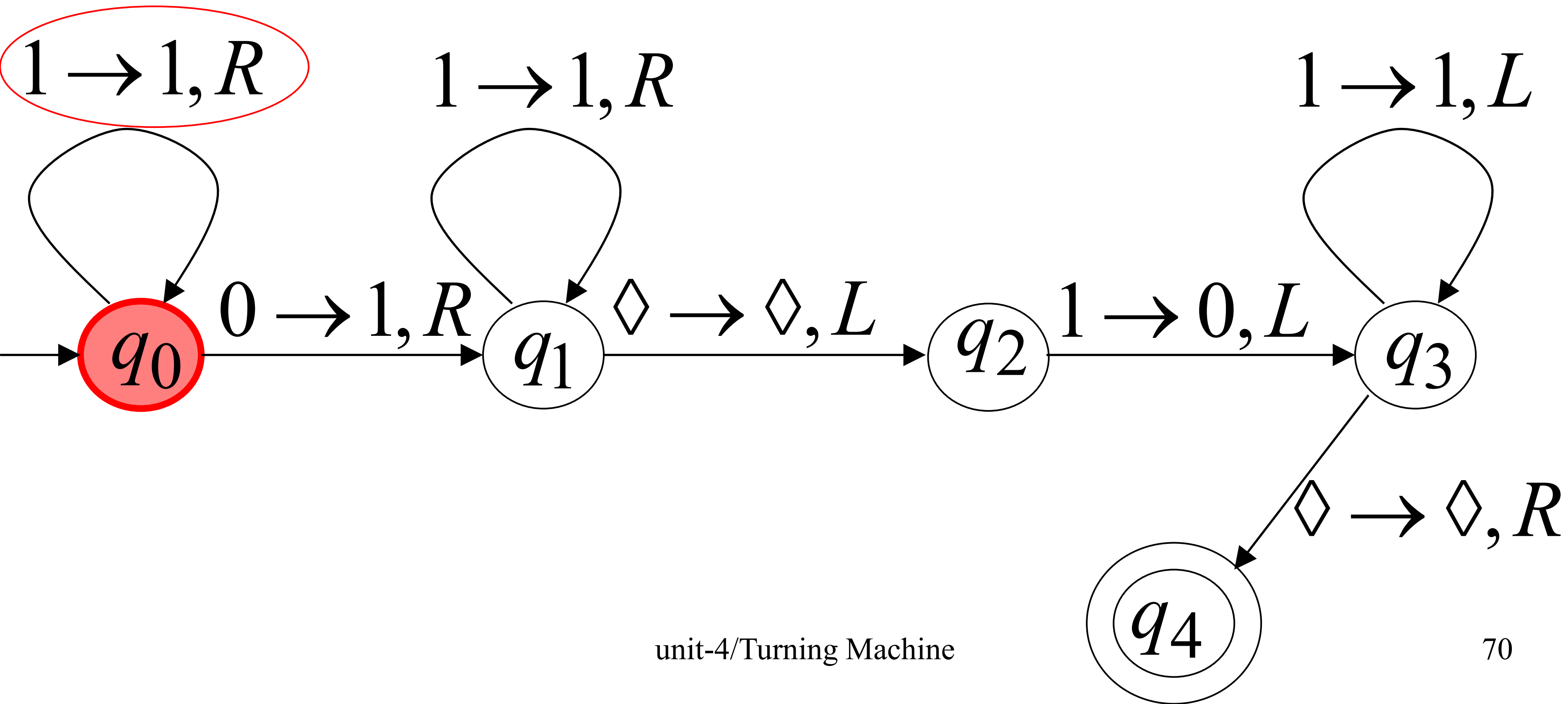


Final Result

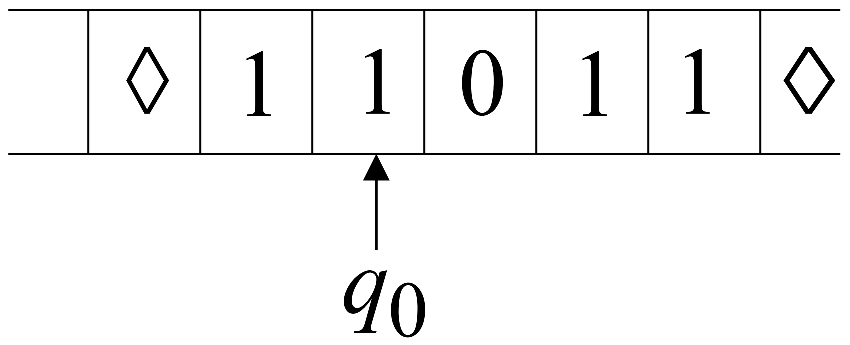


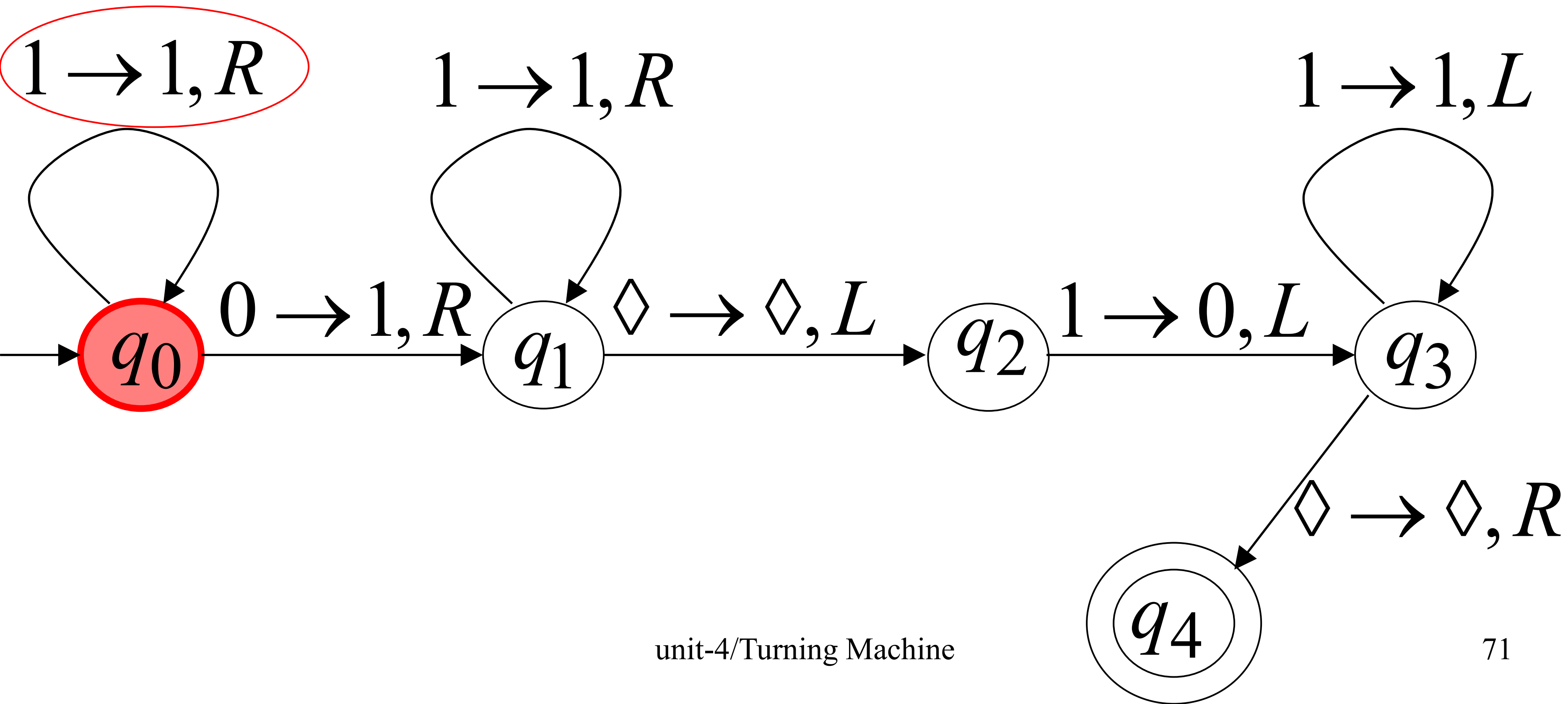
Time 0


◊
1
1
0
1
1
◊
q_0

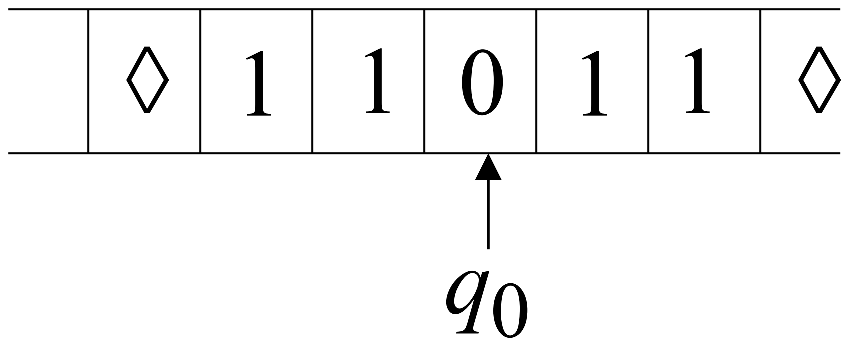

$1 \rightarrow 1, R$
$1 \rightarrow 1, R$
$1 \rightarrow 1, L$
$0 \rightarrow 1, R$
$◊ \rightarrow ◊, L$
$1 \rightarrow 0, L$
$◊ \rightarrow ◊, R$
q_0
q_1
q_2
q_3
q_4

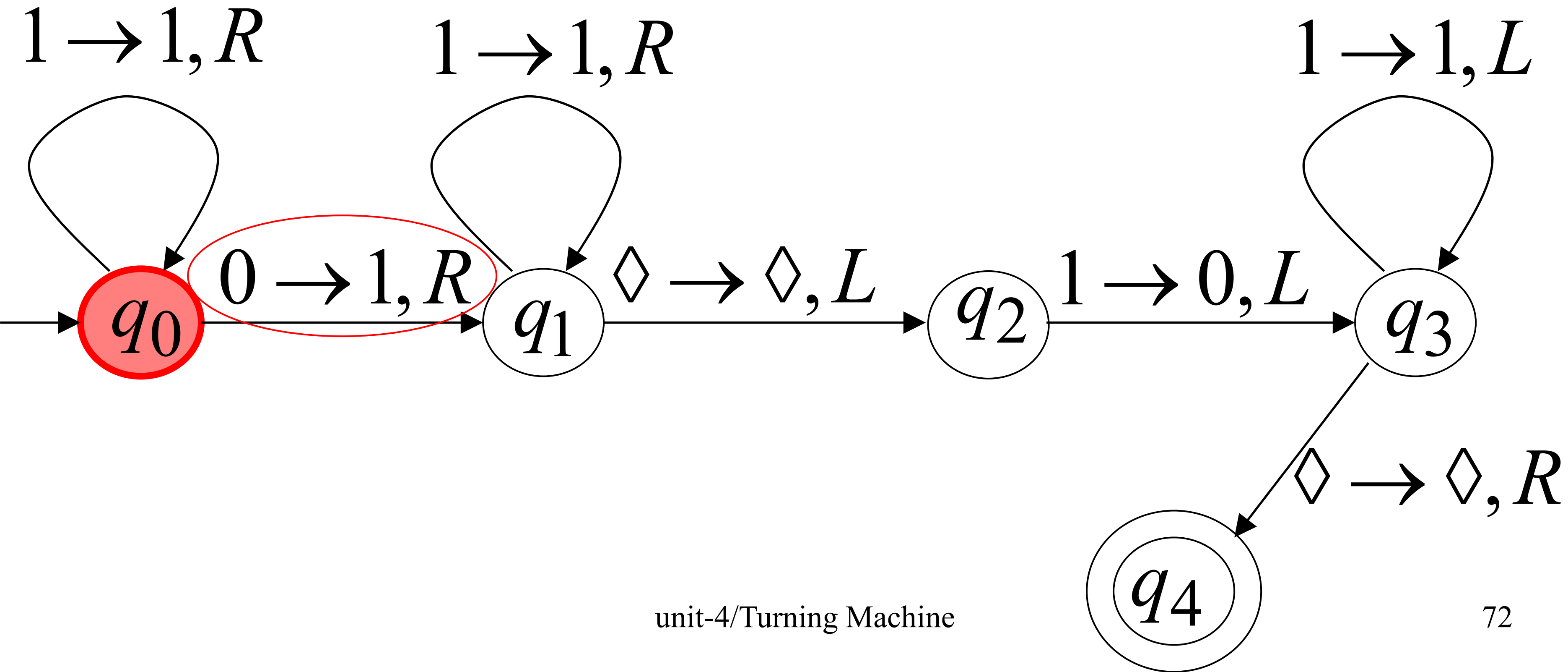
Time 1


◊
1
1
0
1
1
◊
q_0

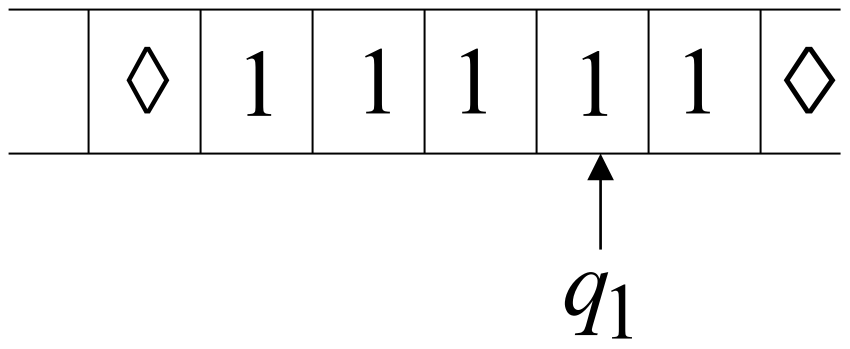

$1 \rightarrow 1, R$
$1 \rightarrow 1, R$
$1 \rightarrow 1, L$
q_0
$0 \rightarrow 1, R$
q_1
$◊ \rightarrow ◊, L$
q_2
$1 \rightarrow 0, L$
q_3
$◊ \rightarrow ◊, R$
q_4

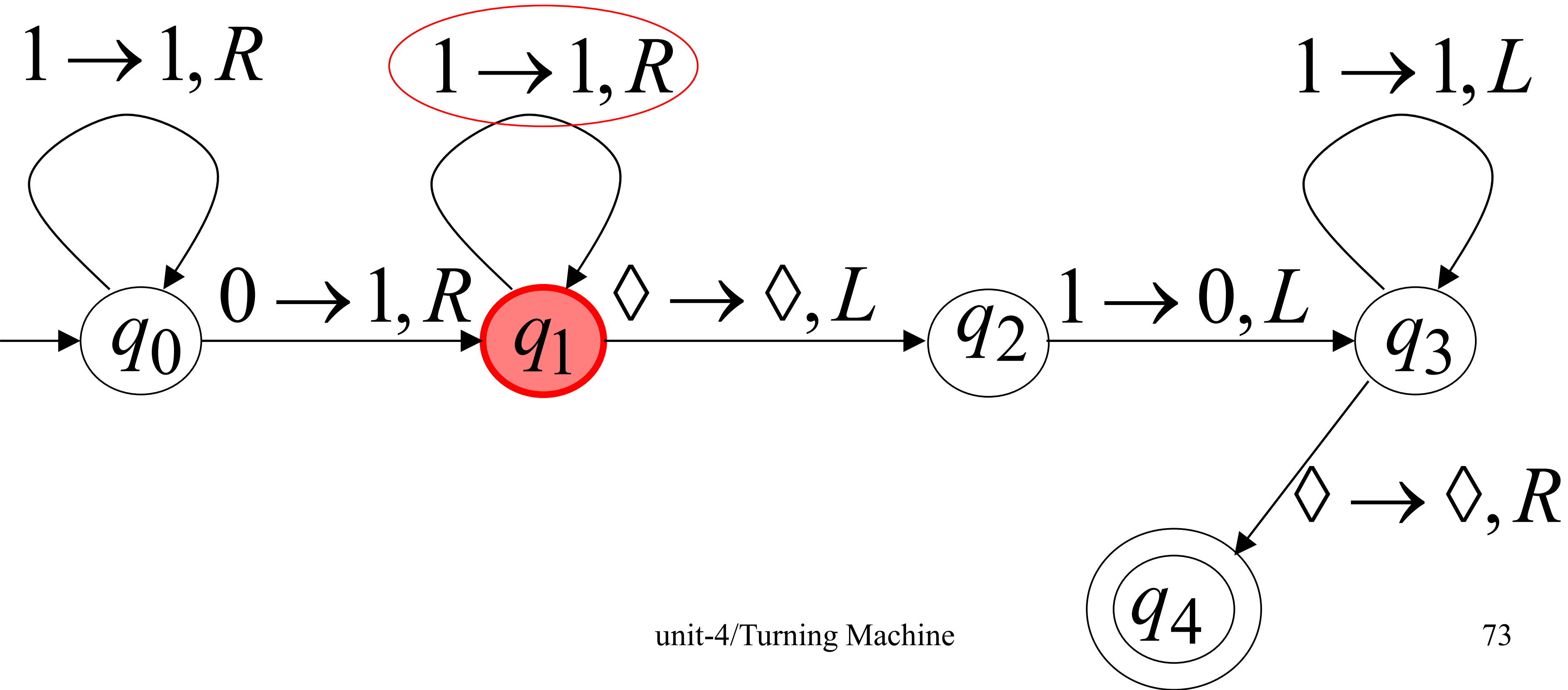
Time 2



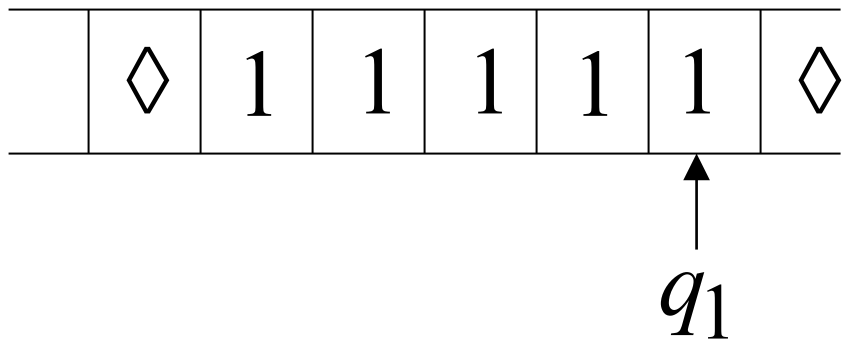


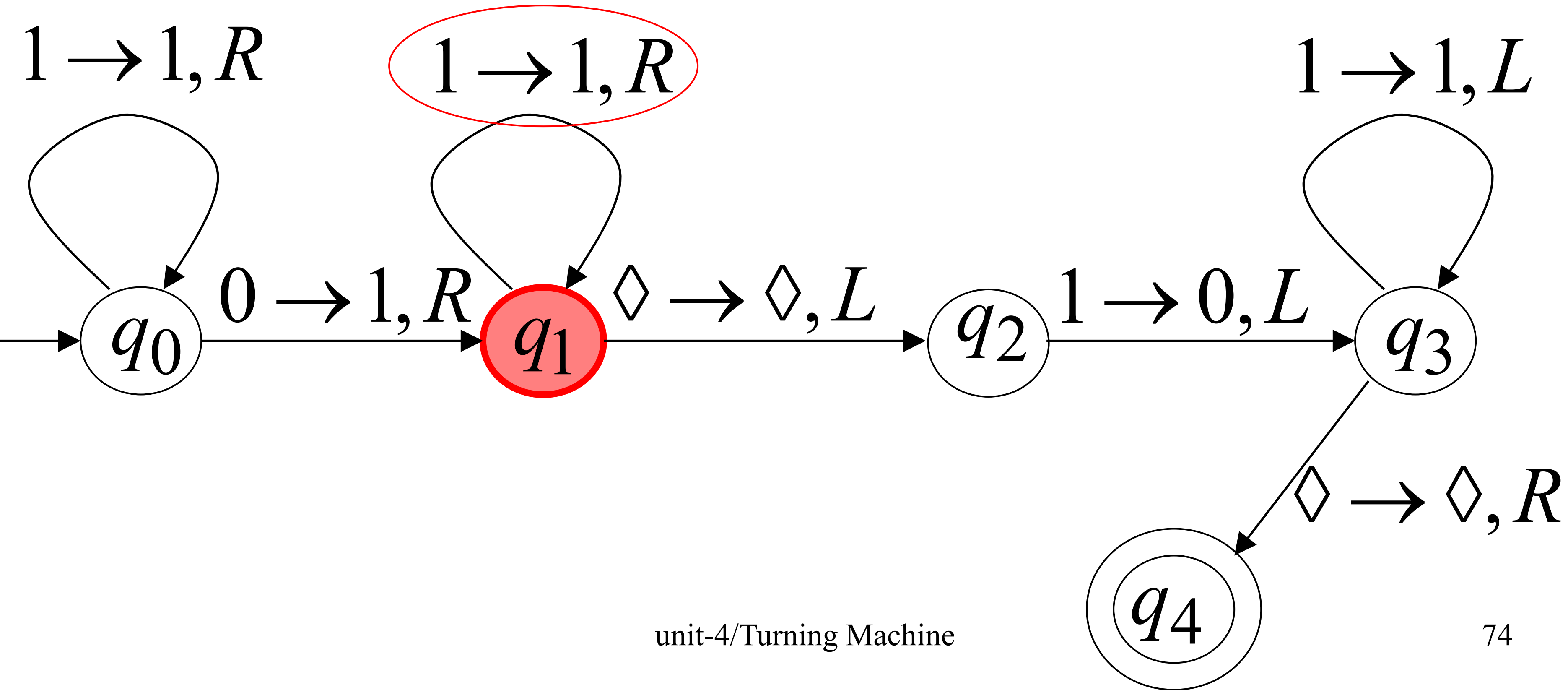
Time 3


◊
1
1
1
1
1
◊
q_1

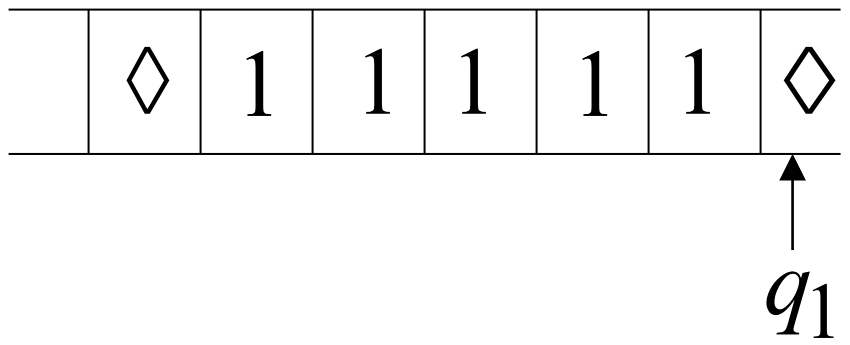

$1 \rightarrow 1, R$
$1 \rightarrow 1, R$
$1 \rightarrow 1, L$
$0 \rightarrow 1, R$
$◊ \rightarrow ◊, L$
$1 \rightarrow 0, L$
$◊ \rightarrow ◊, R$
q_0
q_1
q_2
q_3
q_4

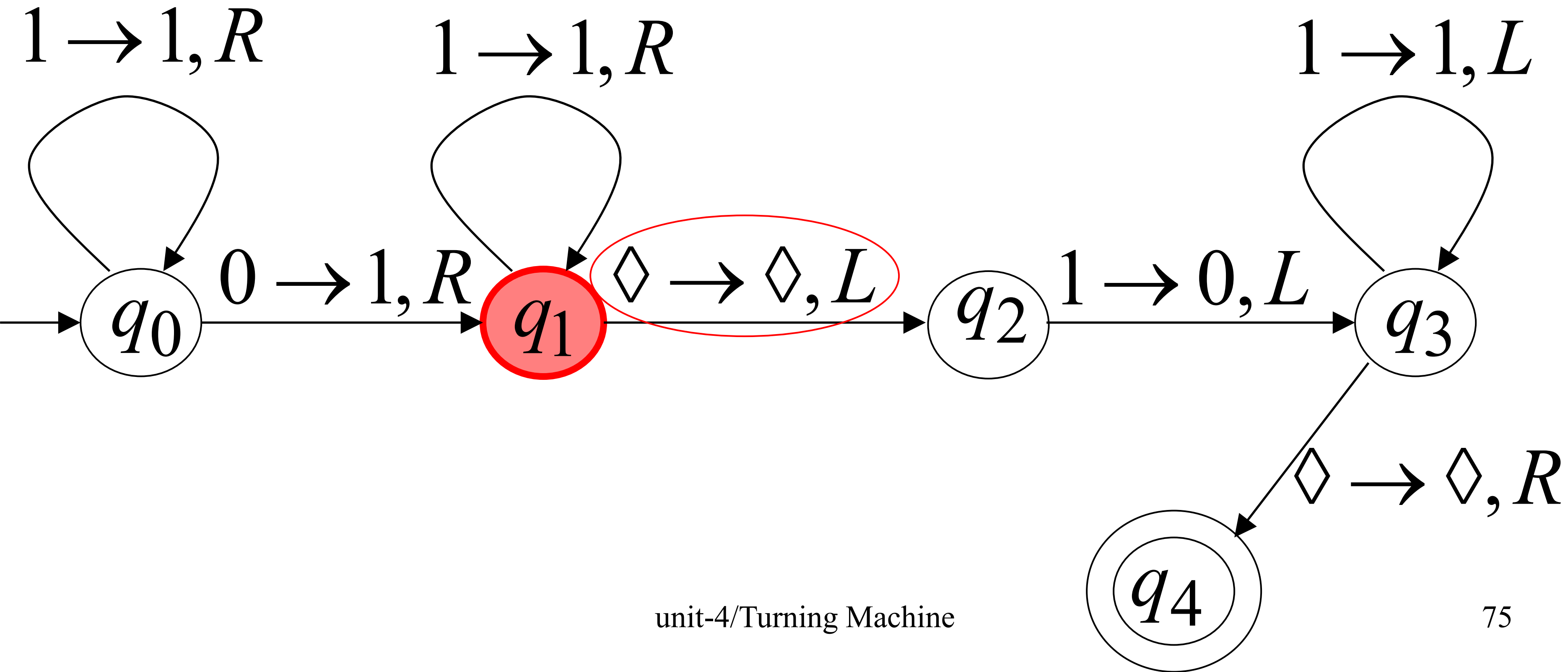
Time 4


◊
1
1
1
1
1
◊
q_1

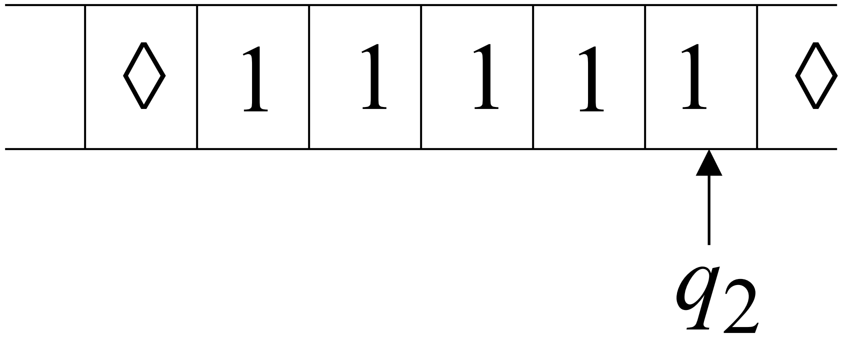

$1 \rightarrow 1, R$
$1 \rightarrow 1, R$
$1 \rightarrow 1, L$
q_0
$0 \rightarrow 1, R$
q_1
$◊ \rightarrow ◊, L$
q_2
$1 \rightarrow 0, L$
q_3
$◊ \rightarrow ◊, R$
q_4

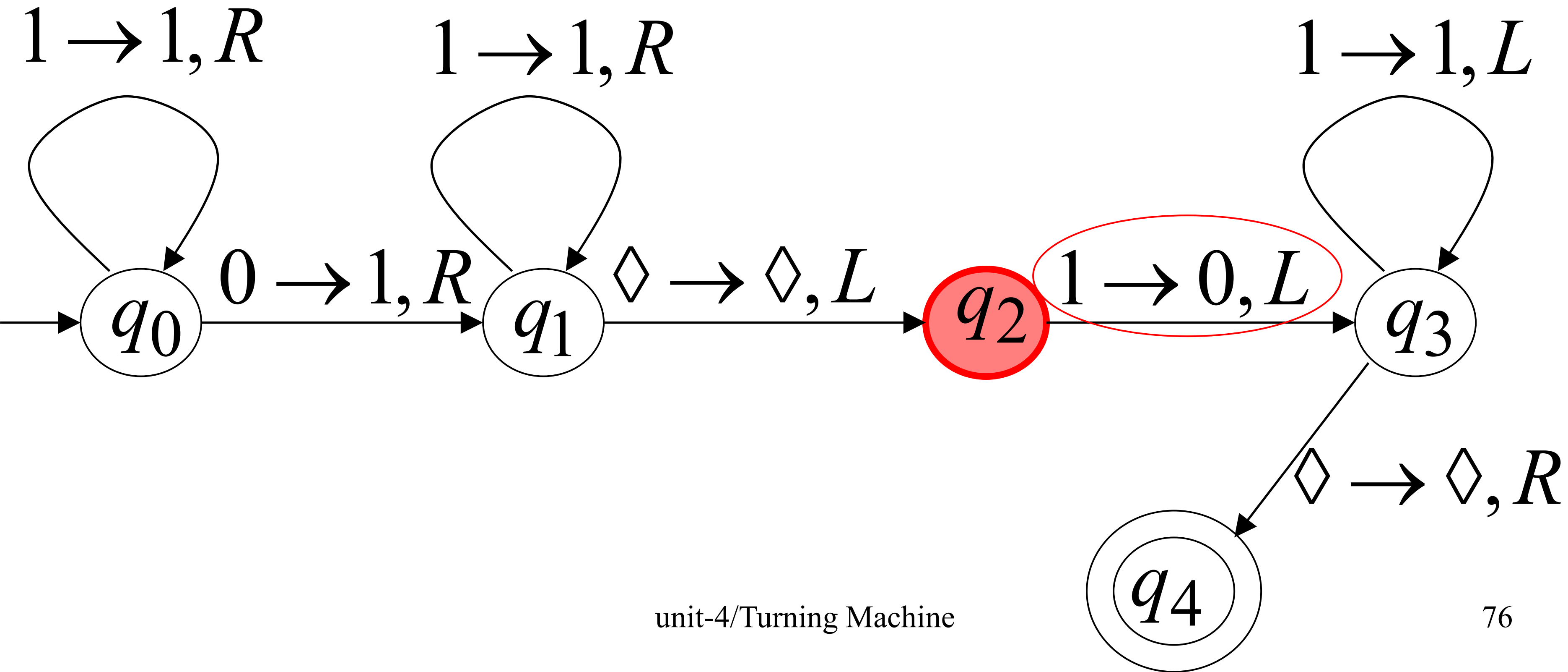
Time 5


◊
1
1
1
1
1
◊
q_1

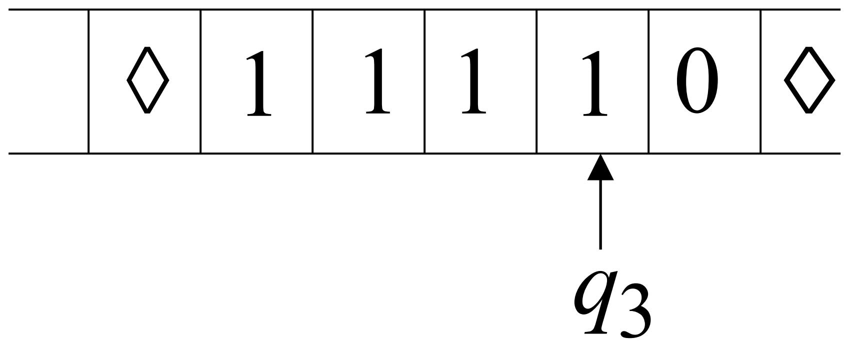

$1 \rightarrow 1, R$
$1 \rightarrow 1, R$
$1 \rightarrow 1, L$
q_0
$0 \rightarrow 1, R$
q_1
$◊ \rightarrow ◊, L$
q_2
$1 \rightarrow 0, L$
q_3
$◊ \rightarrow ◊, R$
q_4

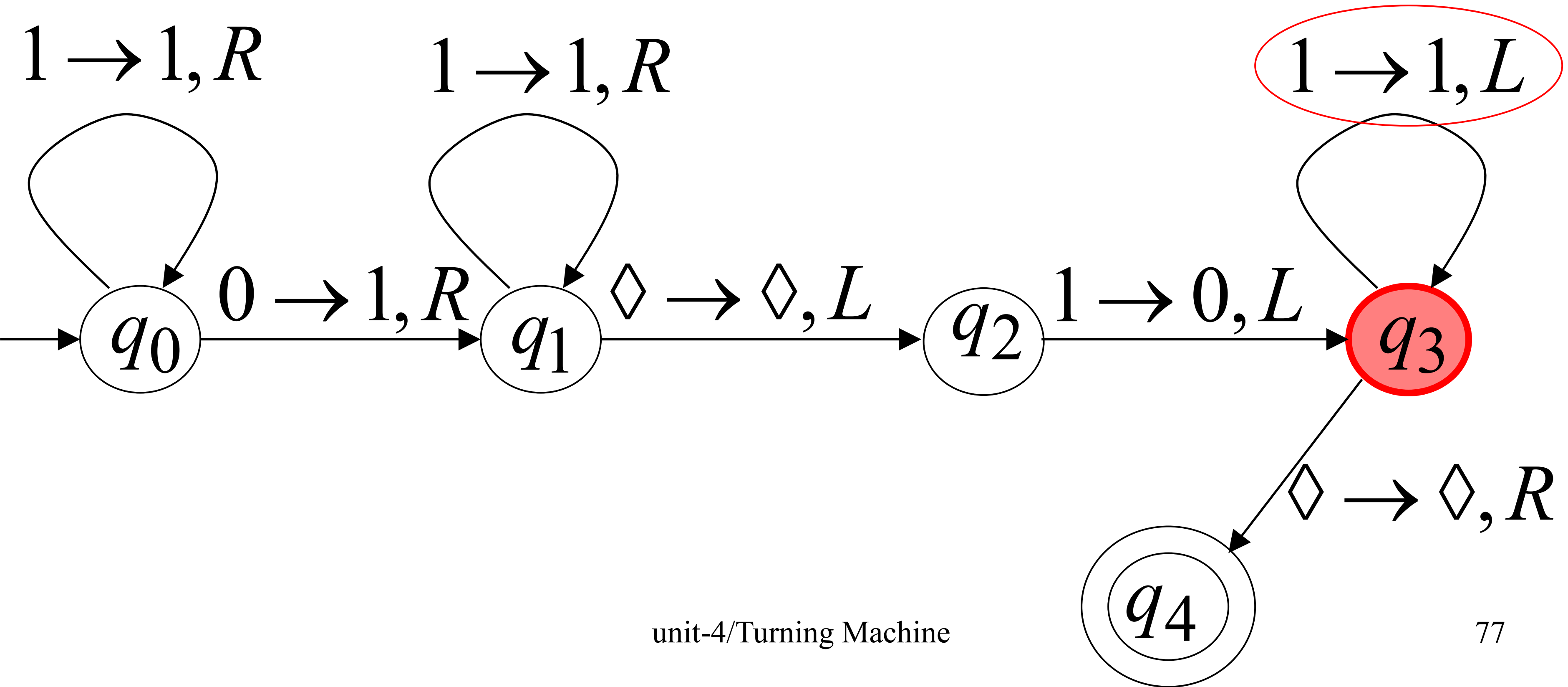
Time 6


◊
1
1
1
1
1
◊
q_2

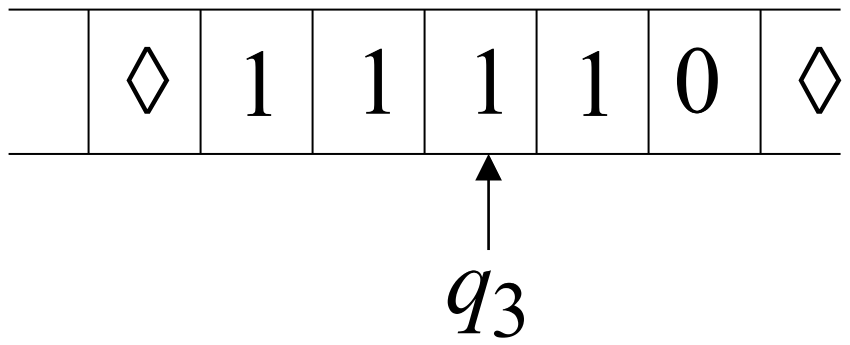

$1 \to 1, R$
$1 \to 1, R$
$1 \to 1, L$
q_0
$0 \to 1, R$
q_1
$\Diamond \to \Diamond, L$
q_2
$1 \to 0, L$
q_3
$\Diamond \to \Diamond, R$
q_4

Time 7

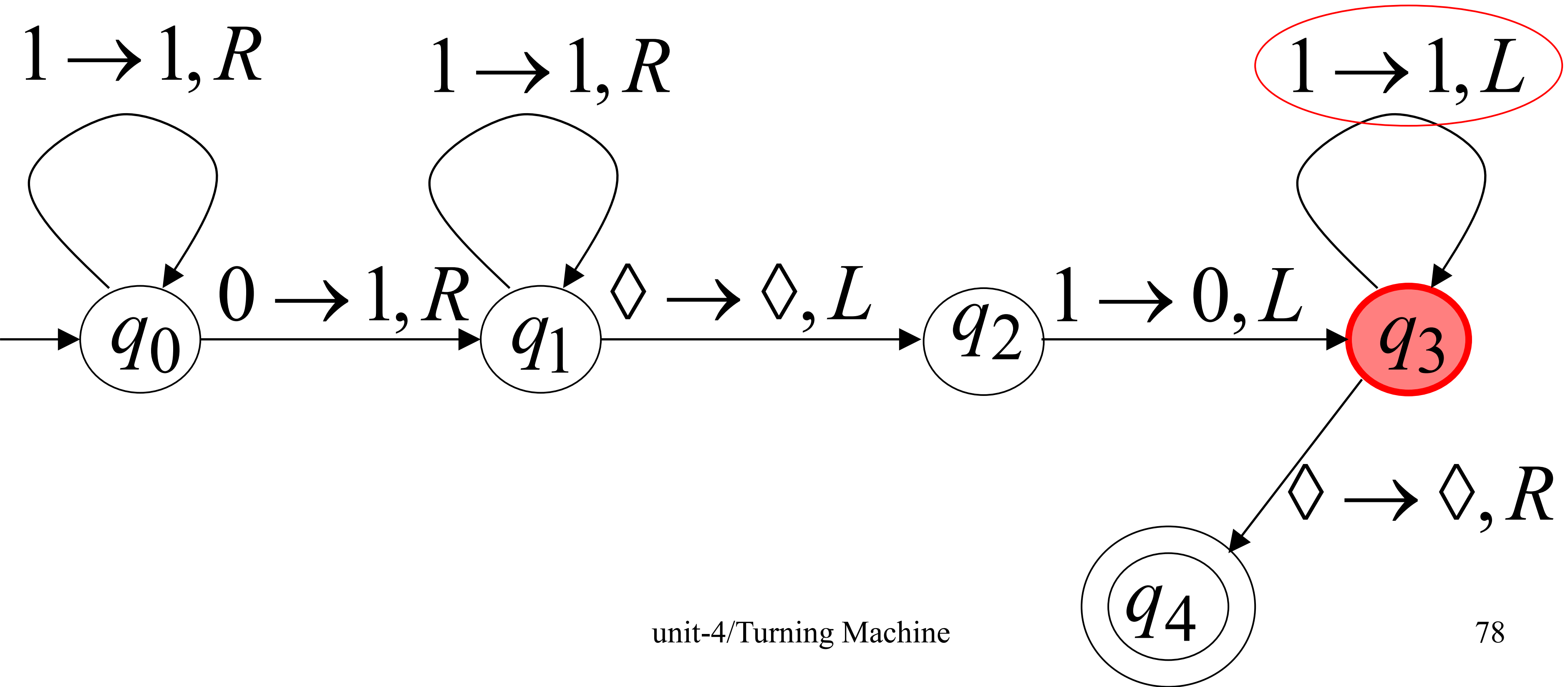

◊
1
1
1
1
0
◊
q_3


$1 \rightarrow 1, R$
$1 \rightarrow 1, R$
$1 \rightarrow 1, L$
q_0
$0 \rightarrow 1, R$
q_1
$◊ \rightarrow ◊, L$
q_2
$1 \rightarrow 0, L$
q_3
$◊ \rightarrow ◊, R$
q_4

Time 8

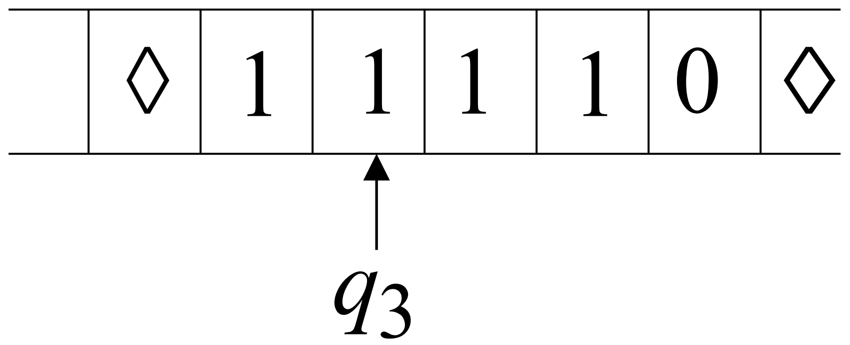


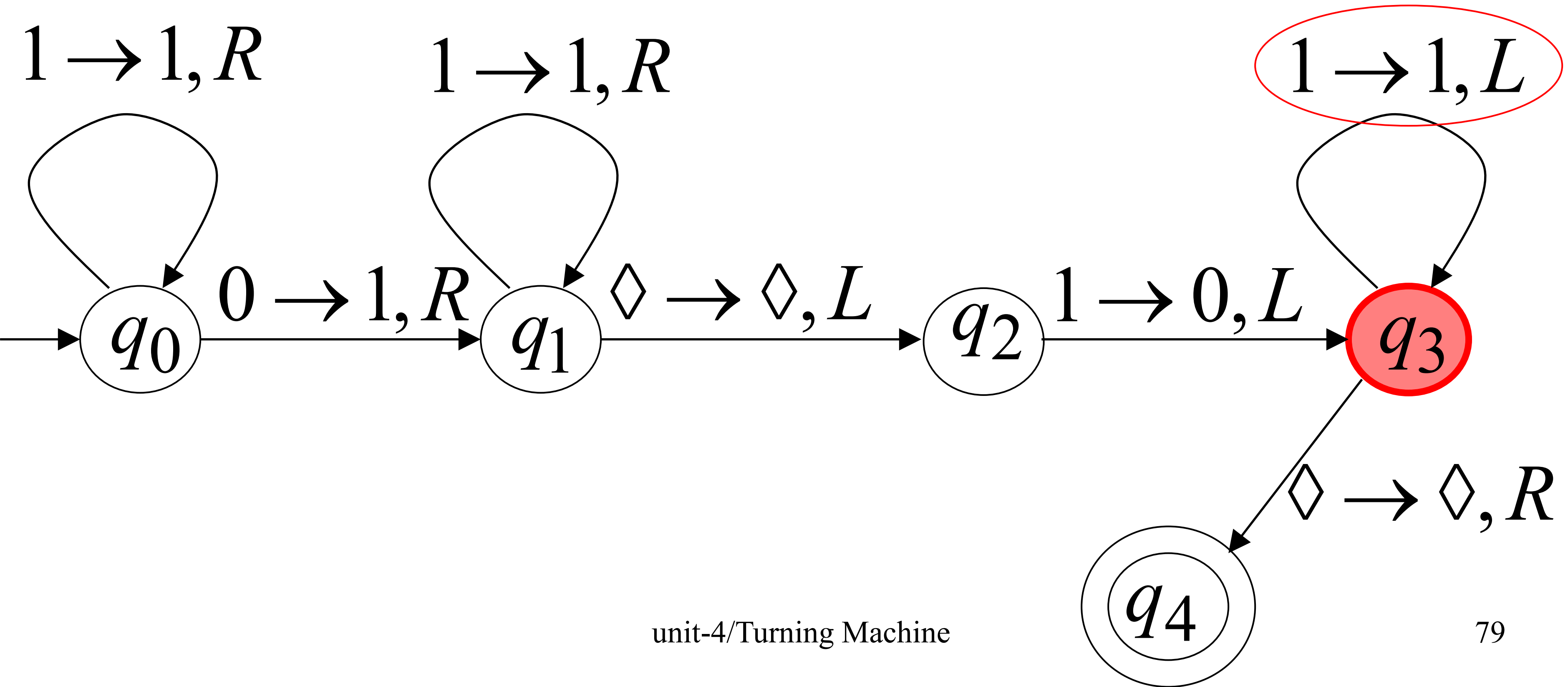
◊
1
1
1
1
0
◊
q_3



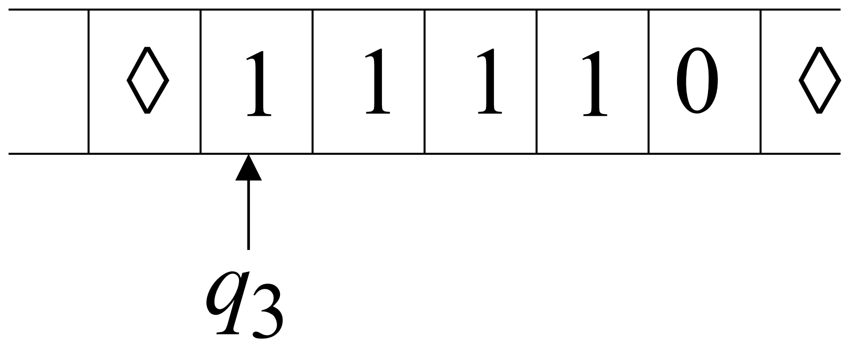
$1 \rightarrow 1, R$
$1 \rightarrow 1, R$
$1 \rightarrow 1, L$
q_0
$0 \rightarrow 1, R$
q_1
$\lozenge \rightarrow \lozenge, L$
q_2
$1 \rightarrow 0, L$
q_3
$\lozenge \rightarrow \lozenge, R$
q_4

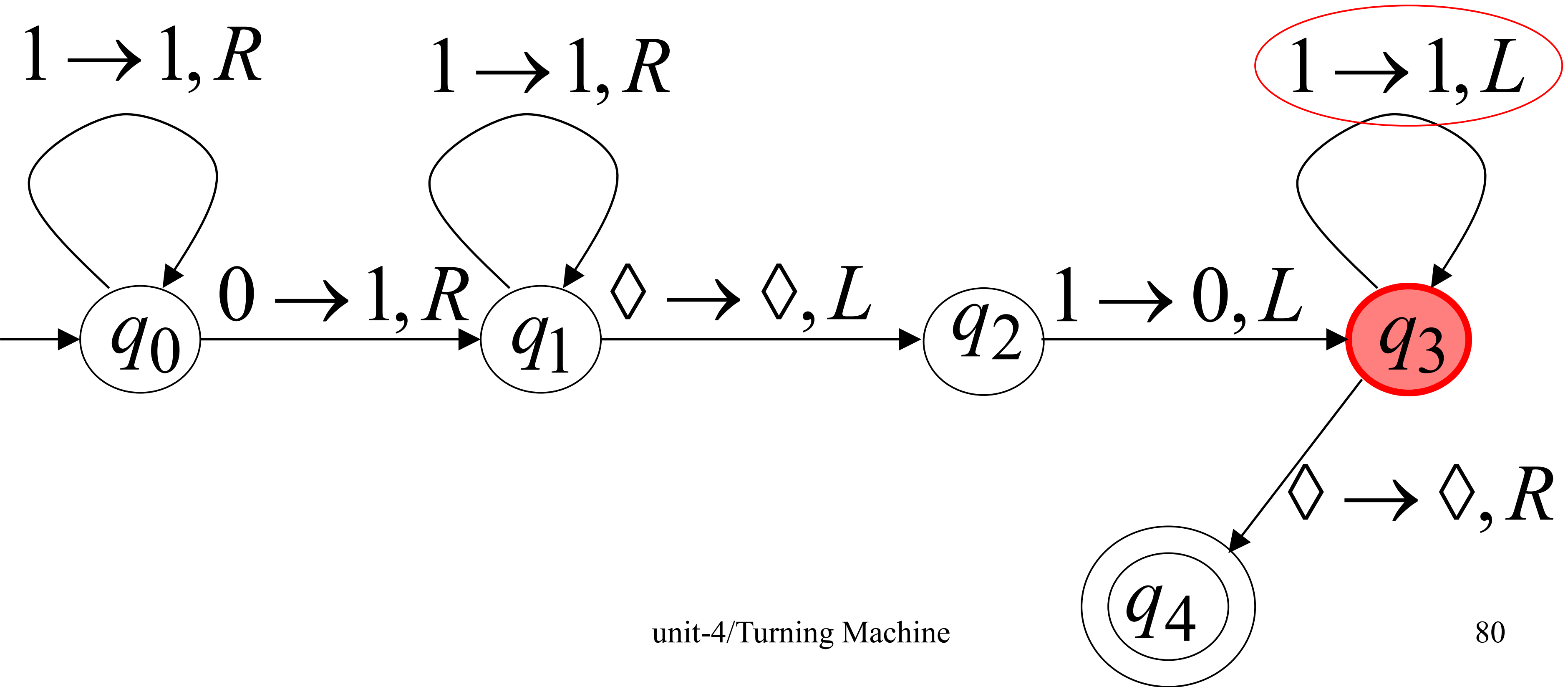
Time 9


◊
1
1
1
1
0
◊
q_3

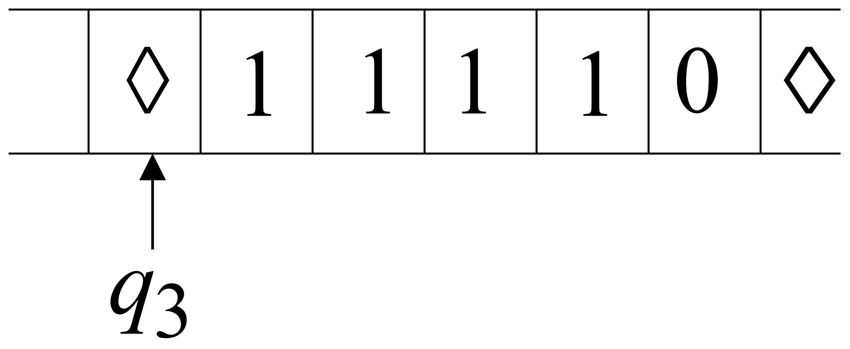

$1 \rightarrow 1, R$
$1 \rightarrow 1, R$
$1 \rightarrow 1, L$
$0 \rightarrow 1, R$
$◊ \rightarrow ◊, L$
$1 \rightarrow 0, L$
$◊ \rightarrow ◊, R$
q_0
q_1
q_2
q_3
q_4

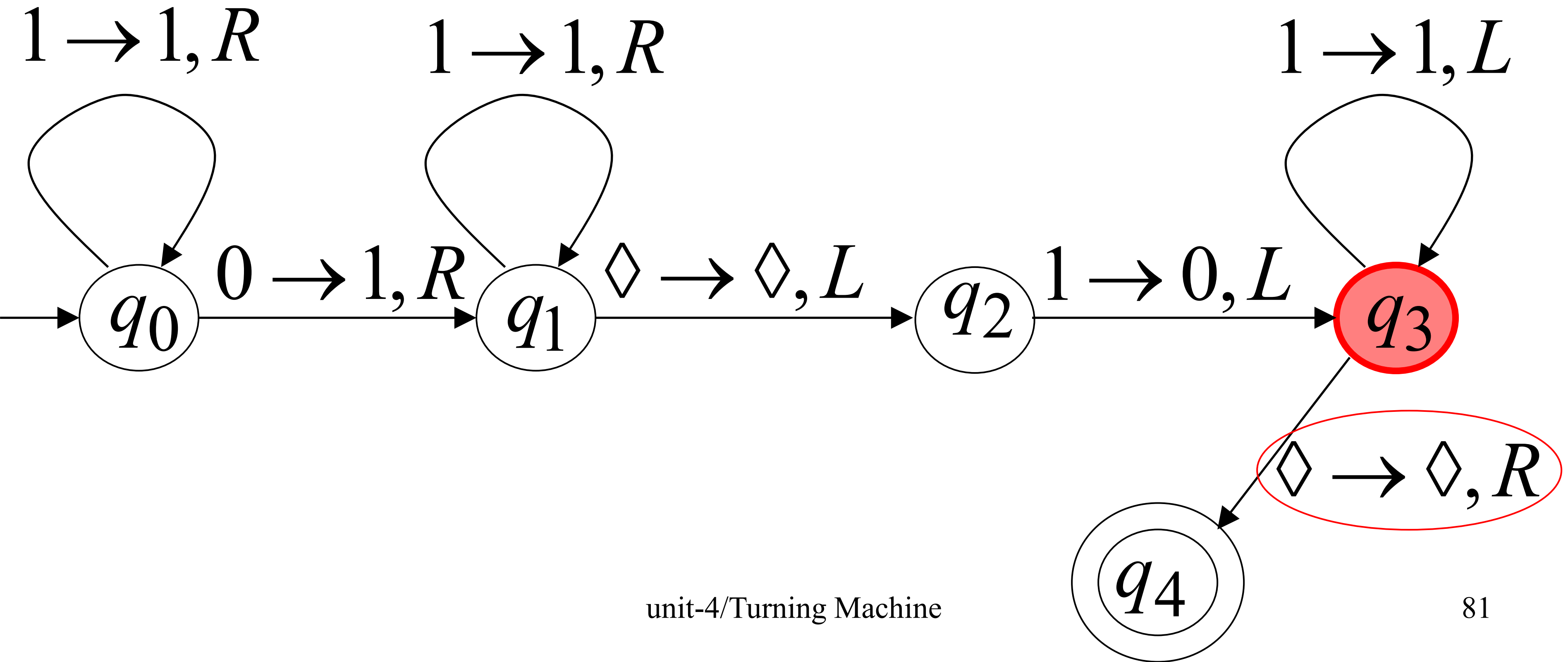
Time 10


◊
1
1
1
1
0
◊
q_3

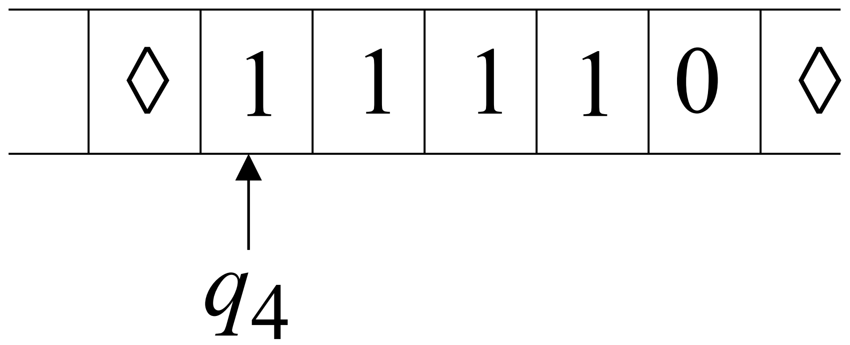

$1 \rightarrow 1, R$
$1 \rightarrow 1, R$
$1 \rightarrow 1, L$
$0 \rightarrow 1, R$
$\lozenge \rightarrow \lozenge, L$
$1 \rightarrow 0, L$
$\lozenge \rightarrow \lozenge, R$
q_0
q_1
q_2
q_3
q_4

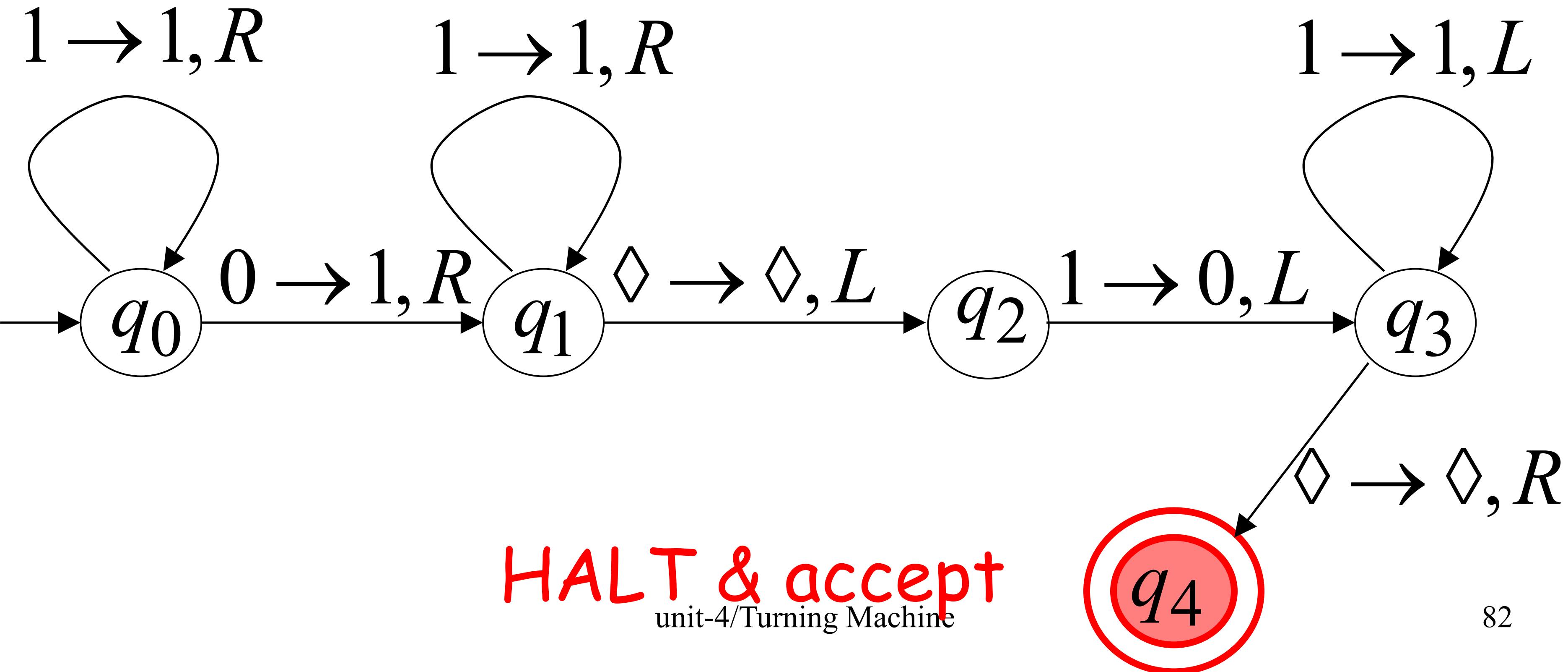
Time 11


◊
1
1
1
1
0
◊
q_3


$1 \rightarrow 1, R$
$1 \rightarrow 1, R$
$1 \rightarrow 1, L$
q_0
$0 \rightarrow 1, R$
q_1
$◊ \rightarrow ◊, L$
q_2
$1 \rightarrow 0, L$
q_3
$◊ \rightarrow ◊, R$
q_4

Time 12





HALT & accept

Another Example

The function $f(x) = 2x$ is computable

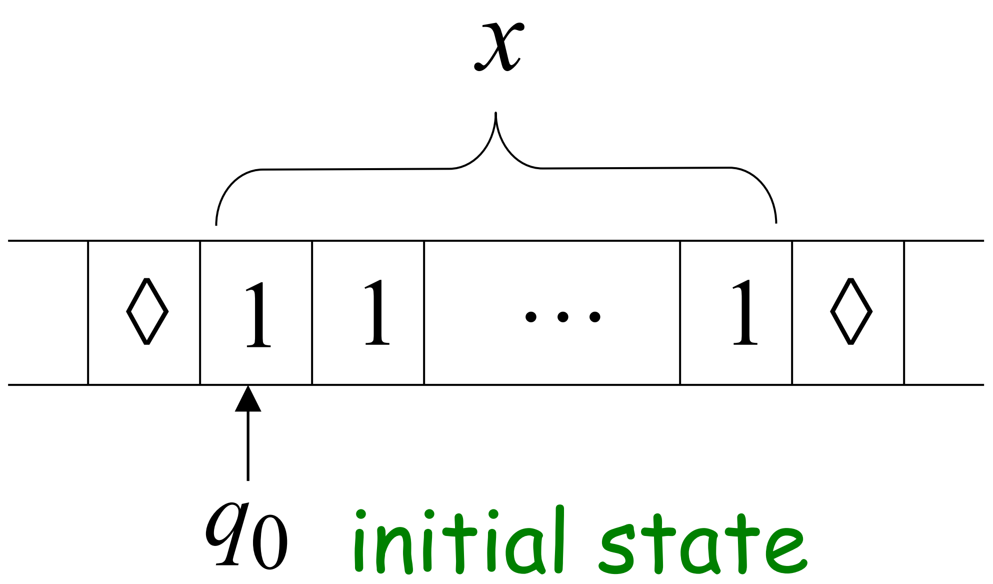
x is integer

Turing Machine:

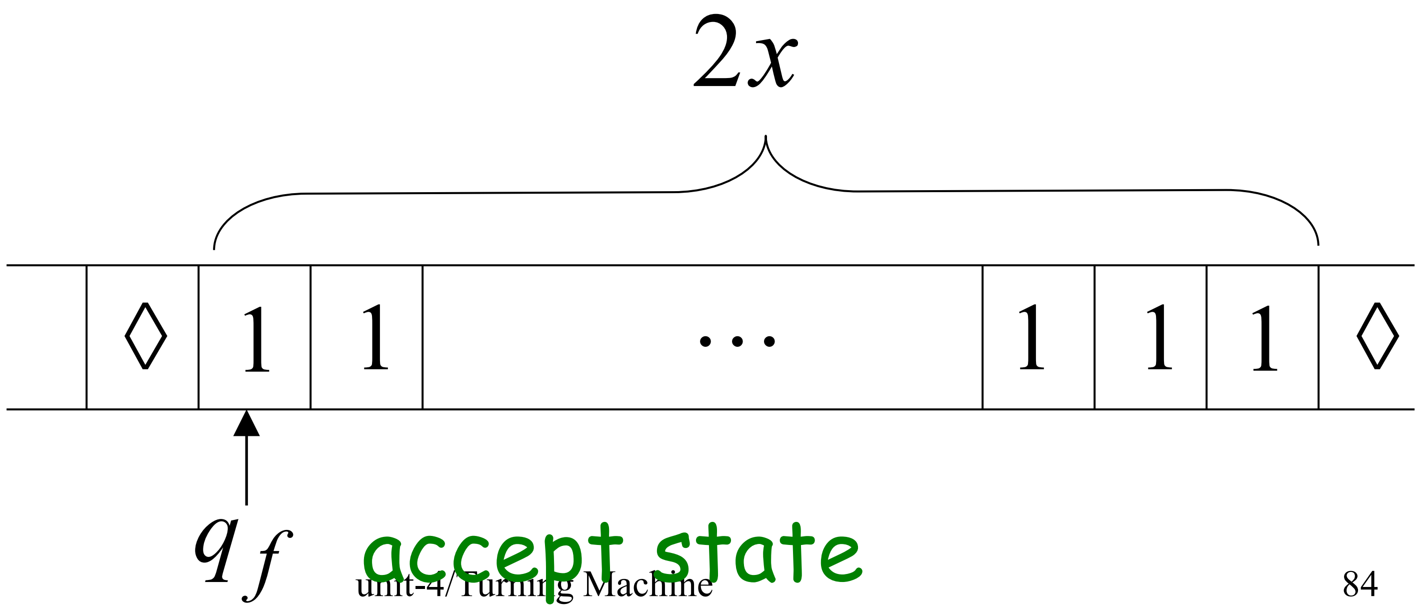
Input string: x unary

Output string: xx unary

Start



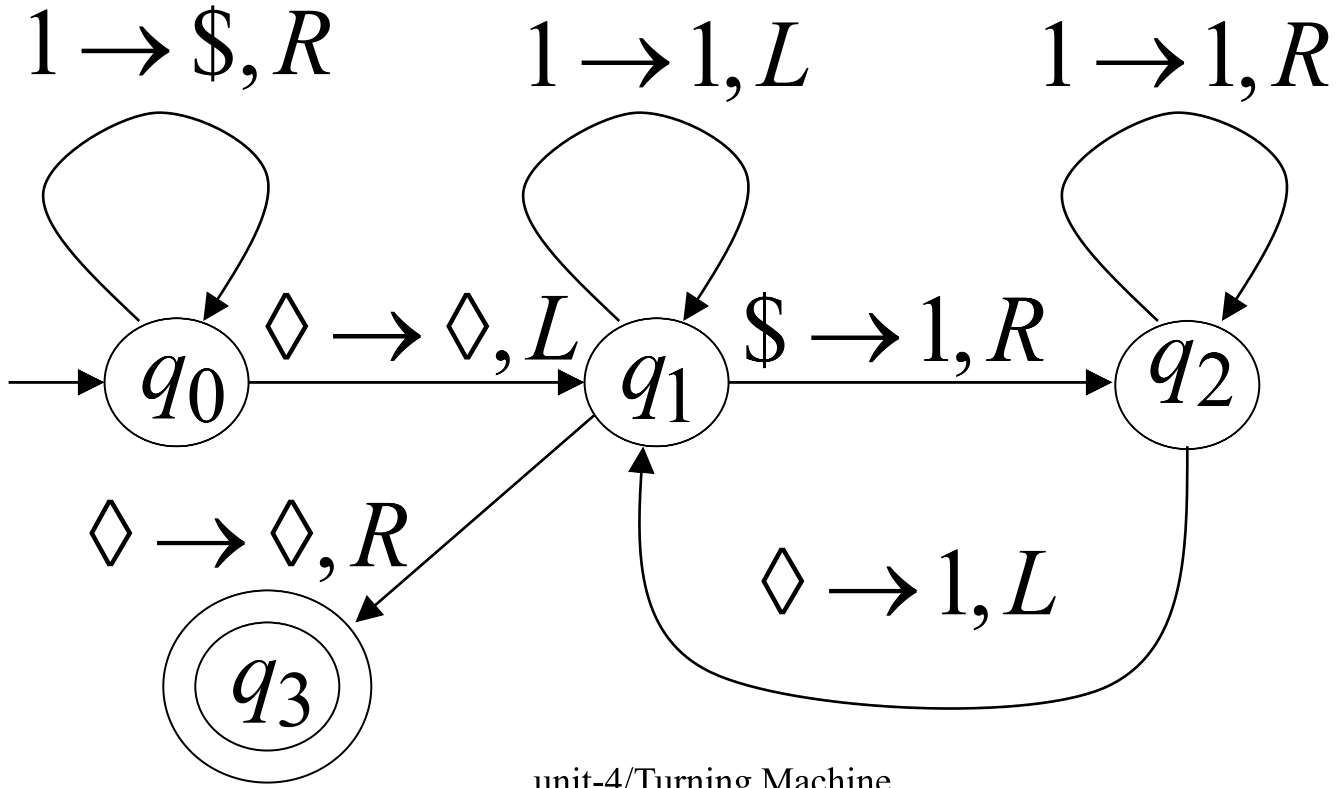
Finish



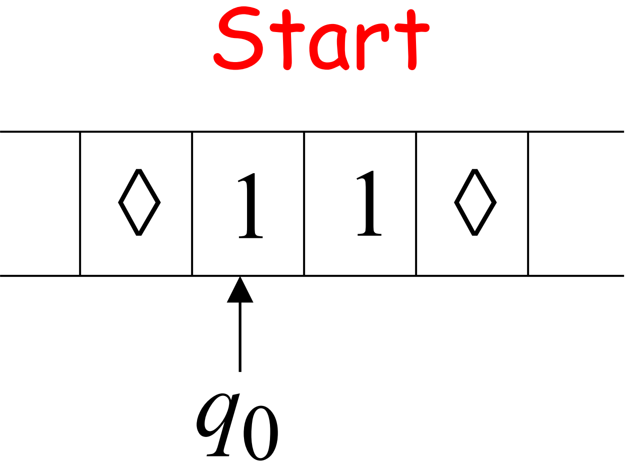
Turing Machine Pseudocode for $f(x) = 2x$

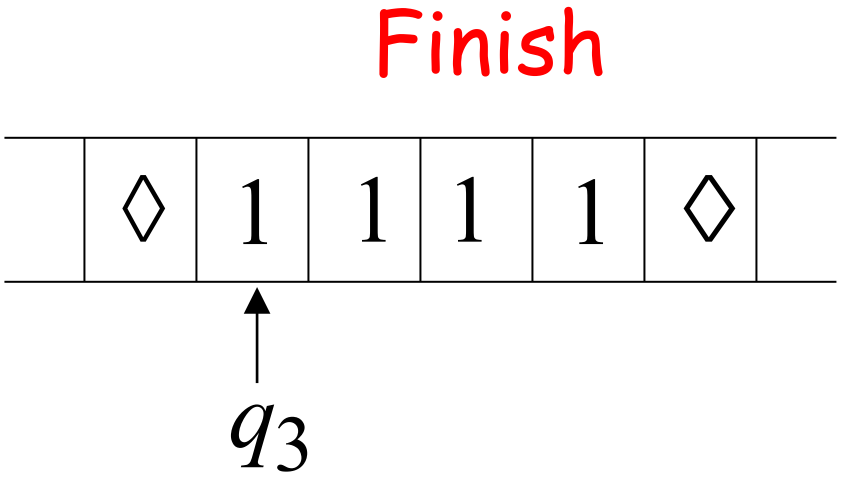
```
• Replace every 1 with $
• Repeat:
        • Find rightmost $, replace it with 1
        • Go to right end, insert 1
  Until no more $ remain
```

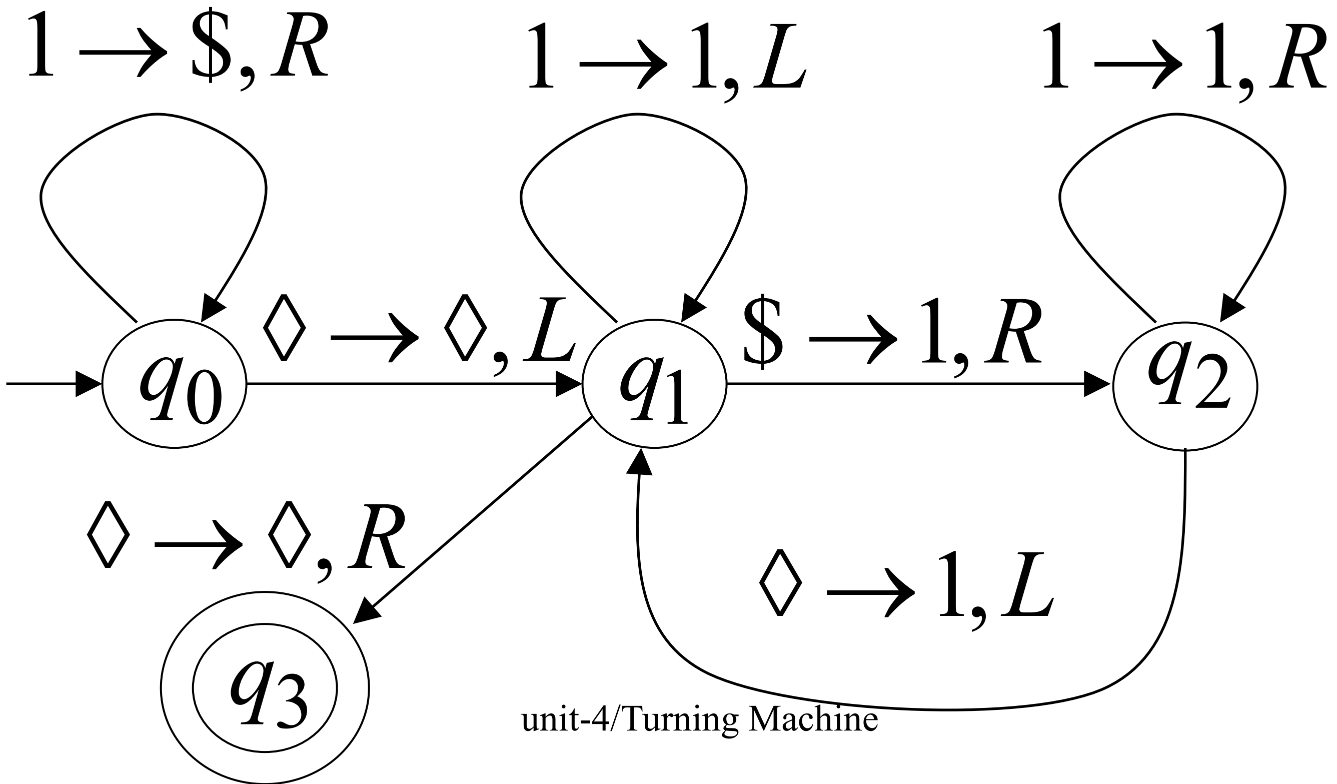
Turing Machine for $f(x) = 2x$


$$\begin{tikzcd}
\arrow[r] & q_0 \arrow[loop above, "1 \rightarrow \$, R"] \arrow[r, "\Diamond \rightarrow \Diamond, L"] & q_1 \arrow[loop above, "1 \rightarrow 1, L"] \arrow[r, "\$ \rightarrow 1, R"] \arrow[dl, "\Diamond \rightarrow \Diamond, R"'] & q_2 \arrow[loop above, "1 \rightarrow 1, R"] \arrow[l, bend left, "\Diamond \rightarrow 1, L"] \\
& q_3 & &
\end{tikzcd}$$

Example


Start
◊
1
1
◊
q_0


Finish
◊
1
1
1
1
◊
q_3


$1 \rightarrow \$, R$
$1 \rightarrow 1, L$
$1 \rightarrow 1, R$
$\Diamond \rightarrow \Diamond, L$
$\$ \rightarrow 1, R$
$\Diamond \rightarrow \Diamond, R$
$\Diamond \rightarrow 1, L$
q_0
q_1
q_2
q_3

Back to binary!

- How do we add two binary numbers on a Turing machine?
- Instead of dealing with it directly, let us divide the problem into 2 parts
  - How to subtract 1 from a binary number
  - How to add 1 to a binary number
- HW – think about how to do this without subroutines/functions/methods

How to add 1 to a binary number?

- What is 11000 + 1 = ?
- What is 10111 + 1 = ?
- The pattern that emerges is …..

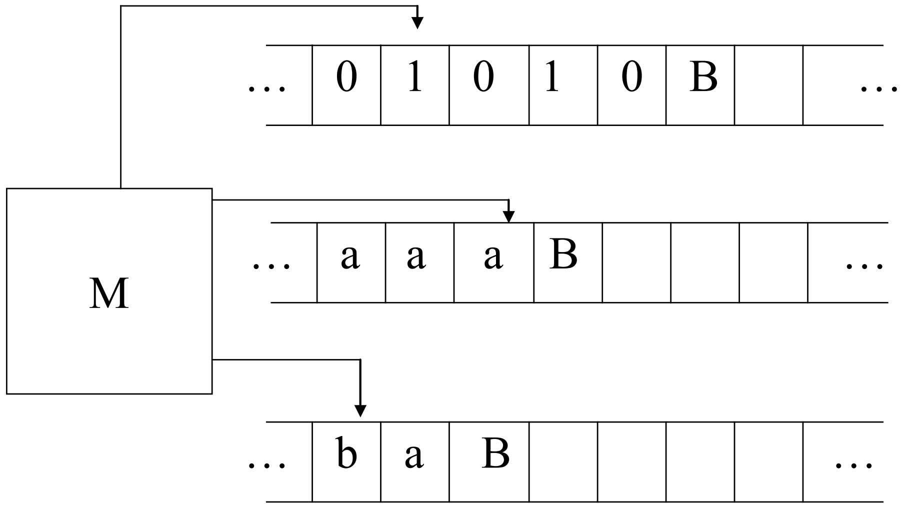
Multitape Turing Machines

- A multitape Turing machine is like an ordinary TM but it has several tapes instead of one tape.
- Initially the input starts on tape 1 and the other tapes are blank.
- The transition function is changed to allow for reading, writing, and moving the heads on all the tapes simultaneously.
  - This means we could read on multiples tape and move in different directions on each tape as well as write a different symbol on each tape, all in one move.

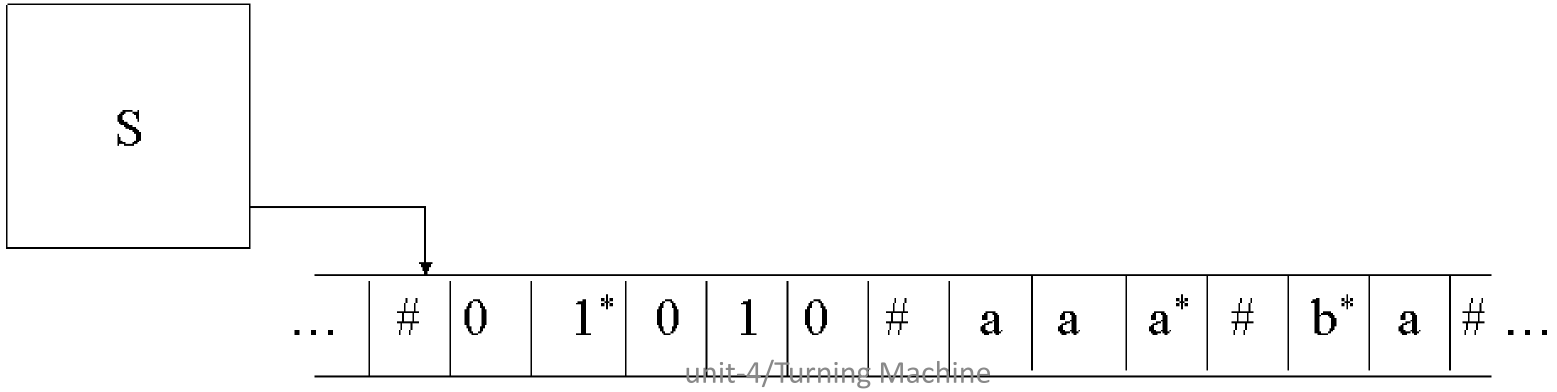
Multitape Turing Machine

- Theorem: A multitape TM is equivalent in power to an ordinary TM. Recall that two TM's are equivalent if they recognize the same language. We can show how to convert a multitape TM, M, to a single tape TM, S:
- Say that M has k tapes.
  - Create the TM S to simulate having k tapes by interleaving the information on each of the k tapes on its single tape
  - Use a new symbol # as a delimiter to separate the contents of each tape
  - S must also keep track of the location on each of the simulated heads
    - Write a type symbol with a * to mark the place where the head on the tape would be
    - The * symbols are new tape symbols that don't exist with M
    - The finite control must have the proper logic to distinguish say, x* and x and realize both refer to the same thing, but one is the current tape symbol.

Multitape Machine



Equivalent Single Tape Machine:



Nondeterministic TM

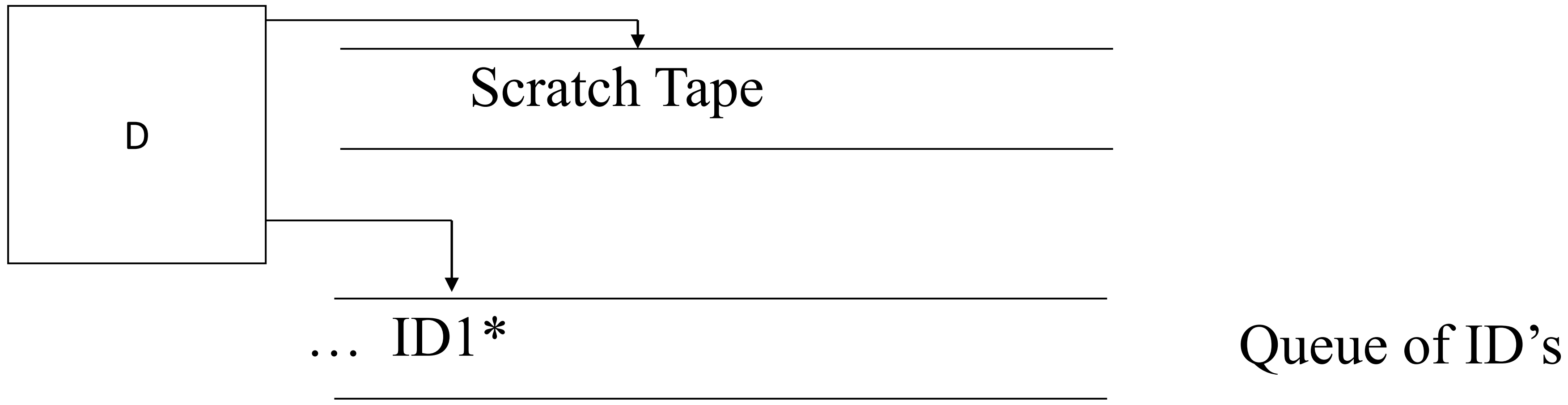
- Replace the “DFA” part of the TM with an “NFA”
  - Each time we make a nondeterministic move, you can think of this as a branch or “fork” to two simultaneously running machines. Each machine gets a copy of the entire tape. If any one of these machines ends up in an accepting state, then the input is accepted.
- Although powerful, nondeterminism does not affect the power of the TM model
- Theorem: Every nondeterministic TM has an equivalent deterministic TM.
  - We can prove this theorem by simulating any nondeterministic TM, N, with a deterministic TM, D.

Nondeterministic TM

- Visualize N as a tree with branches whenever we fork off to two (or more) simultaneous machines.
  - Use D to try all possible branches of N, sequentially.
  - If D ever finds the accept state on one of these branches, then D accepts.
  - It is quite likely that D will never terminate in the event of a loop if there is no accepting state.
- Search be done in a breadth-first rather than depth-first manner.
  - An individual branch may extend to infinity, and if we start down this branch then D will be stuck forever when some other branch may accept the input.
- We can simulate N on D by a tape with a queue of ID's and a scratch tape for temporary storage.
  - Each ID contains all the moves we have made from one state to the next, for one "branch" of the nondeterministic tree.
  - From the previous theorem, we can make as many multiple tapes as we like and this is still equivalent to a single tape machine. Initially, D looks like the following:

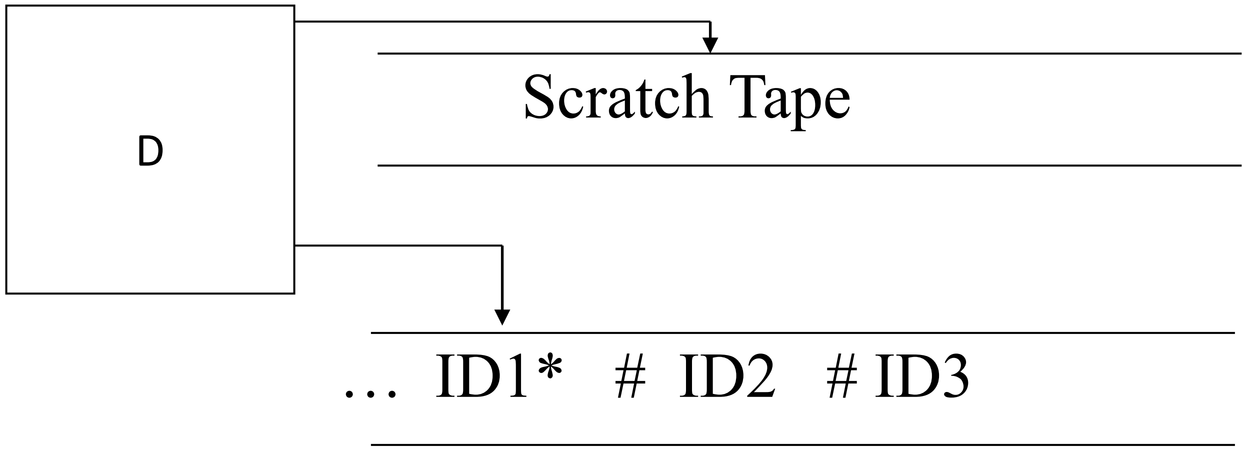
Nondeterministic TM

- ID1 is the sequence of moves we make from the start state. The * indicates that this is the current ID we are executing.
- We make a move on the TM. If this move results in a “fork” by following nondeterministic paths, then we create a new ID and copy it to the end of the queue using the scratch tape.



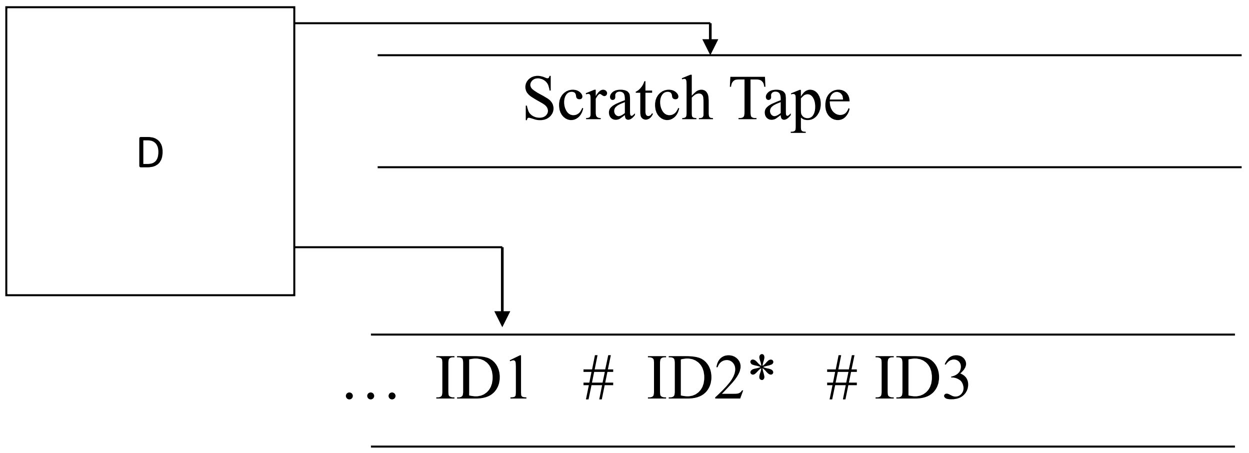
Nondeterministic TM

- For example, say that in ID1 we have two nondeterministic moves, resulting in ID2 and ID3:



Nondeterministic TM

- After we’re done with a single move in ID1, which may result in increasing the length of ID1 and storing it back to the tape, we move on to ID2:



Nondeterministic TM

- If any one of these states is accepting in an ID, then the machine quits and accepts.
- If we ever reach the last ID, then we repeat back with the first ID.

- Note that although the constructed deterministic TM is equivalent to accepting the same language as a nondeterministic TM, the deterministic TM might take exponentially more time than the nondeterministic TM.
  - It is unknown if this exponential slowdown is necessary. We'll come back to this in the discussion of P vs. NP.

- Theorem: Since any deterministic Turing Machine is also nondeterministic (there just happens to be no nondeterministic moves), there exists a nondeterministic TM for every deterministic TM.

Equivalence of TM's and Computers

- In one sense, a real computer has a finite amount of memory, and thus is **weaker** than a TM.
- But, we can postulate an infinite supply of tapes, disks, or some peripheral storage device to simulate an infinite TM tape. Additionally, we can assume there is a human operator to mount disks, keep them stacked neatly on the sides of the computer, etc.
- Need to show both directions, a TM can simulate a computer and that a computer can simulate a TM

THANK YOU!!!