Turing Machine

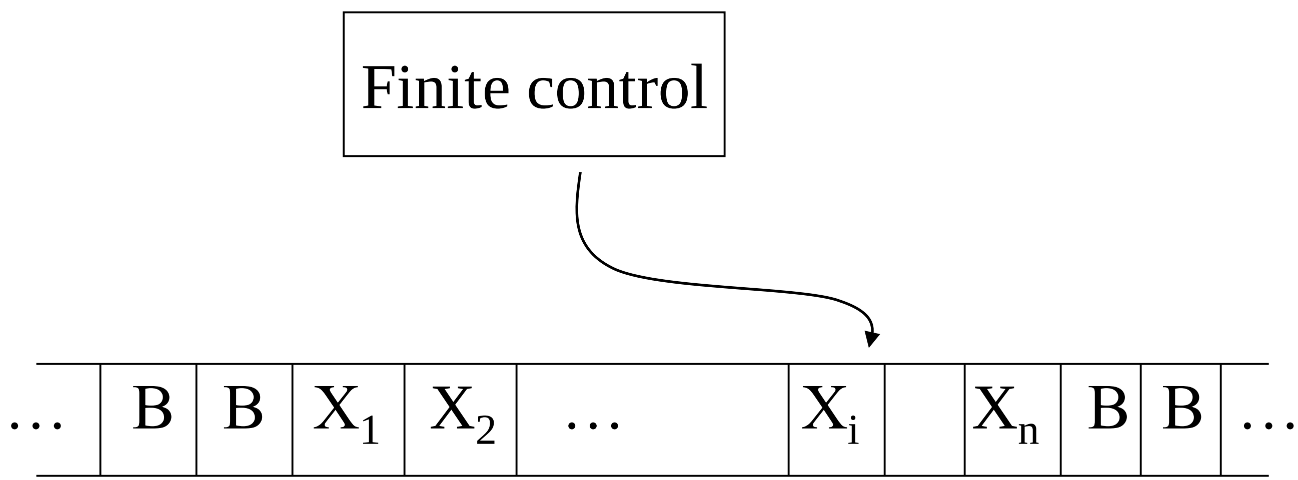
- Definition
- TM as language acceptors
- Combining Turing Machines
- Computing partial function with a TM
- Multi-tape TMs
- Universal TM

Turing Machines

- 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.



Definition 7.1 Turing Machines

A Turing machine (TM) is a 5-tuple $T = (Q, \Sigma, \Gamma, q_0, \delta)$, where

Q is a finite set of states. The two *halt* states h_a and h_r are not elements of Q.

Σ, the input alphabet, and Γ, the tape alphabet, are both finite sets, with $\Sigma \subseteq \Gamma$. The *blank* symbol Δ is not an element of Γ.

q_0, the initial state, is an element of Q.

δ is the transition function:

$$\delta : Q \times (\Gamma \cup \{\Delta\}) \rightarrow (Q \cup \{h_a, h_r\}) \times (\Gamma \cup \{\Delta\}) \times \{\text{R}, \text{L}, \text{S}\}$$

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 5-tuple:
  - $M = (Q, \sum, \Gamma, \delta, q_0)$
  - $M = (Q, \sum, \Gamma, \delta, q_0, B/\$, h_a, h_r)$

Turing Machines and Halting

- One way for a TM to accept input is to end in a final state.
  - Another way is acceptance by halting. We say that a TM halts if it enters a state q, scanning a tape symbol X, and there is no move in this situation; i.e. $\delta(q,X)$ is undefined.
- Note that this definition of halting was not used in the transition diagram for the TM we described earlier; instead that TM died on unspecified input!

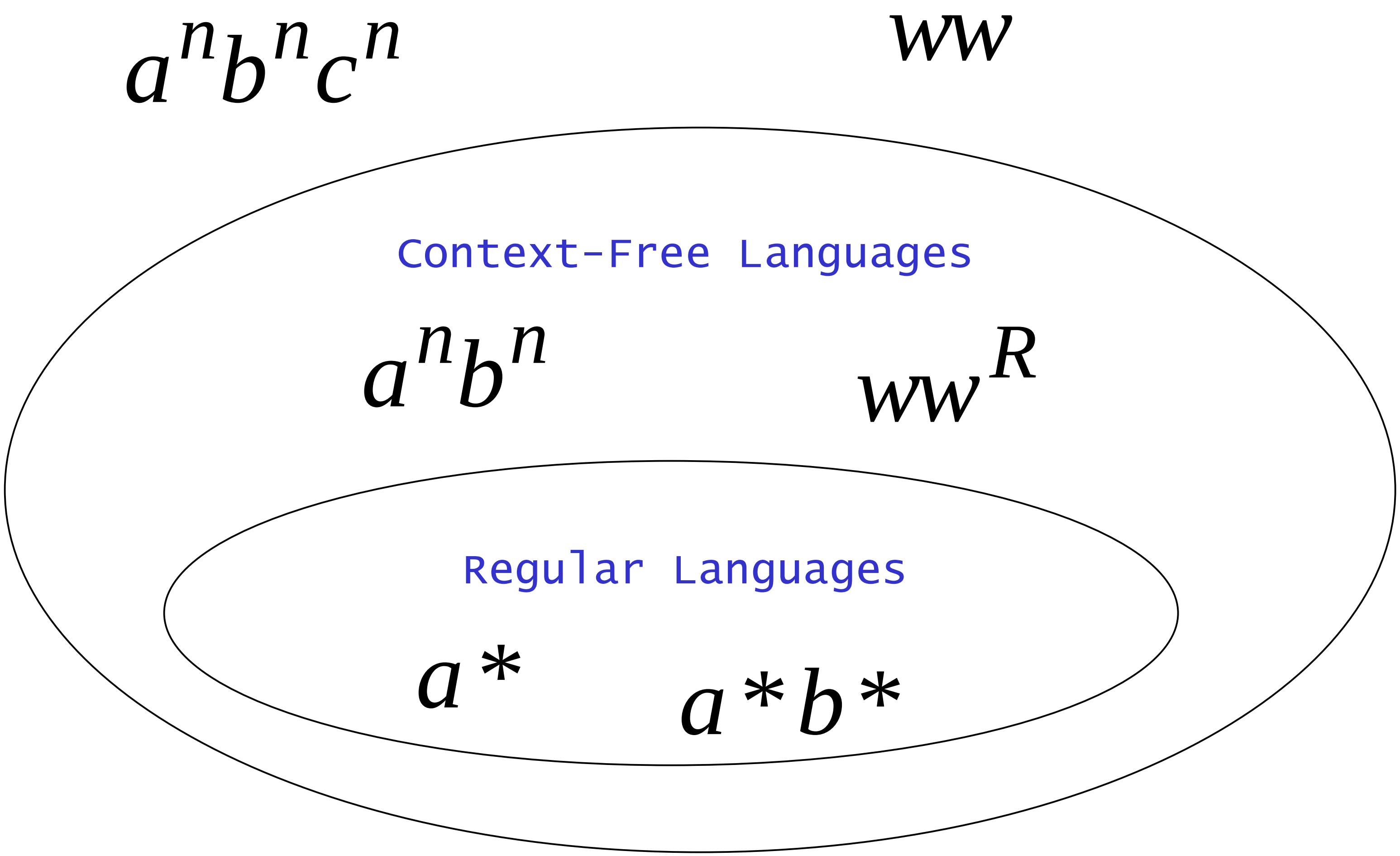
- It is possible to modify the prior example so that there is no unspecified input except for our accepting state. An equivalent TM that halts exists for a TM that accepts input via final state.

- In general, we assume that a TM always halts when it is in an accepting state.
- Unfortunately, it is not always possible to require that a TM halts even if it does not accept the input. Turing machines that always halt, regardless of accepting or not accepting, are good models of algorithms for decidable problems. Such languages are called *recursive*.

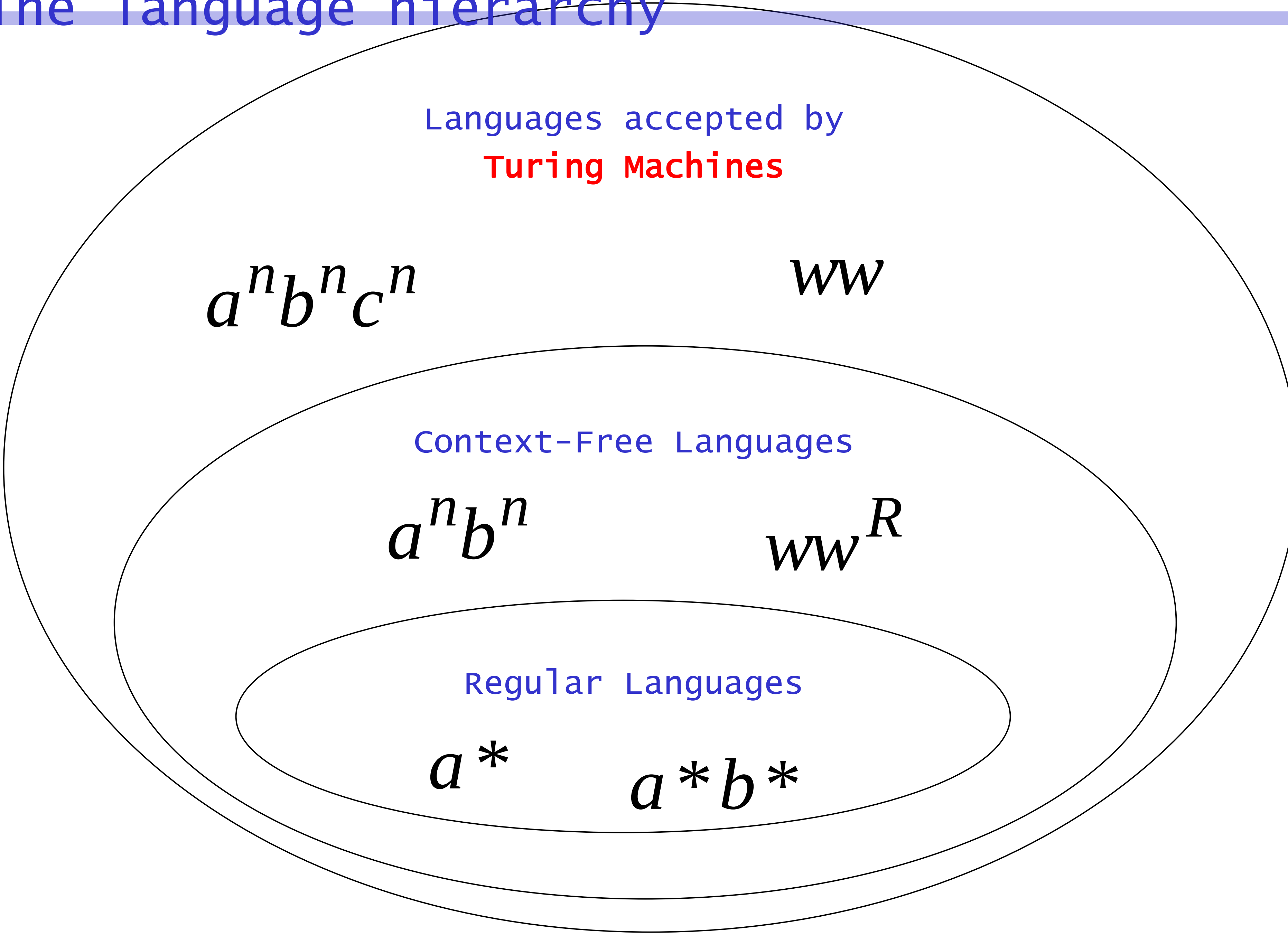
Turing Machine Variants

- There are many variations we can make to the basic TM
  - Extensions can make it useful to prove a theorem or perform some task
  - However, these extensions do not add anything extra the basic TM can't already compute
- Example: consider a variation to the Turing machine where we have the option of staying put instead of forcing the tape head to move left or right by one cell.
  - In the old model, we could replace each "stay put" move in the new machine with two transitions, one that moves right and one that moves left, to get the same behavior.

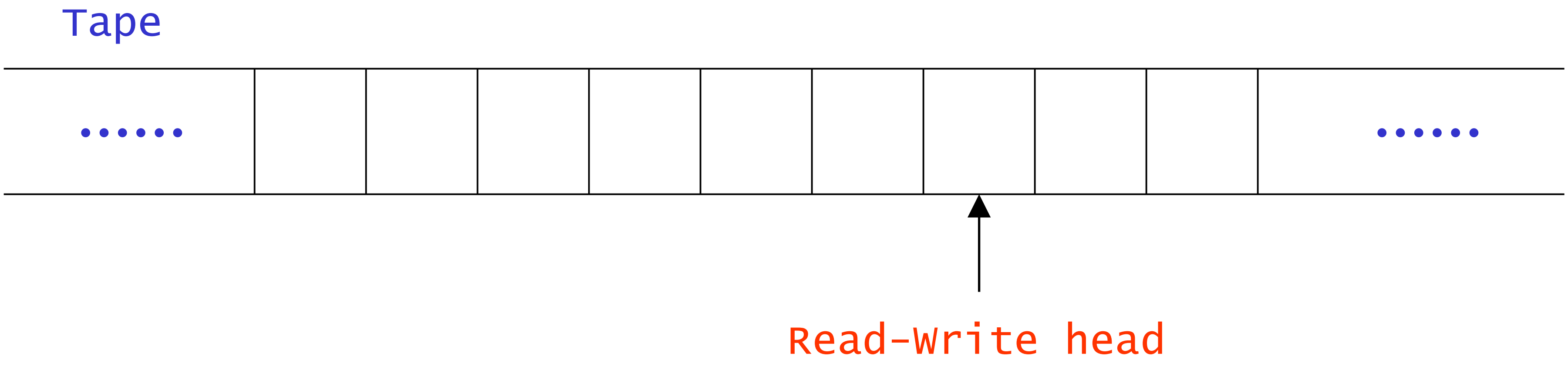
The language hierarchy

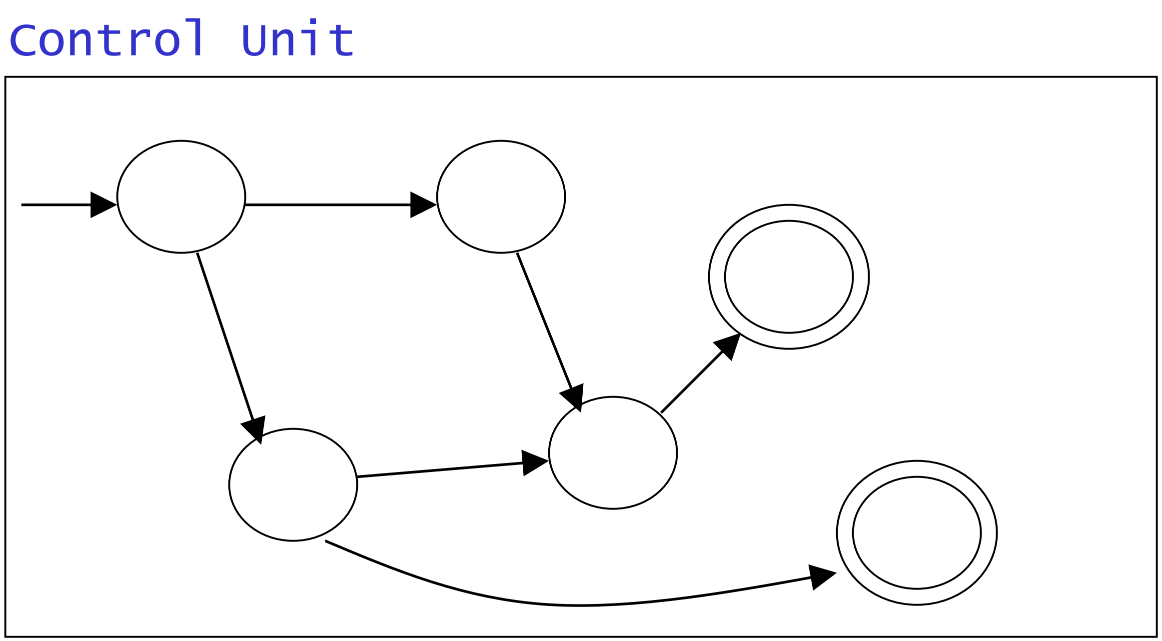


The language hierarchy



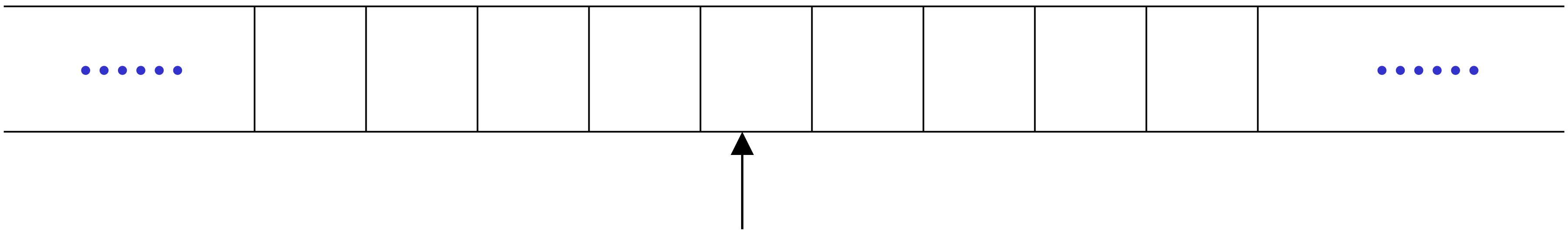
A Turing machine





The tape

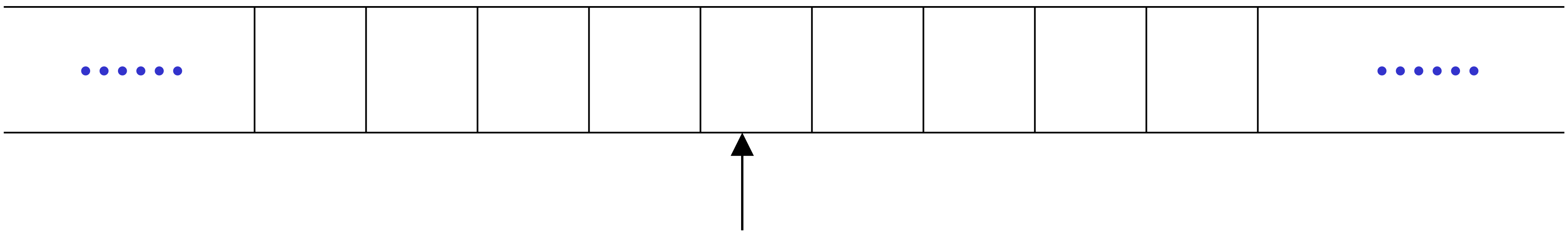
No boundaries -- infinite length



Read-Write head

The head moves Left or Right

The tape

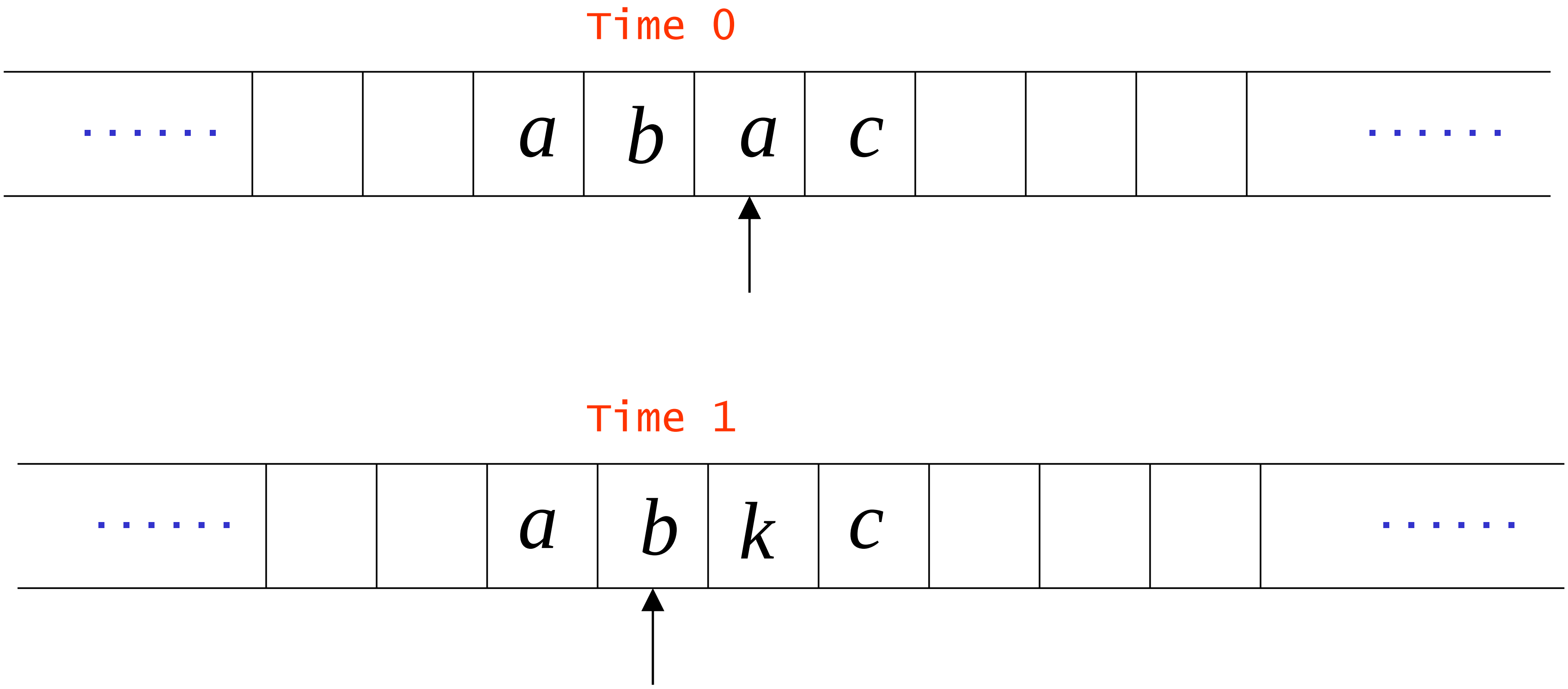


Read-Write head

The head at each time step:

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

Example



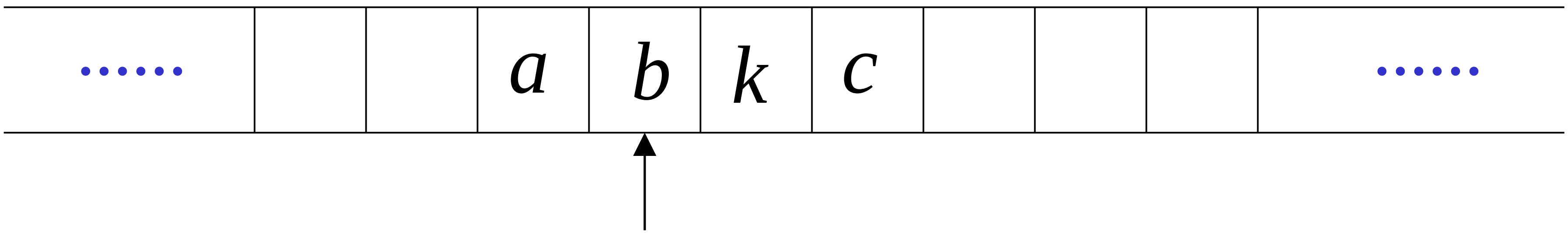
1. Reads a

2. Writes k

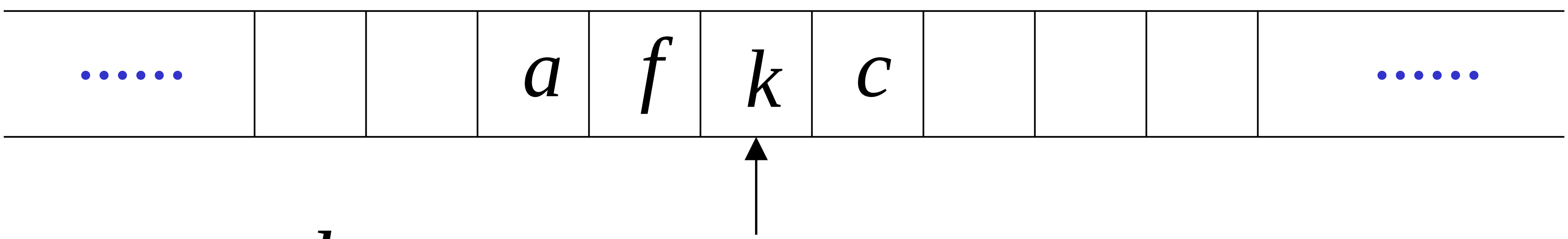
3. Moves Left

Example

Time 1



Time 2

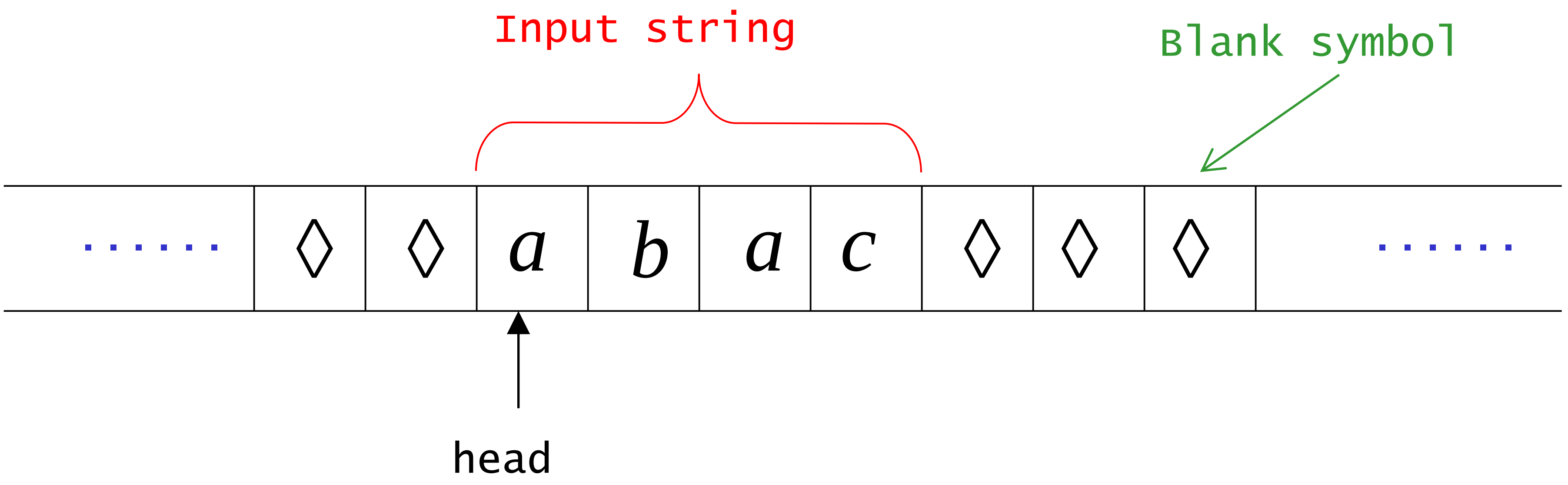


1. Reads b

2. Writes f

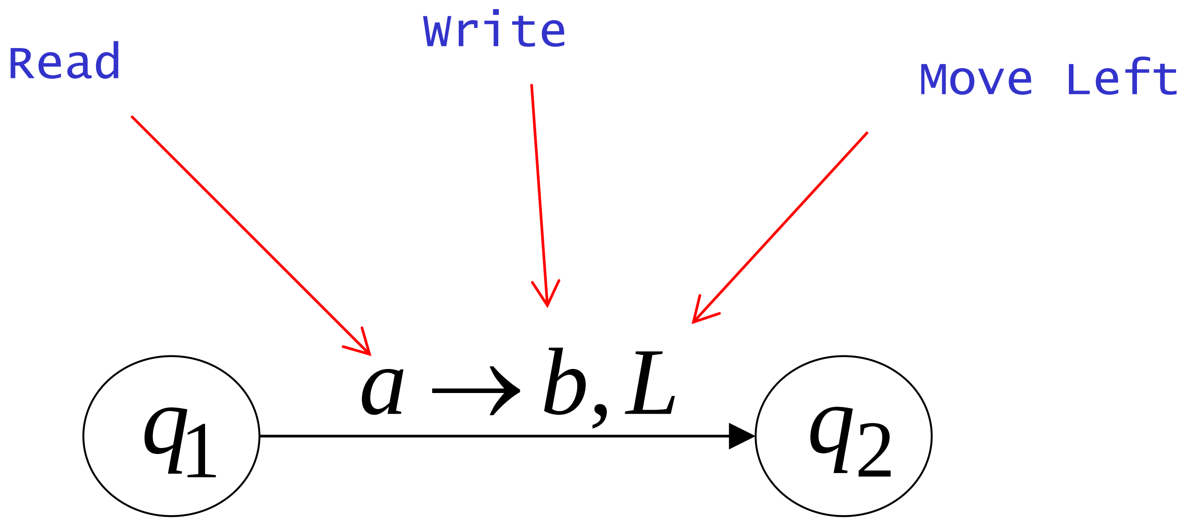
3. Moves Left

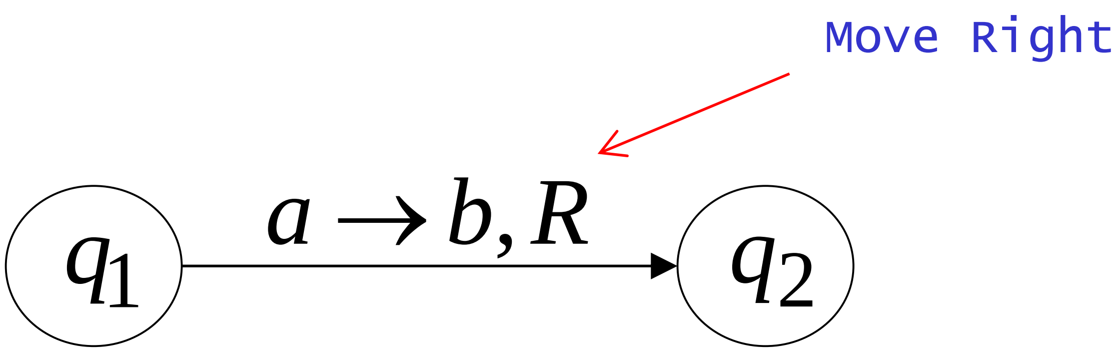
The input string



Head starts at the leftmost
position of the input string

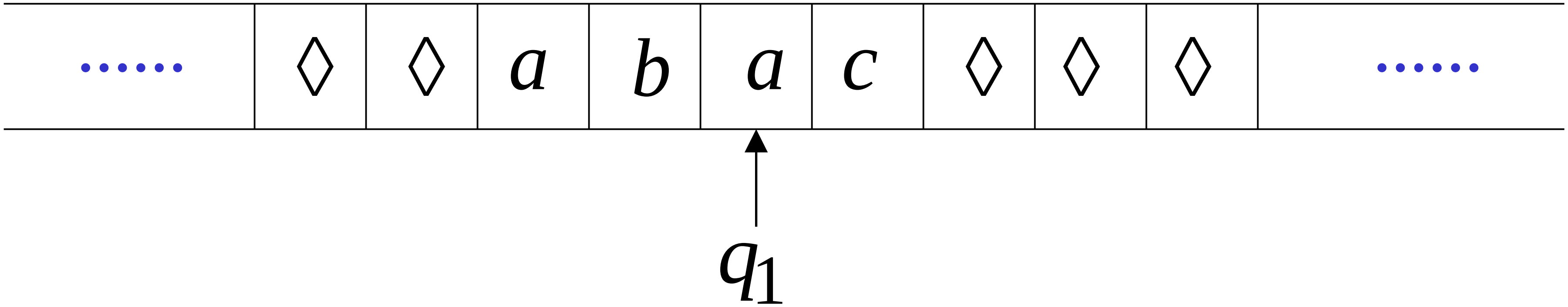
States and transitions



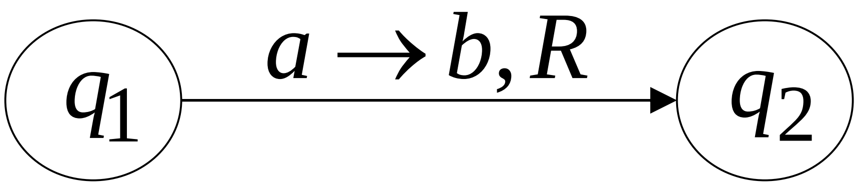


Example

Time 1

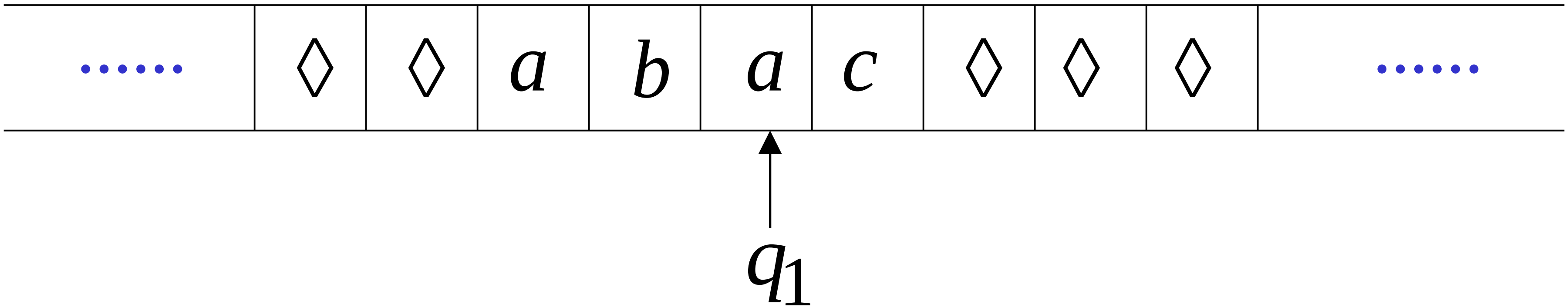


current state

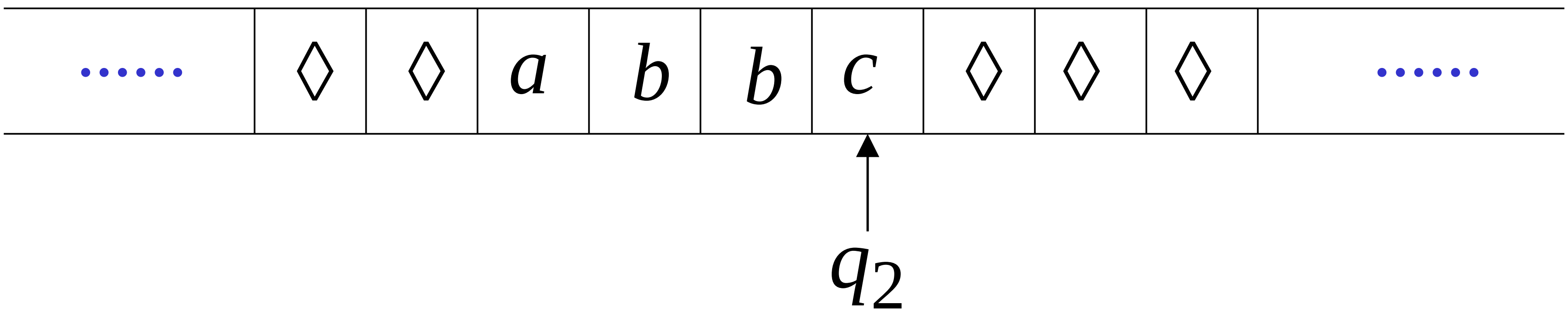


Example

Time 1

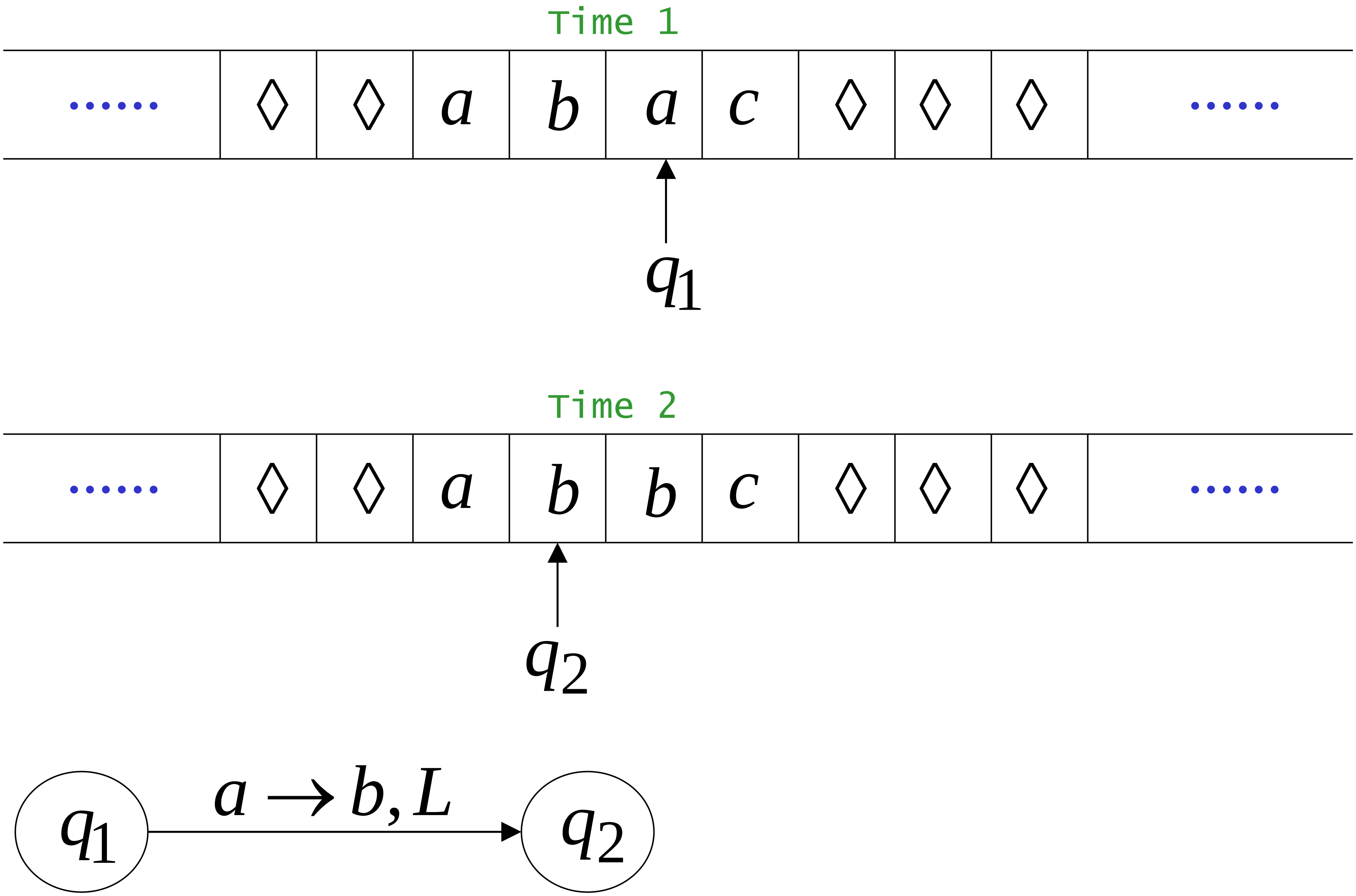


Time 2

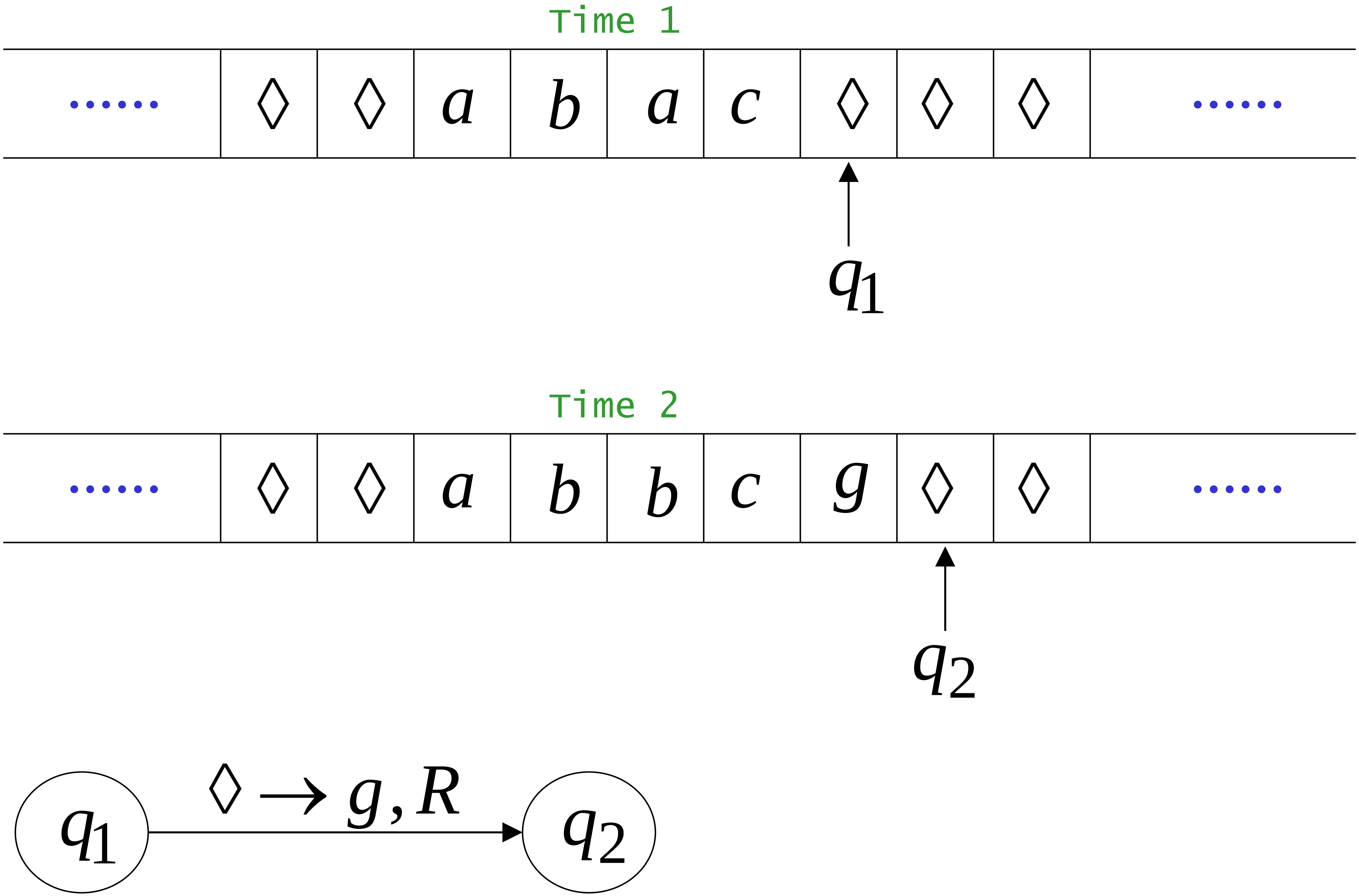


$$q_1 \xrightarrow{a \to b, R} q_2$$

Example



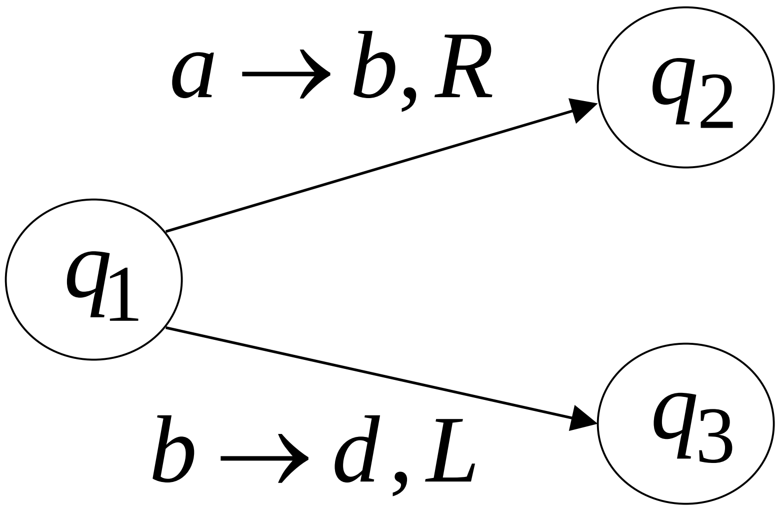
Example



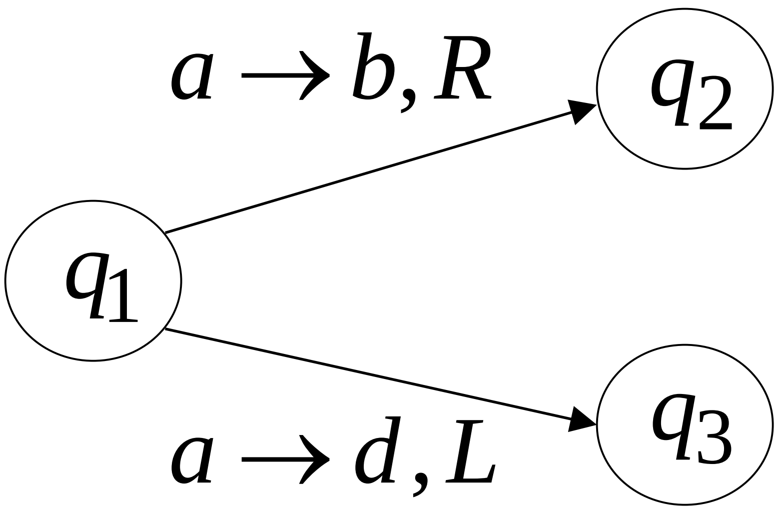
Determinism

Turing Machines are deterministic

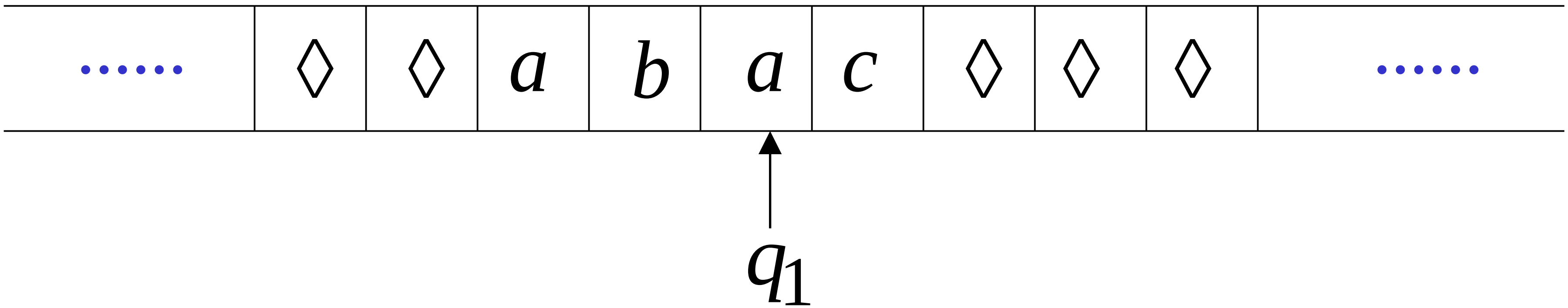
Allowed

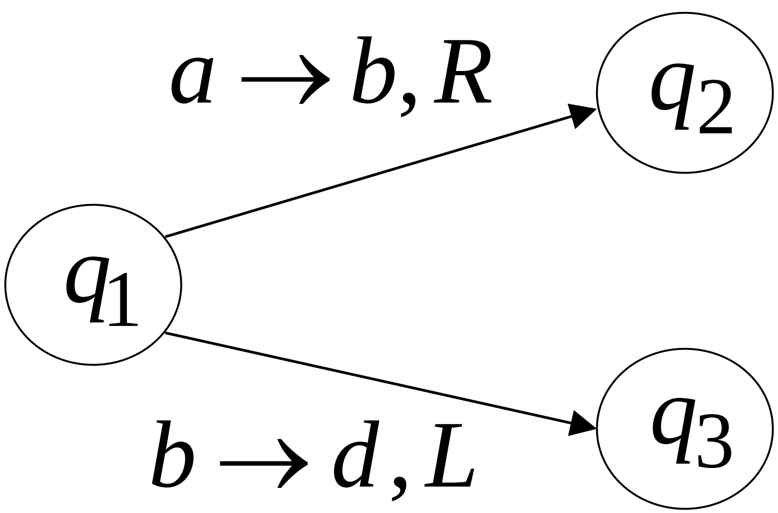


Not Allowed



Example: partial transition function





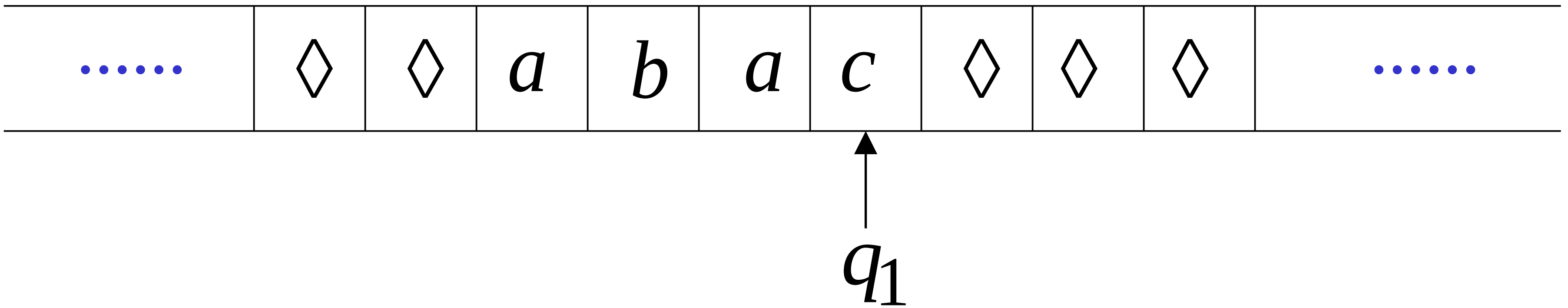
Allowed:

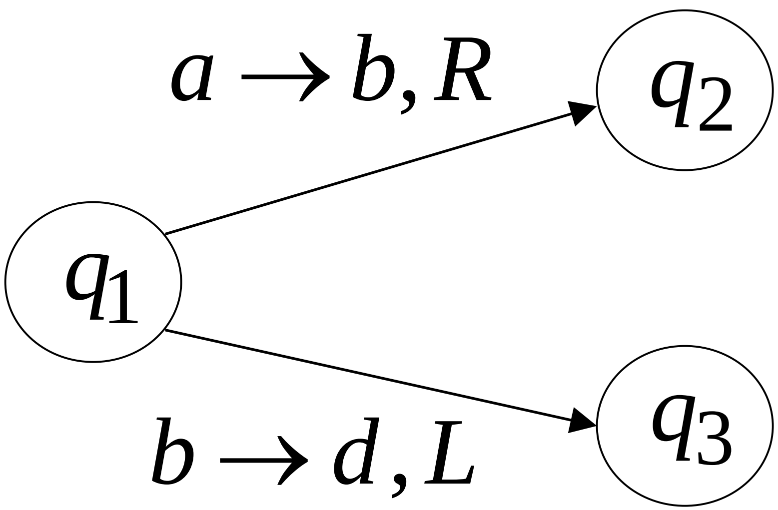
No transition
for input symbol c

Halting

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

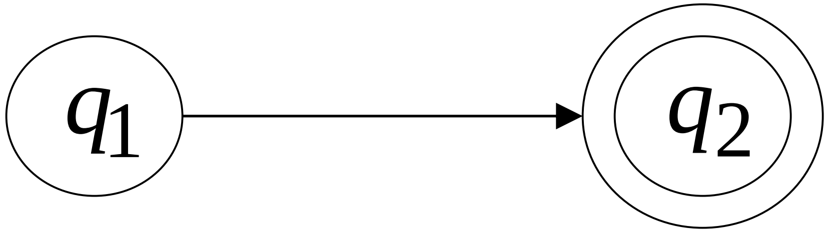
Example



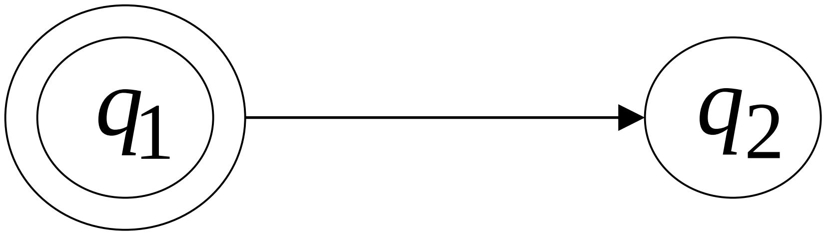


No possible transition **HALT**

Final states



Allowed



Not Allowed

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

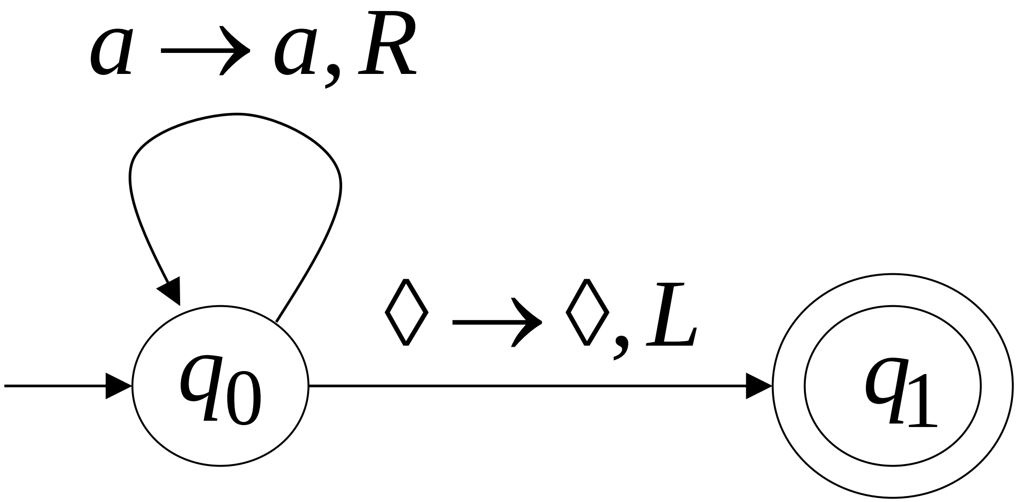
Acceptance

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*

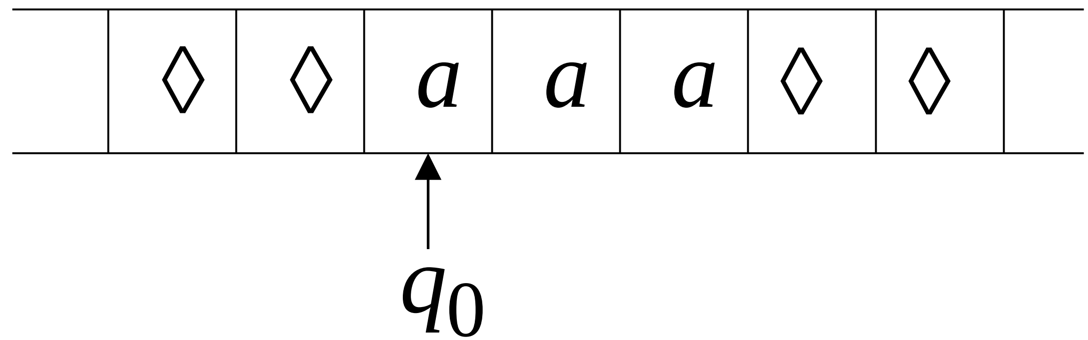
Turing machine example

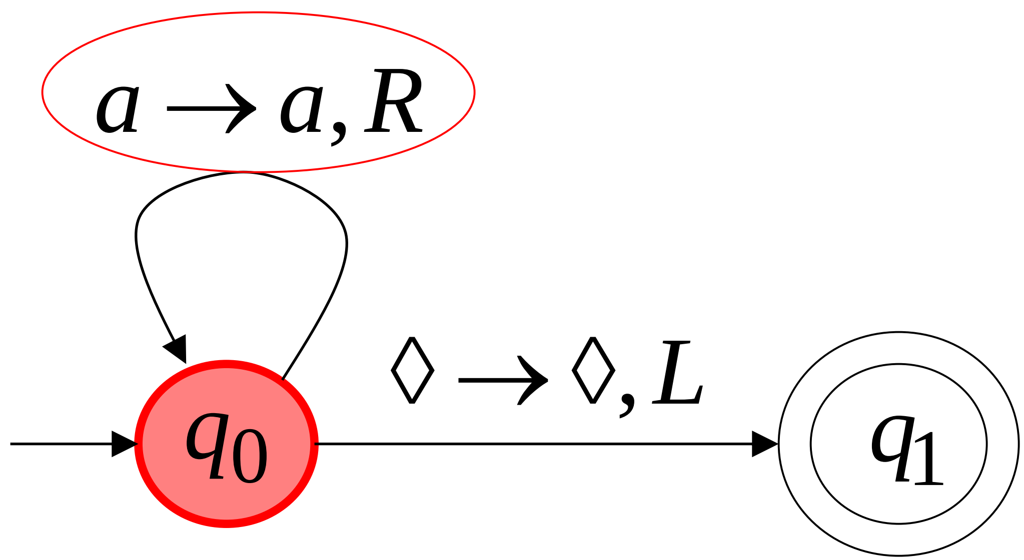
A Turing machine that accepts language a^*



Turing machine example

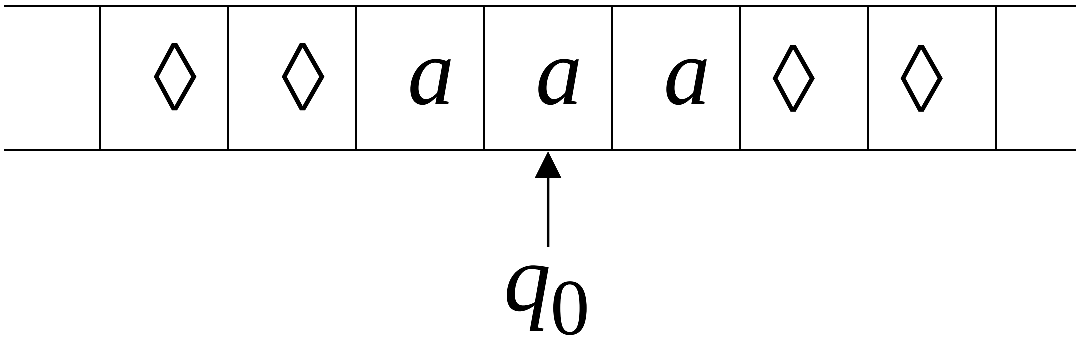
Time 0

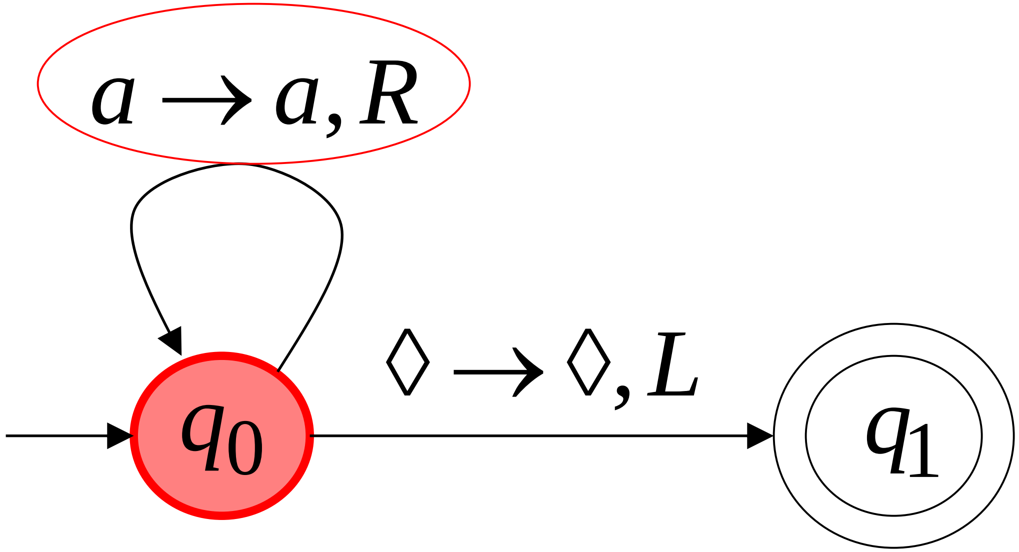




Turing machine example

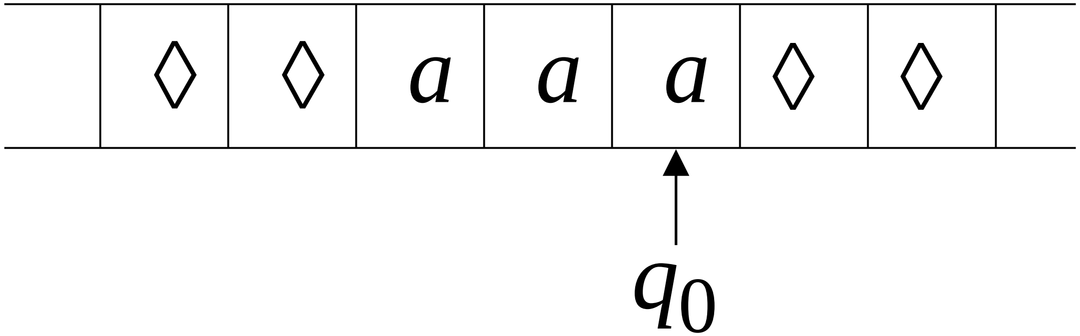
Time 1

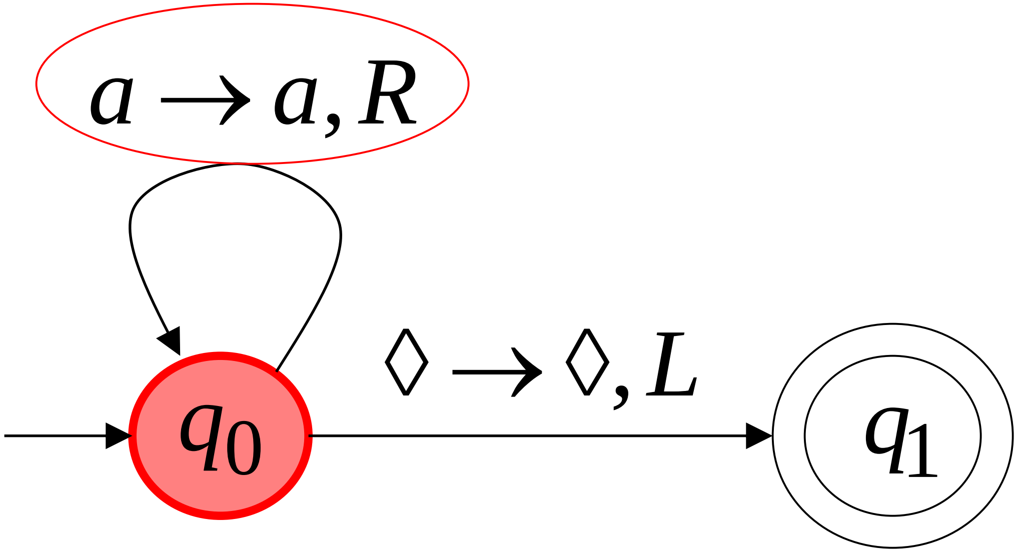




Turing machine example

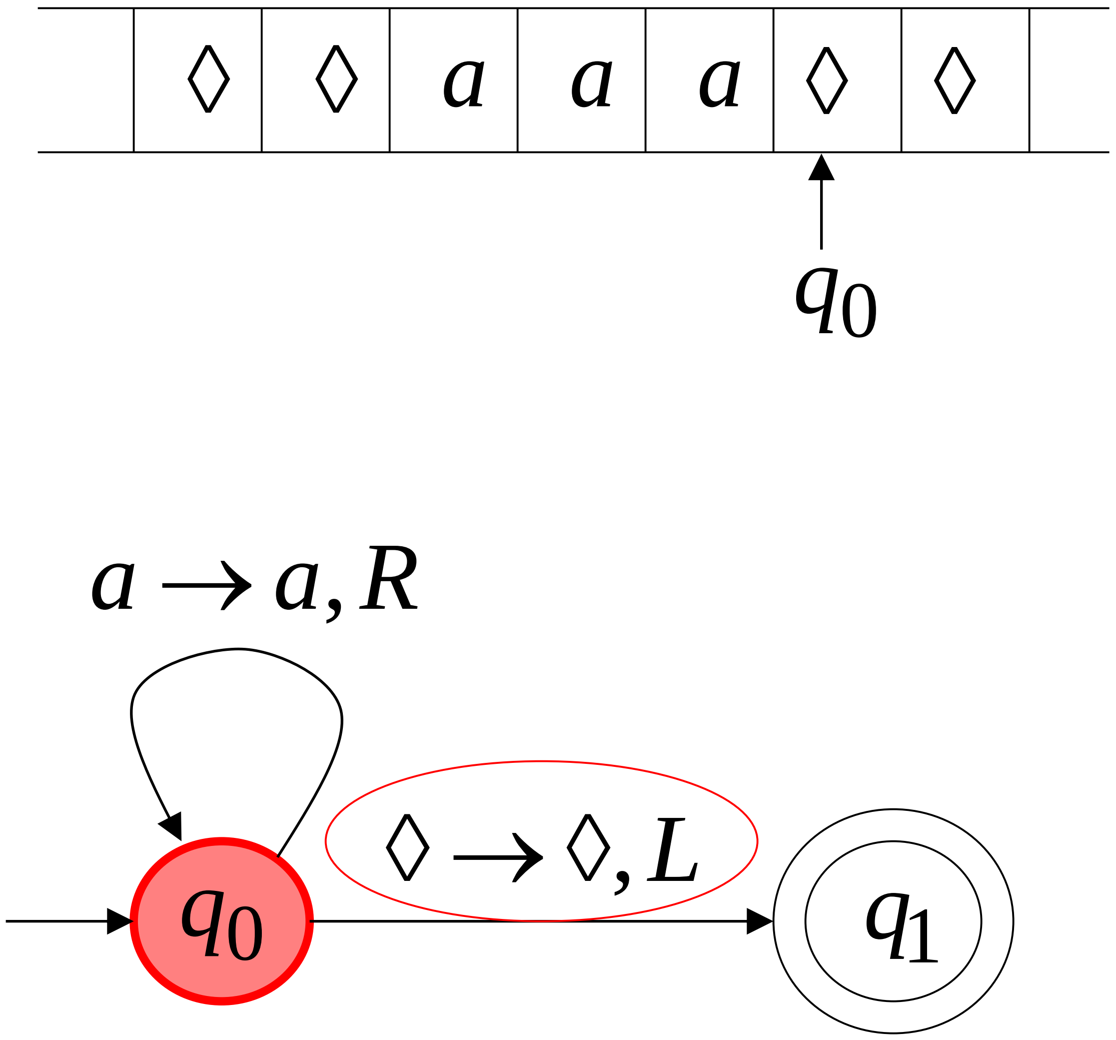
Time 2





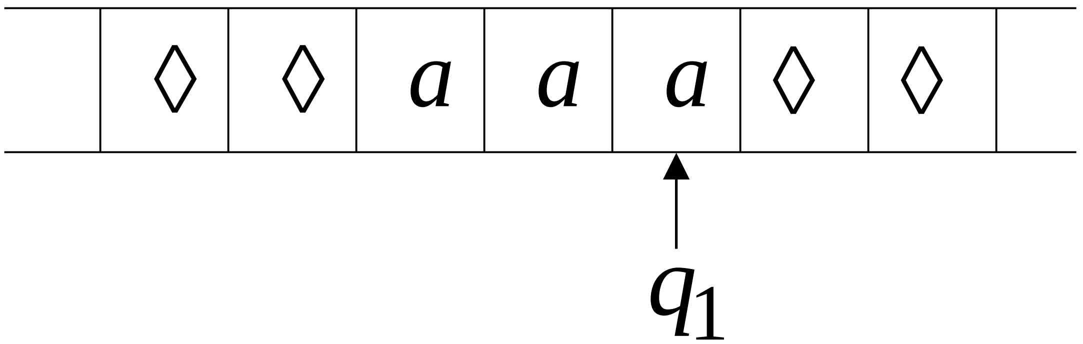
Turing machine example

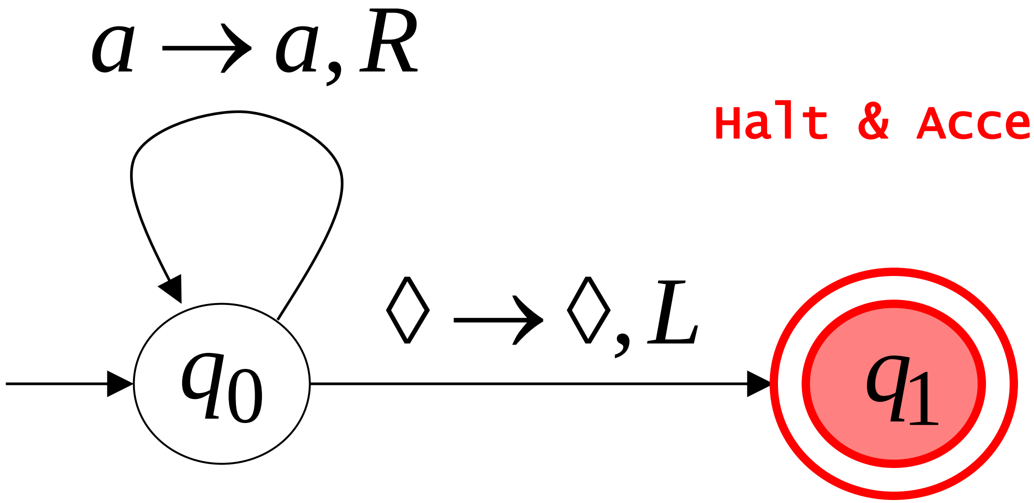
Time 3



Turing machine example

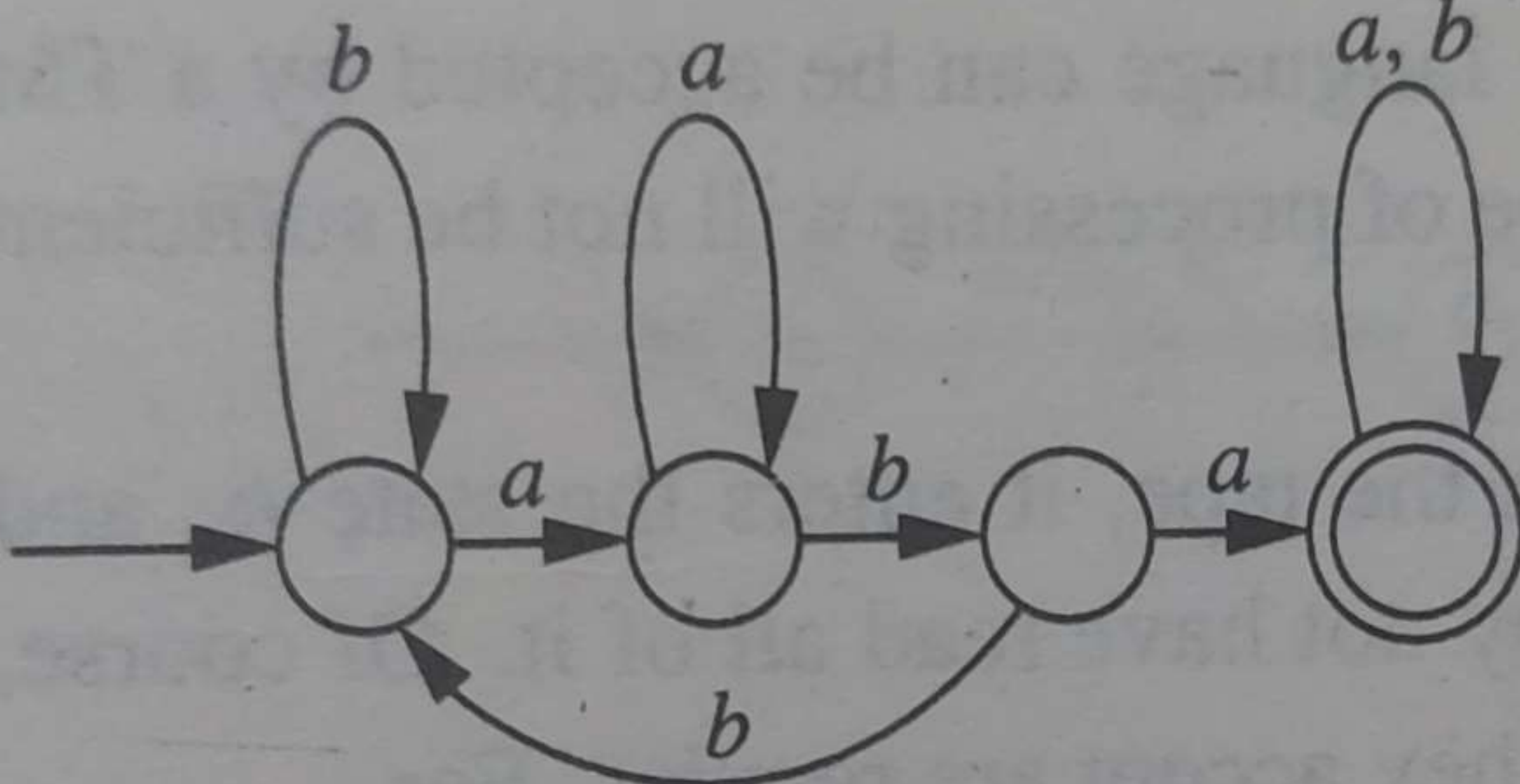
Time 4



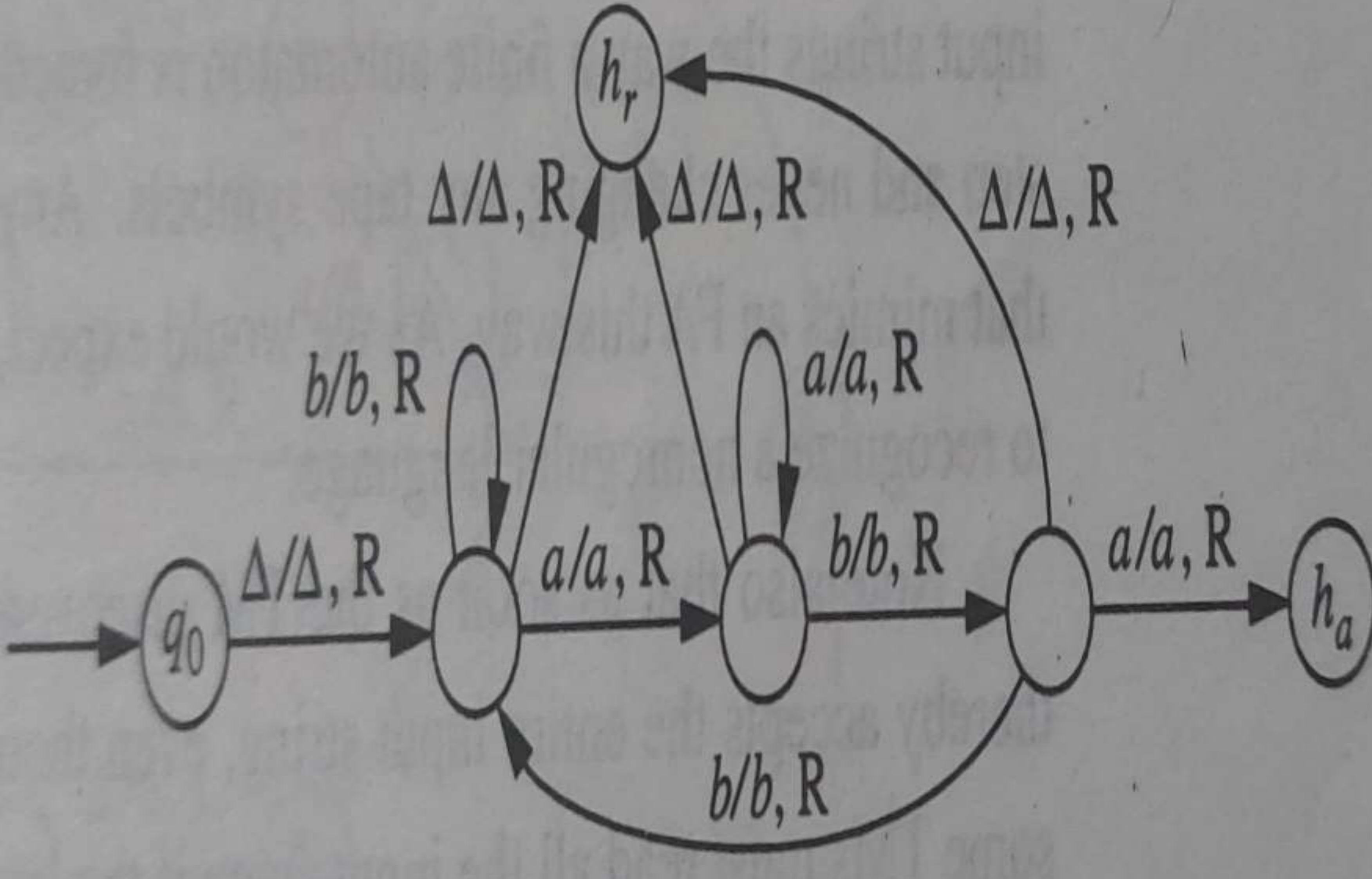


Turing Machine Example

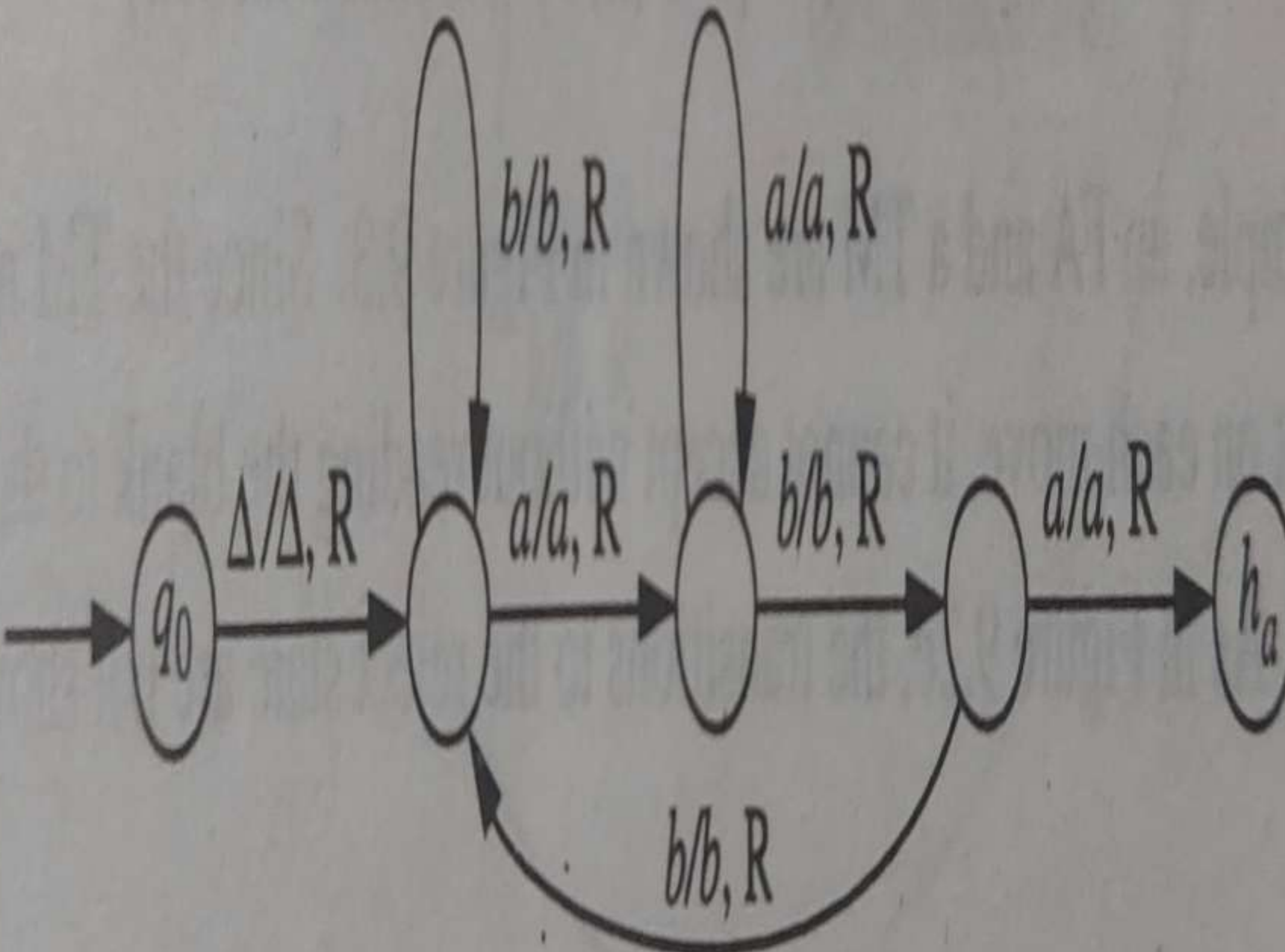
{a,b}*{aba}{a,b}*



(a)

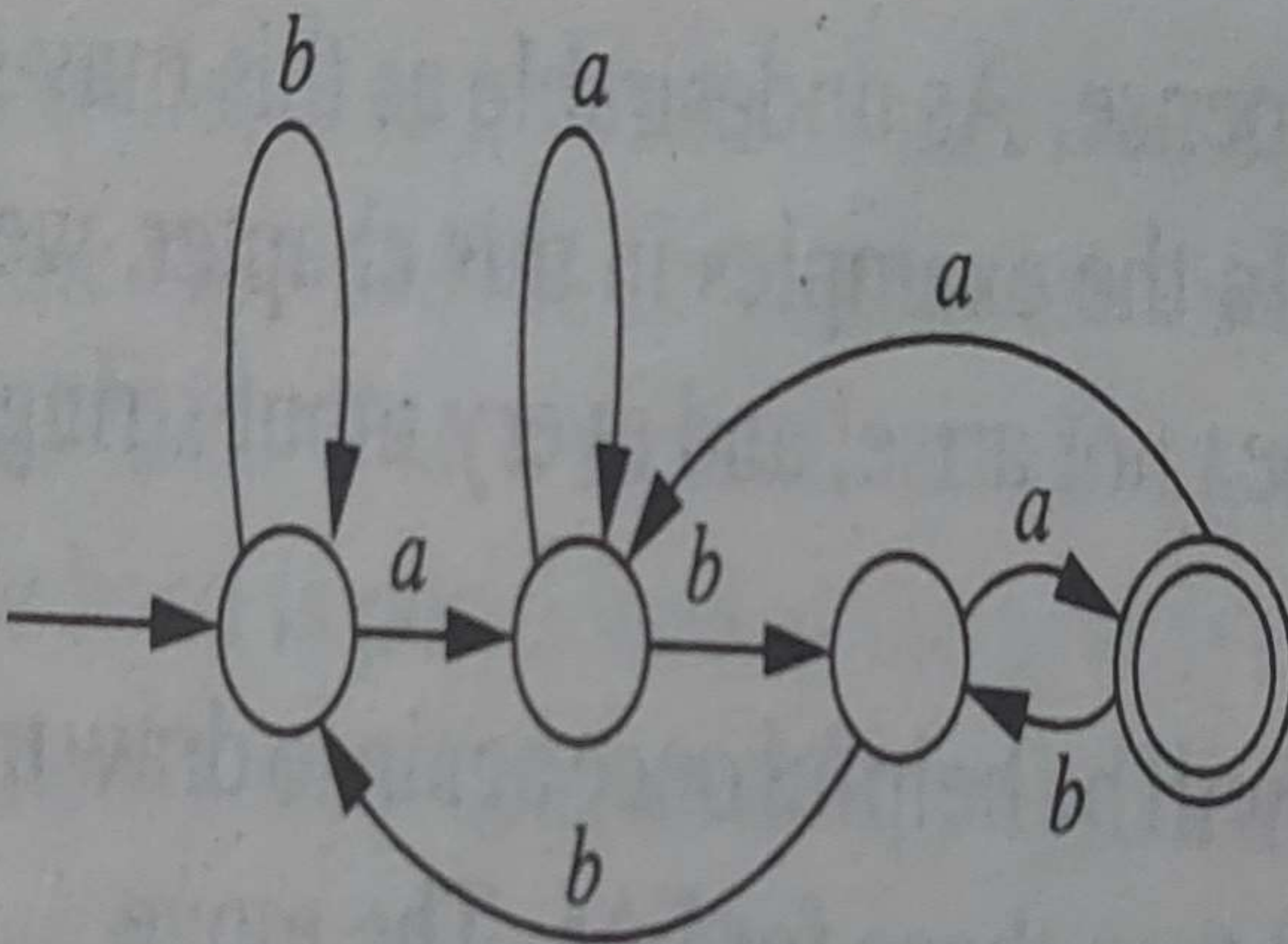

h_r
Δ/Δ, R
Δ/Δ, R
Δ/Δ, R
b/b, R
a/a, R
Δ/Δ, R
a/a, R
b/b, R
a/a, R
q_0
h_a
b/b, R

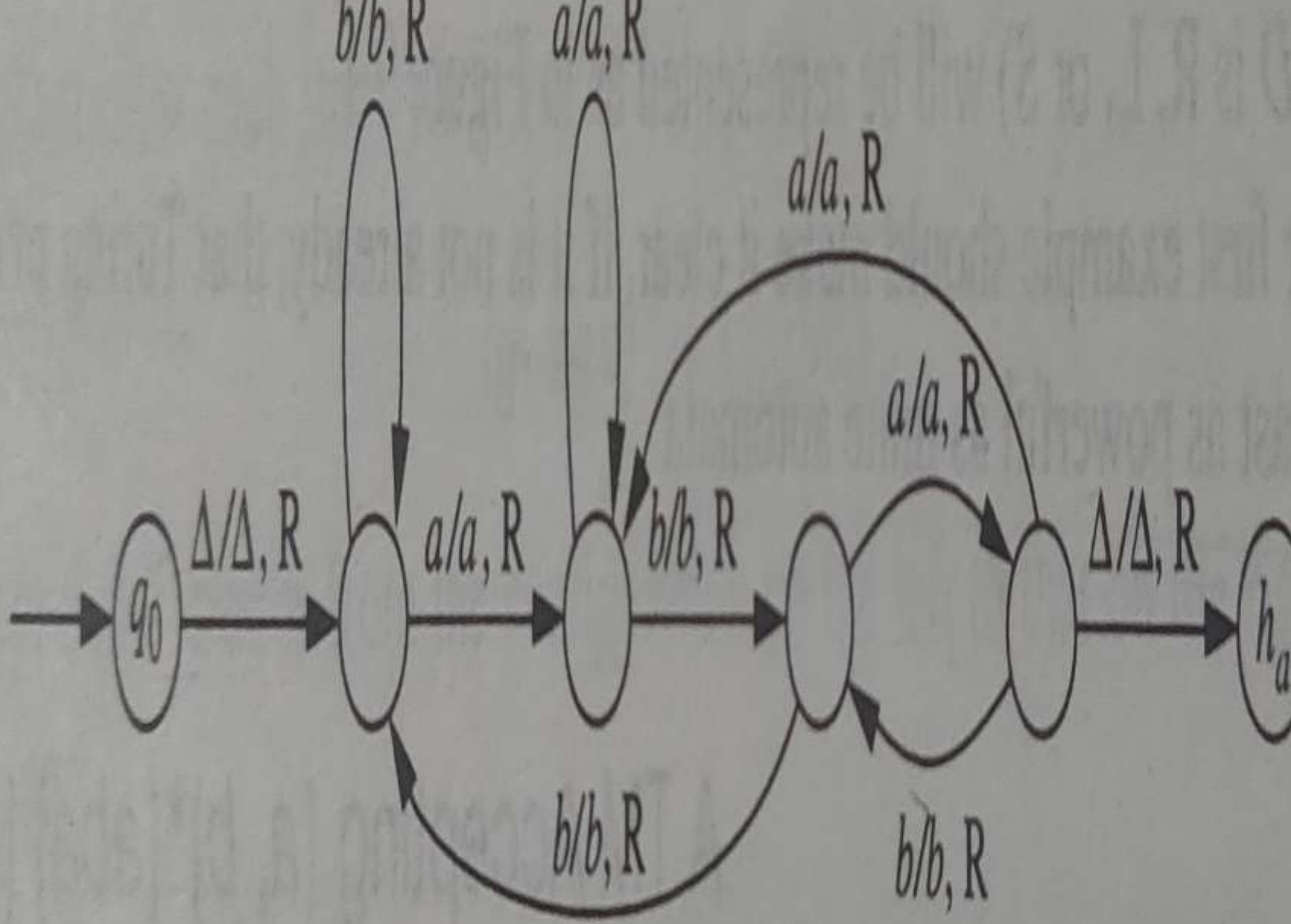
(*b*)


b/b, R
a/a, R
Δ/Δ, R
a/a, R
b/b, R
a/a, R
b/b, R
q_0
h_a

Turing Machine Example

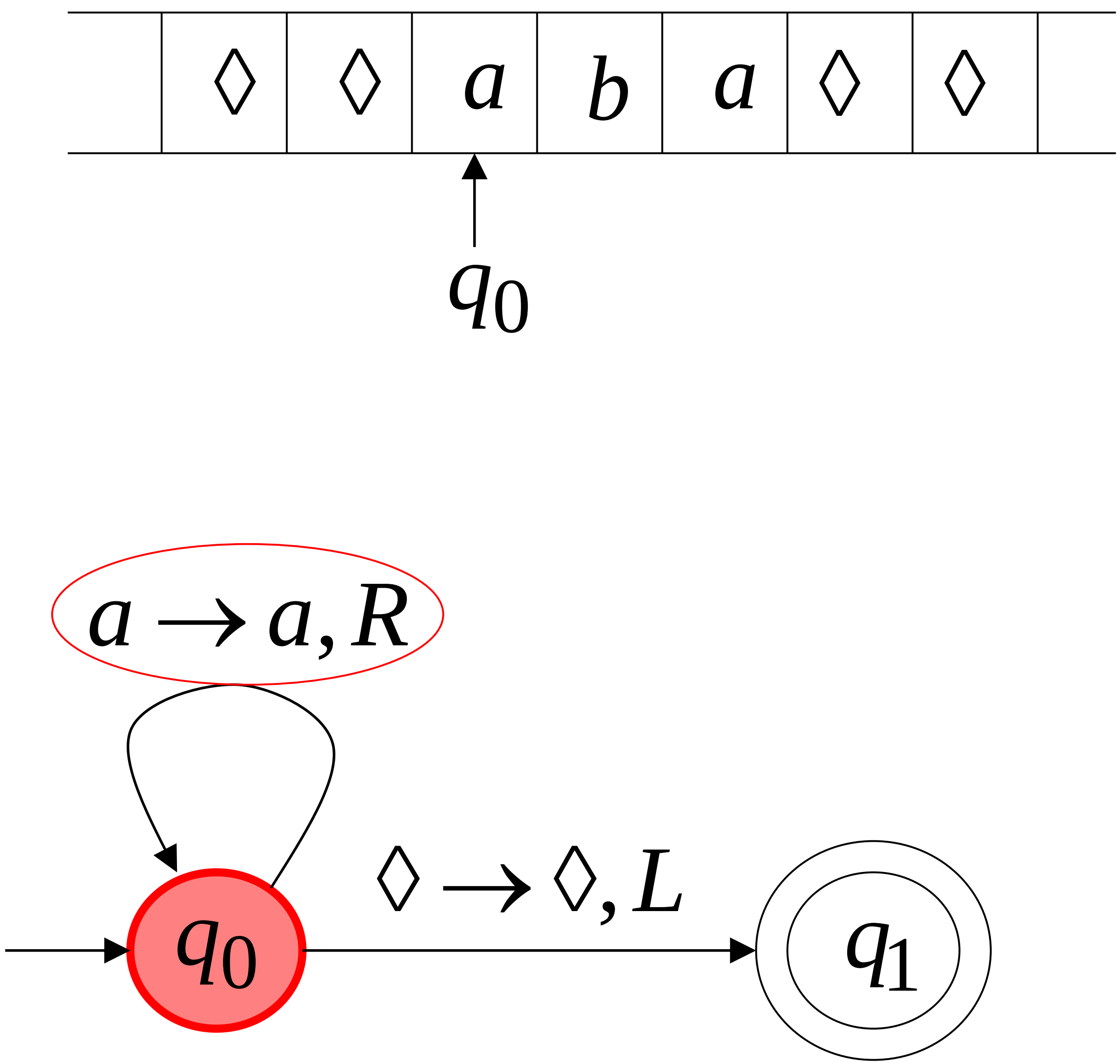
{a,b}*{aba}


b
a
a
a
b
a
b
b


b/b, R
a/a, R
a/a, R
a/a, R
Δ/Δ, R
a/a, R
b/b, R
Δ/Δ, R
q0
ha
b/b, R
b/b, R

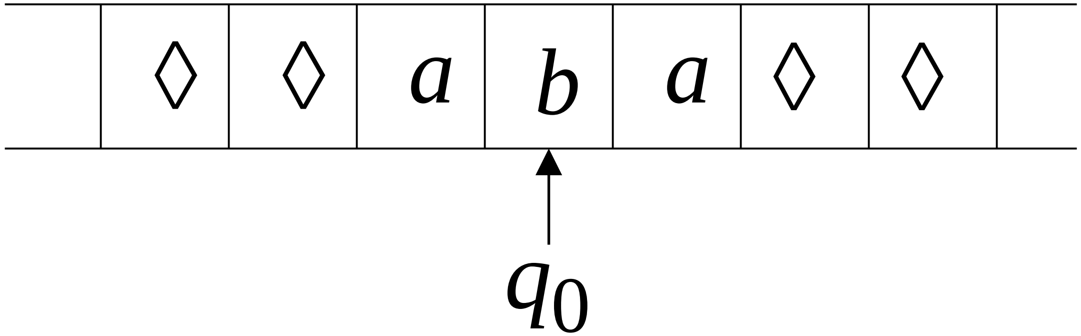
Rejection example

Time 0



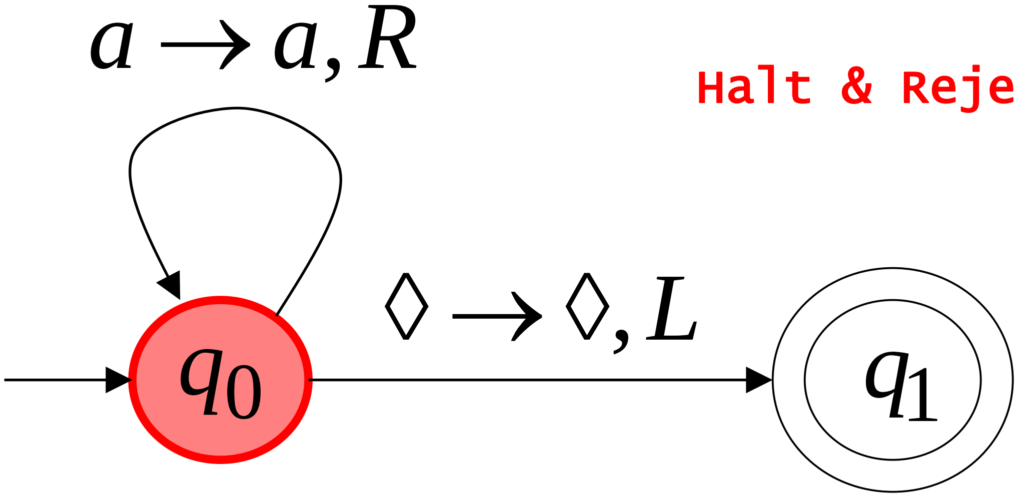
Rejection example

Time 1



No possible Transition

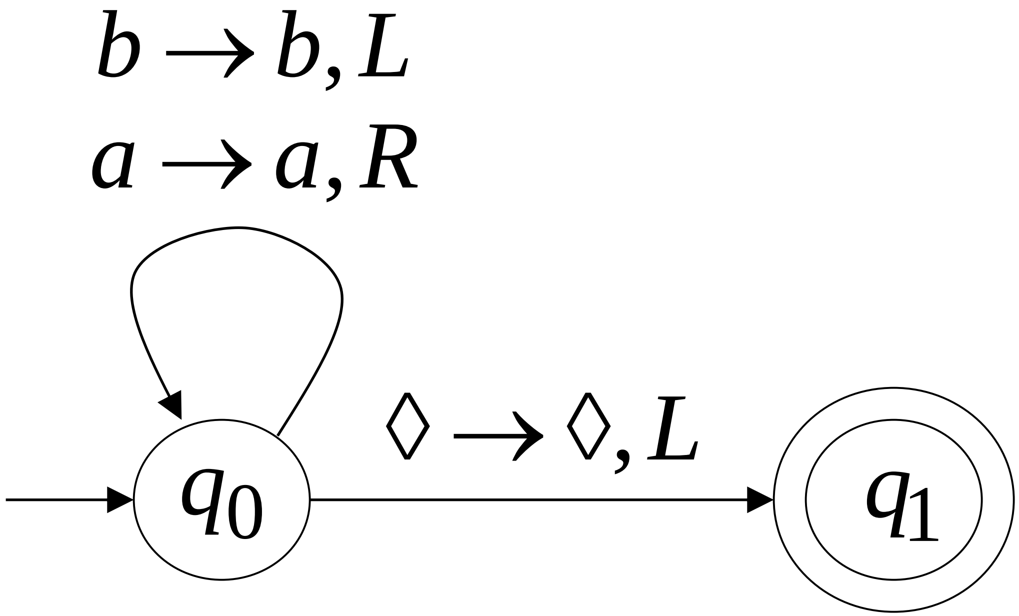
Halt & Reject



Infinite loop example

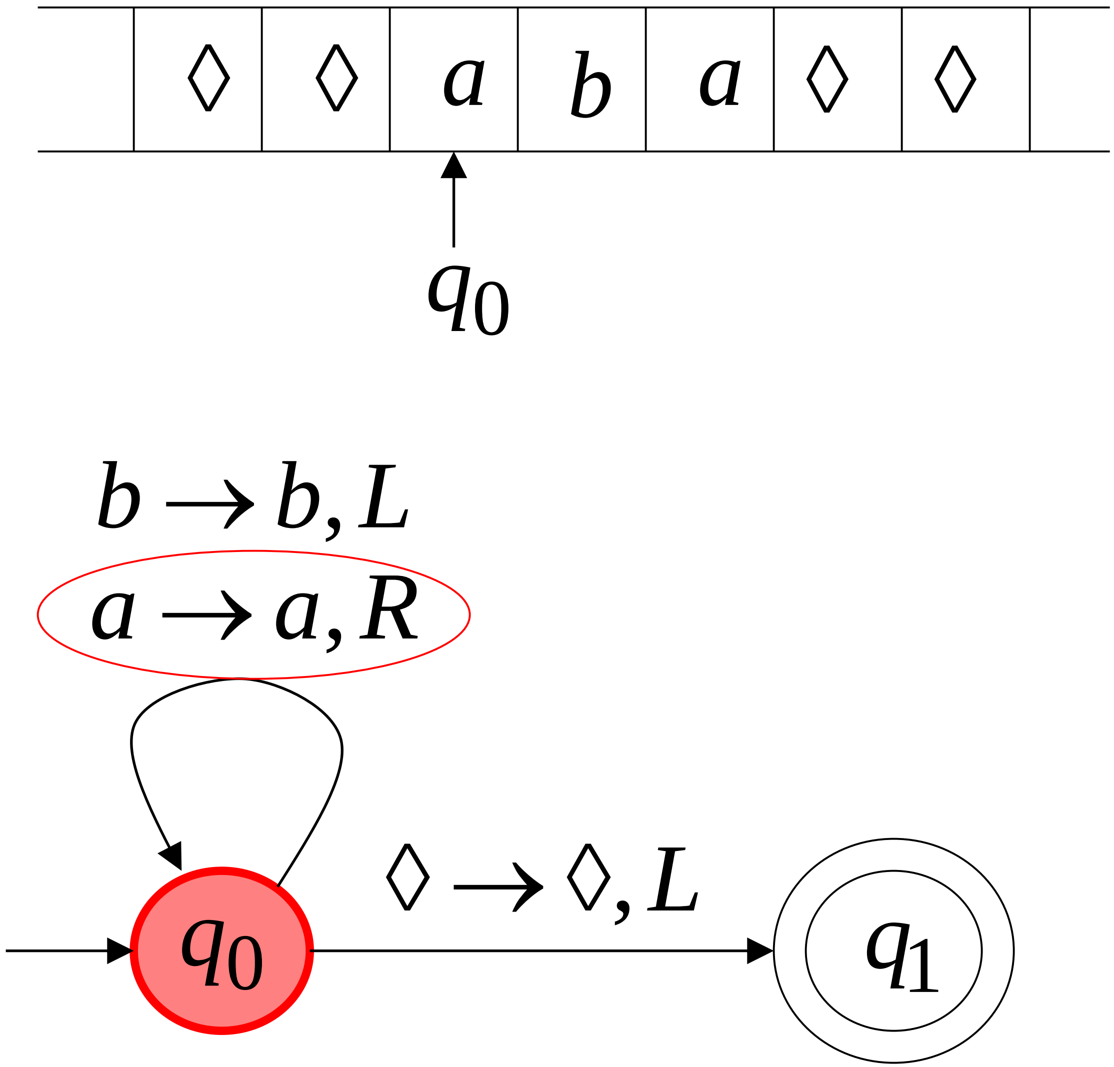
Another Turing machine for language a^*

and is this one correct???



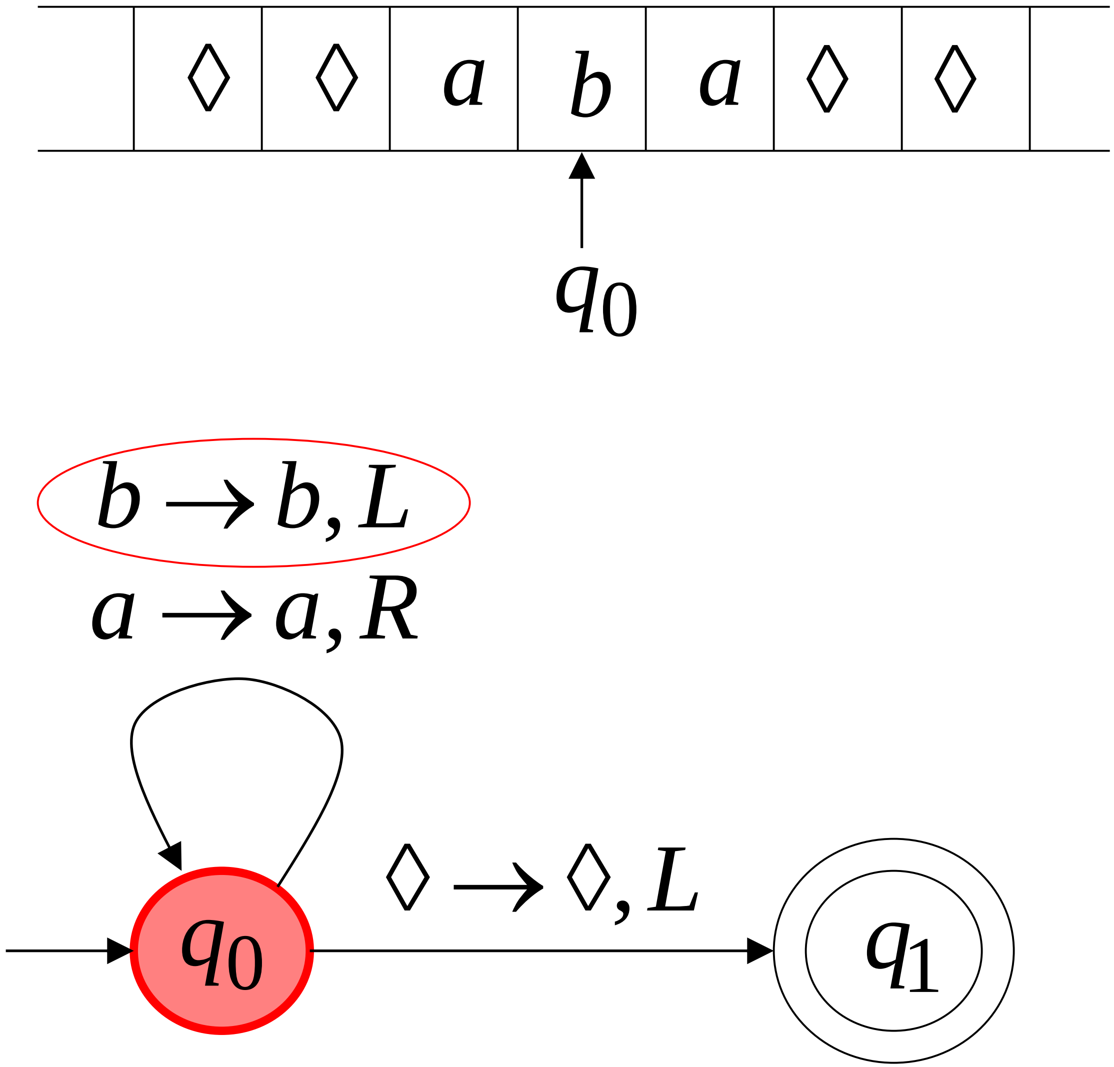
Infinite loop example

Time 0



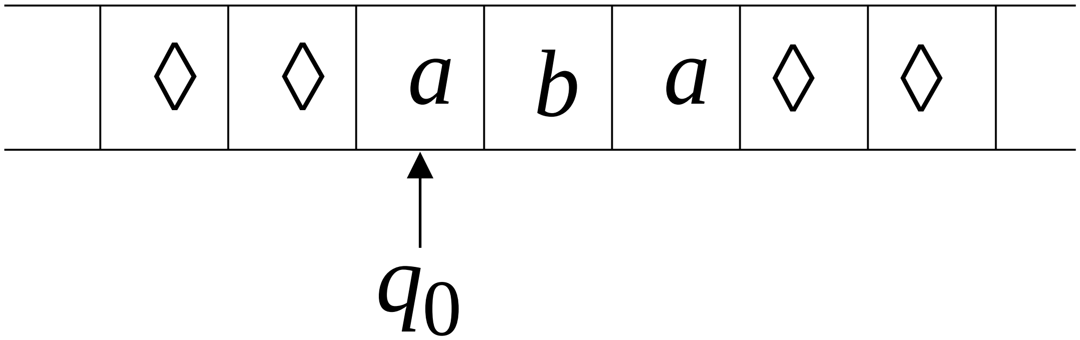
Infinite loop example

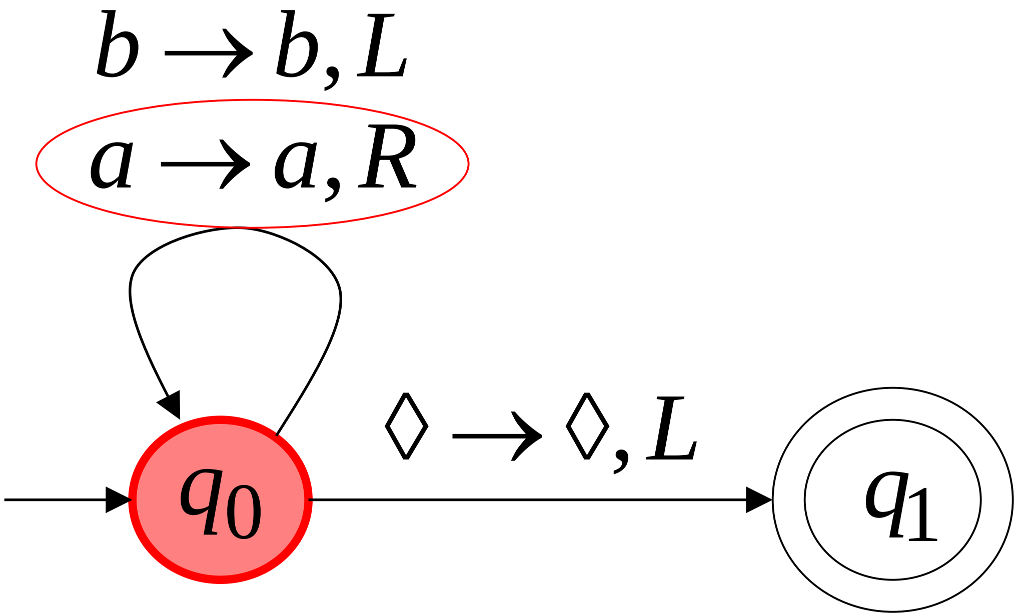
Time 1



Infinite loop example

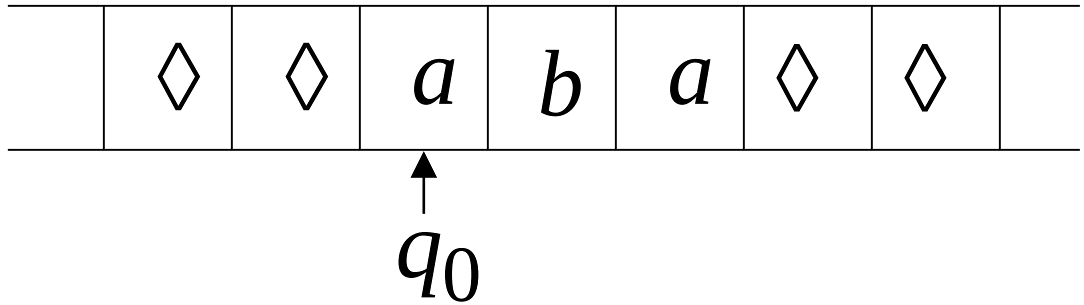
Time 2



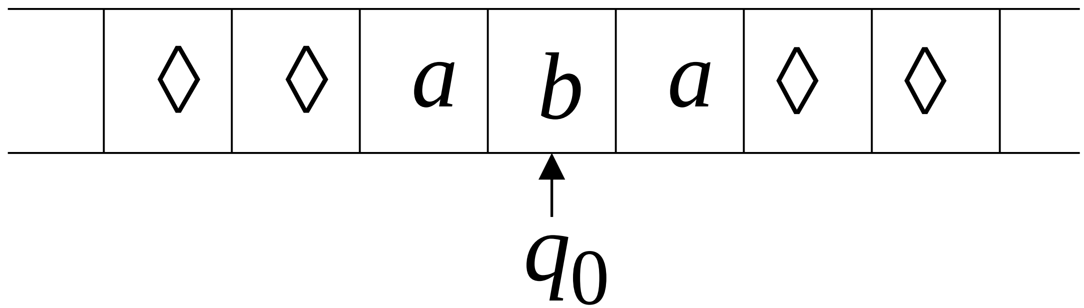


Infinite loop example

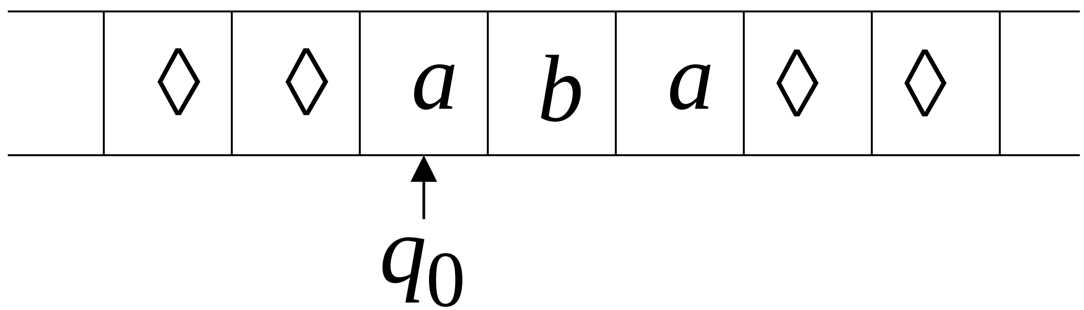
Time 2



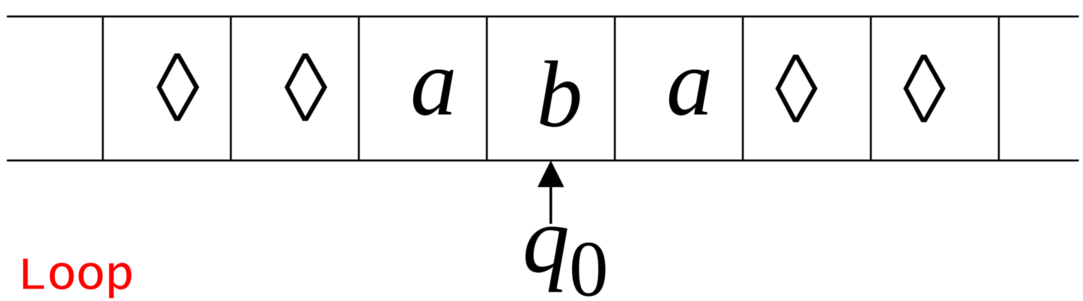
Time 3



Time 4



Time 5



... Infinite Loop

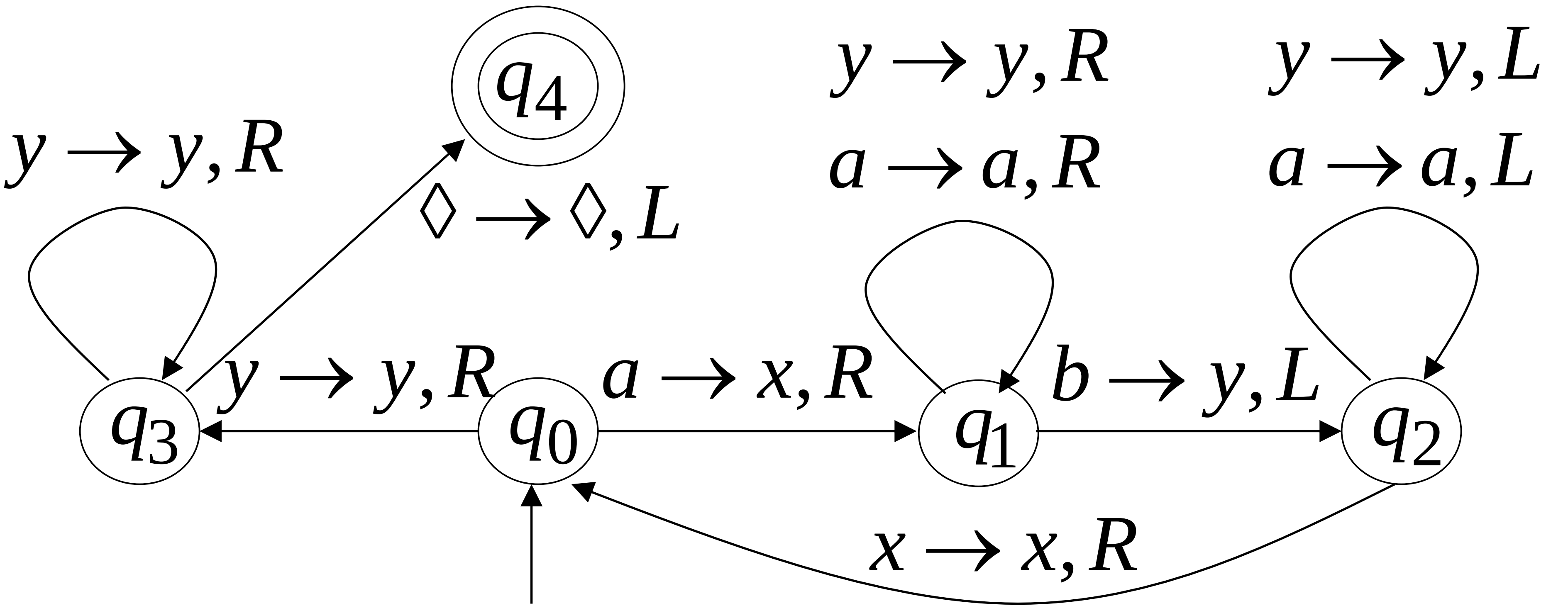
Infinite loop example

Because of the infinite loop:

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

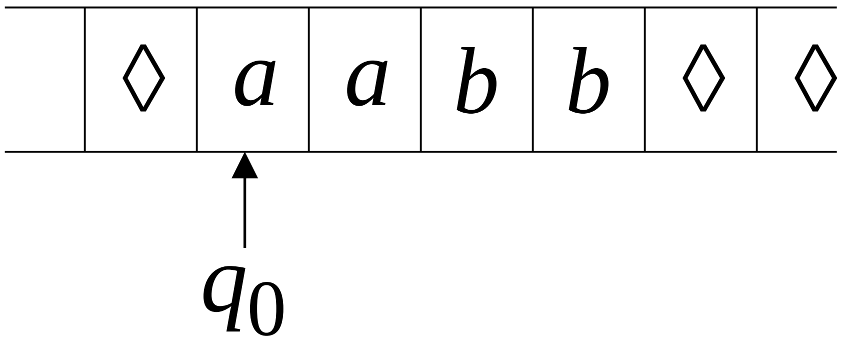
Another Turing machine example

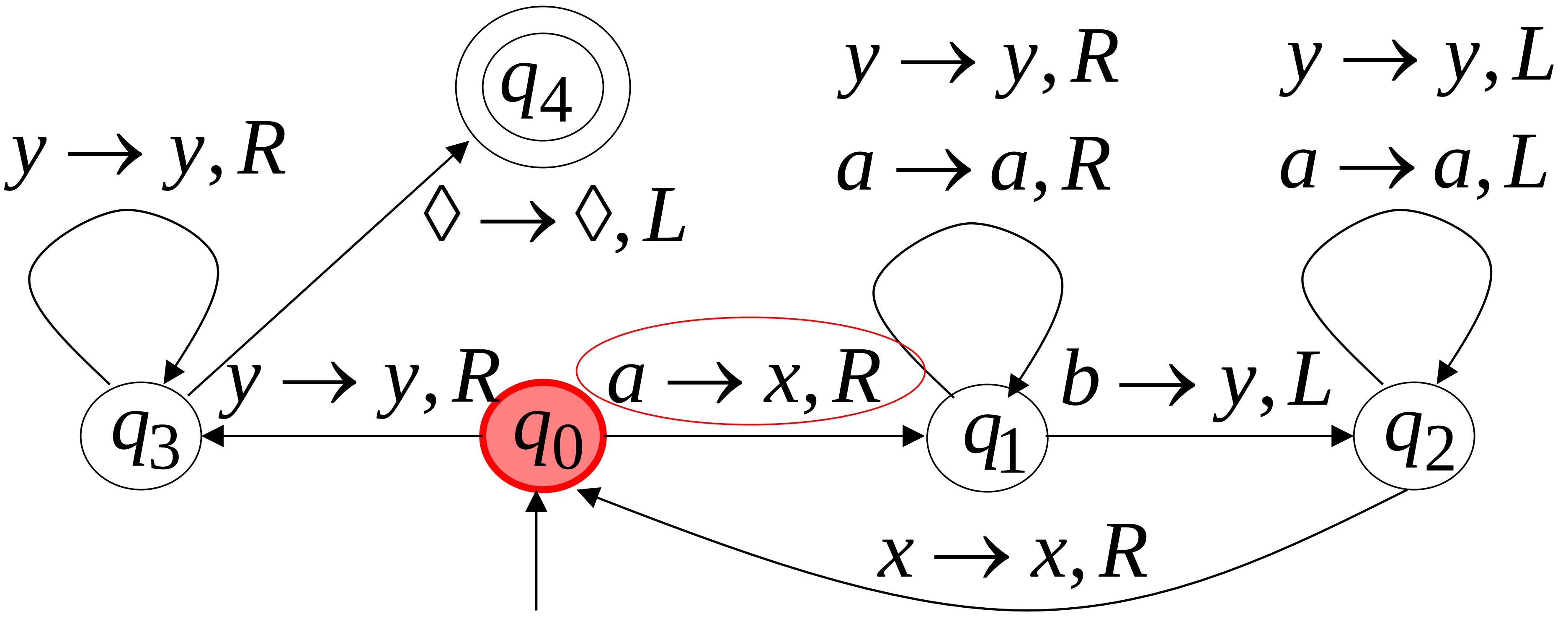
Turing machine for the language $\{a^n b^n\}$



Another Turing machine example

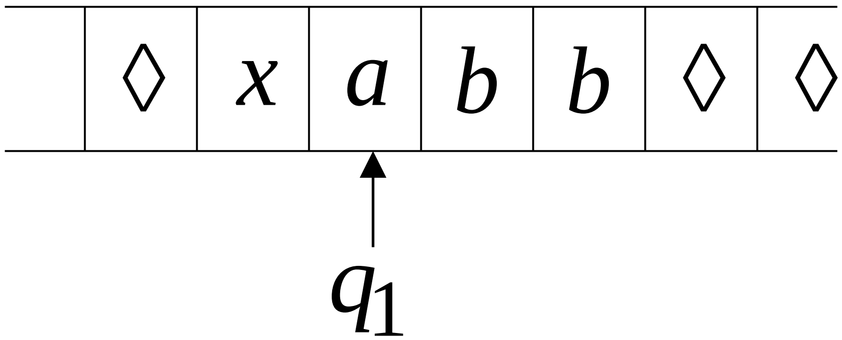
Time 0

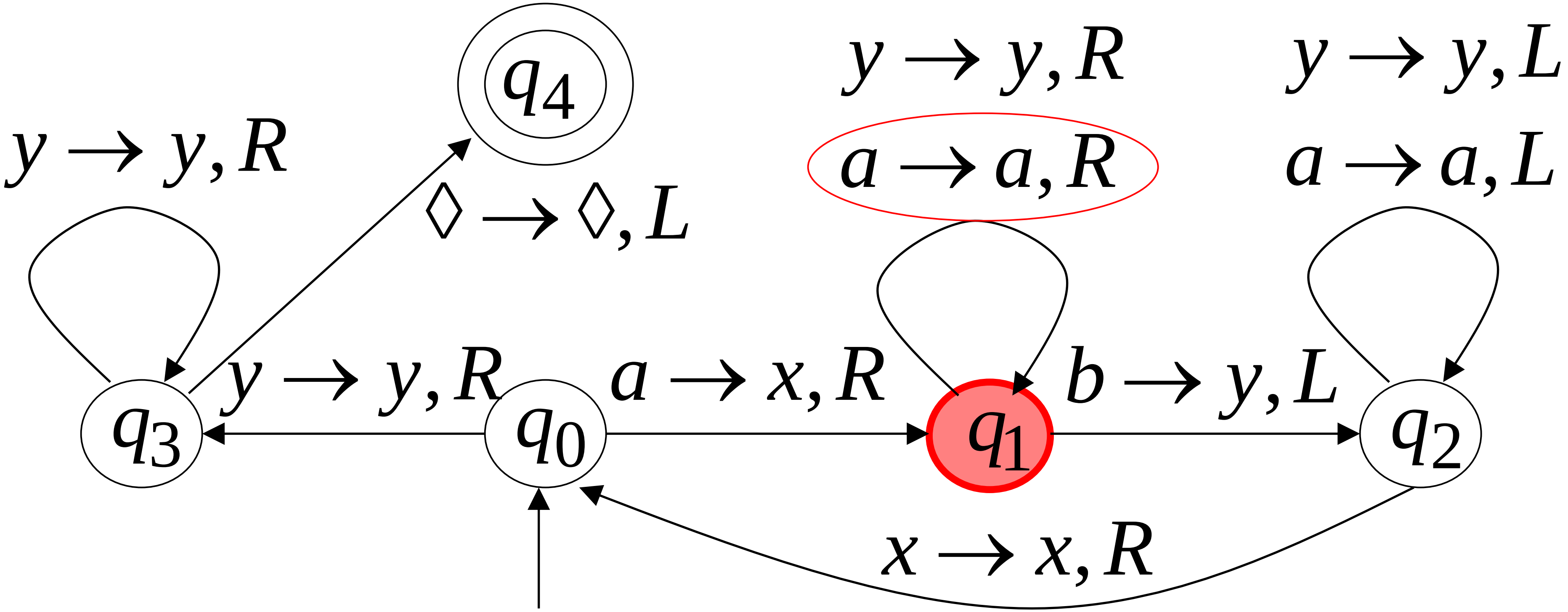




Another Turing machine example

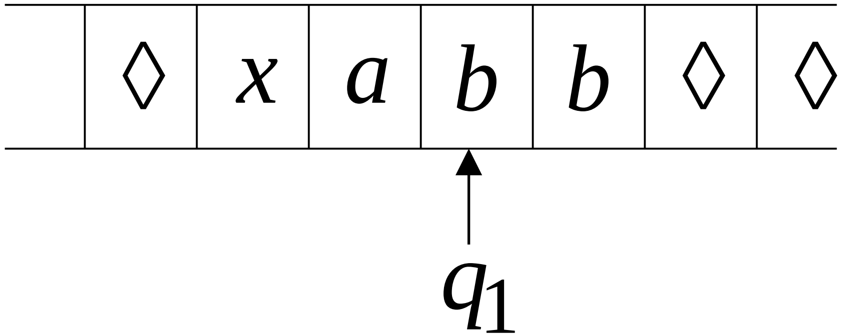
Time 1

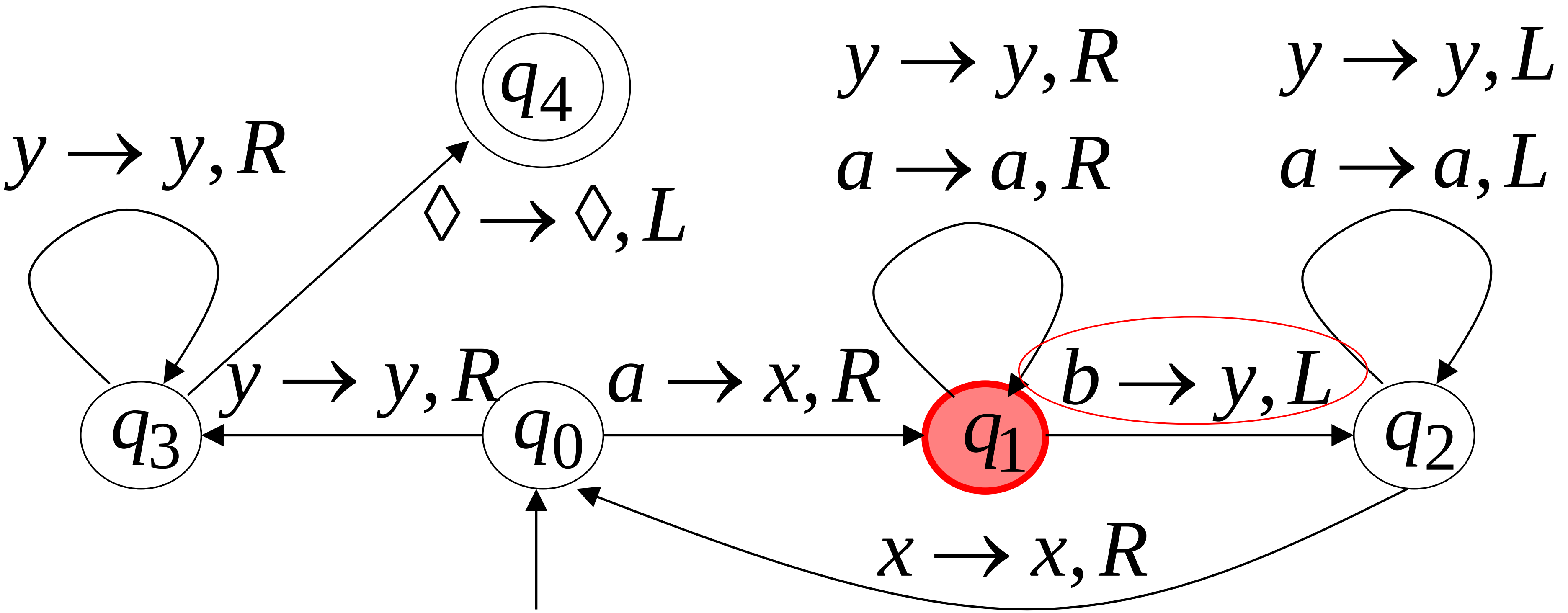




Another Turing machine example

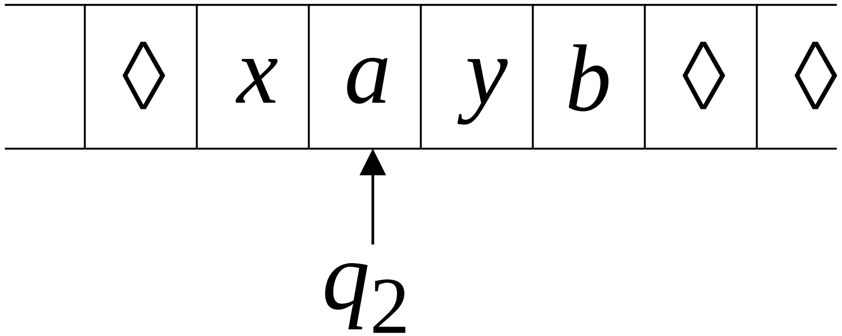
Time 2

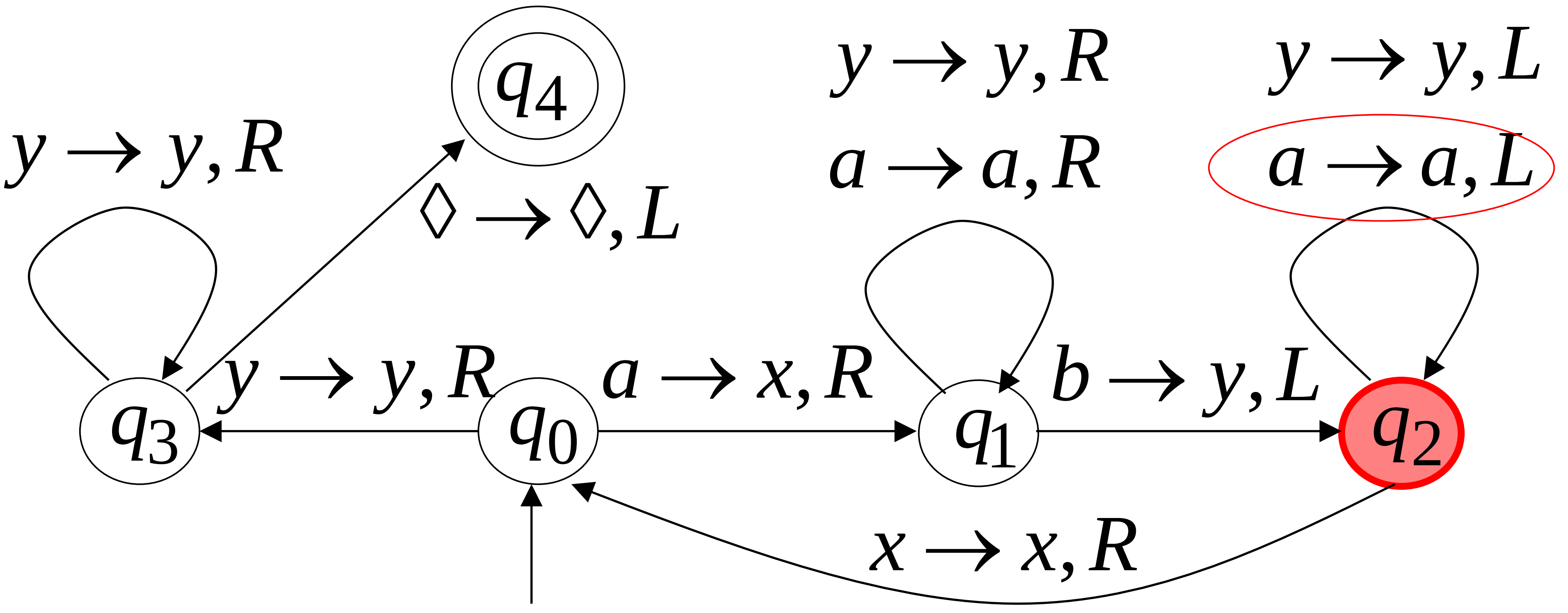




Another Turing machine example

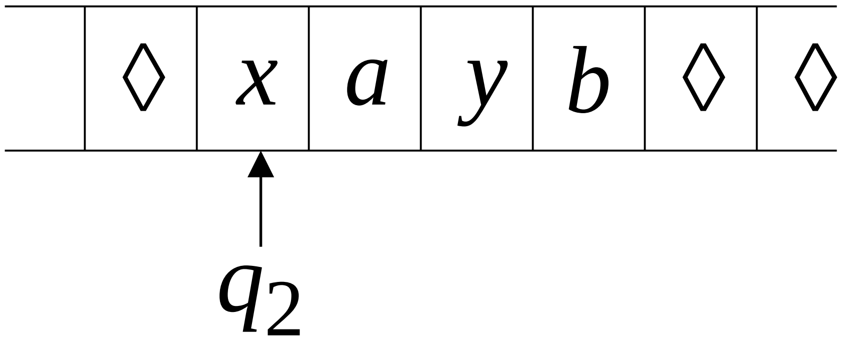
Time 3

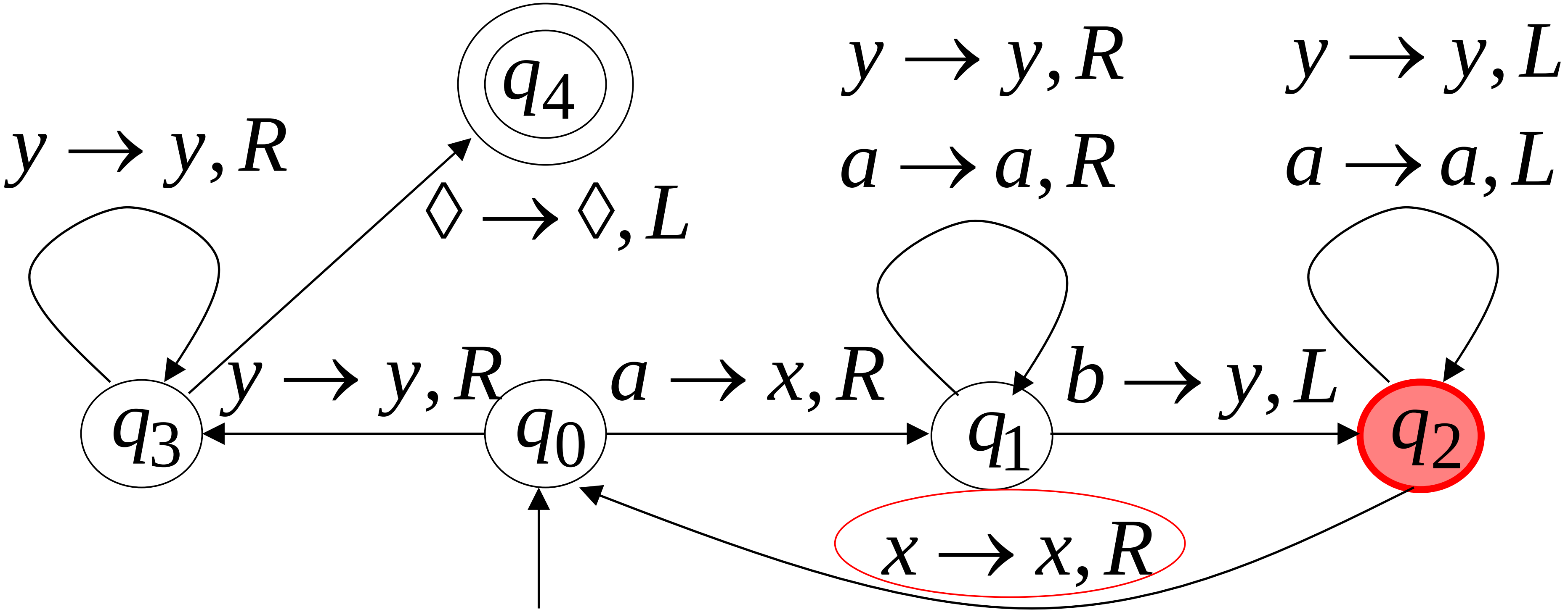




Another Turing machine example

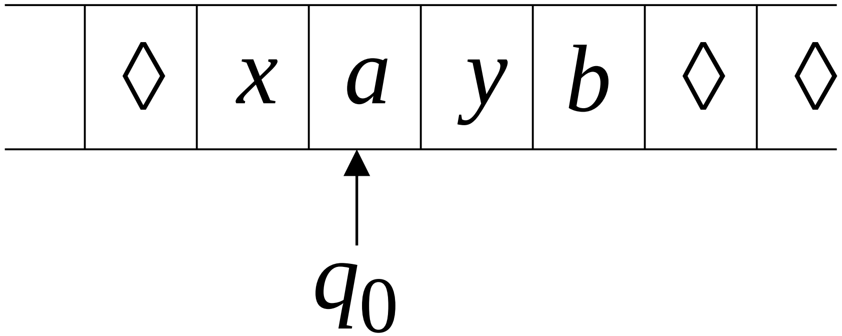
Time 4

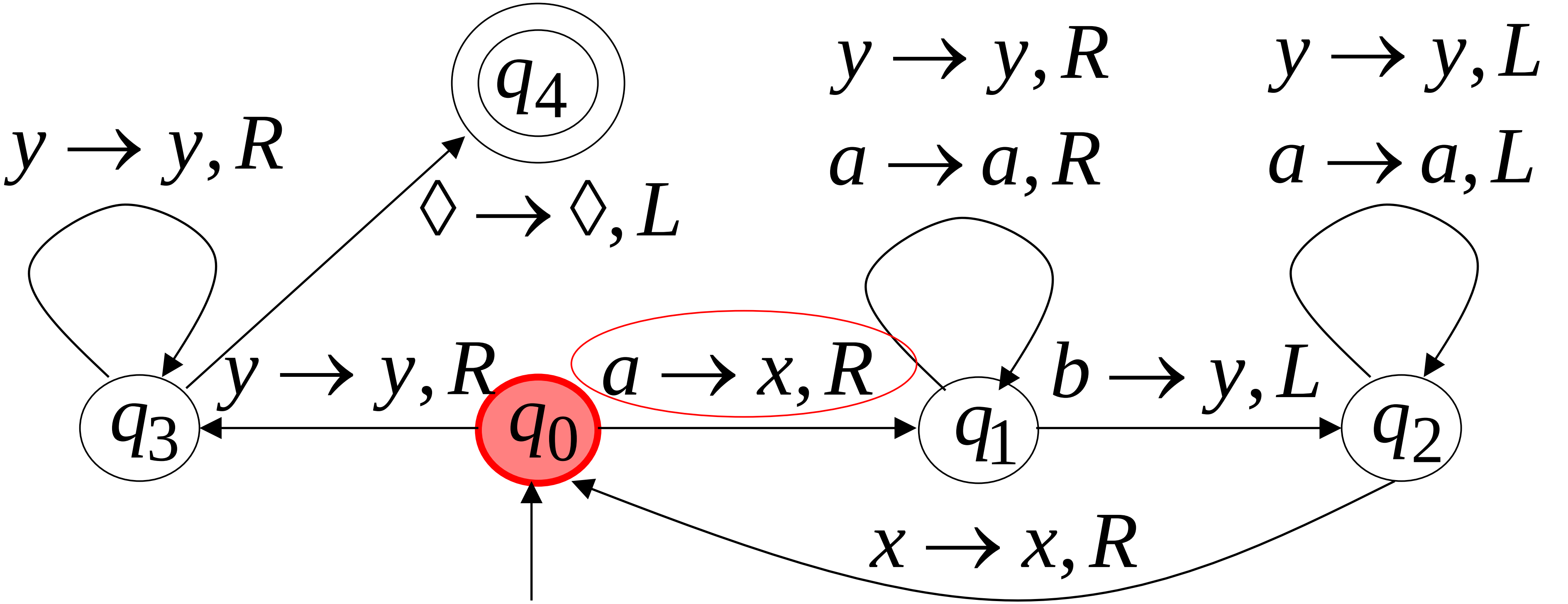




Another Turing machine example

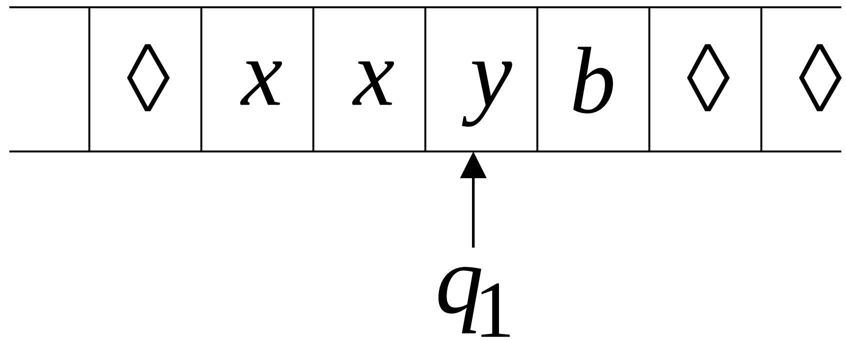
Time 5

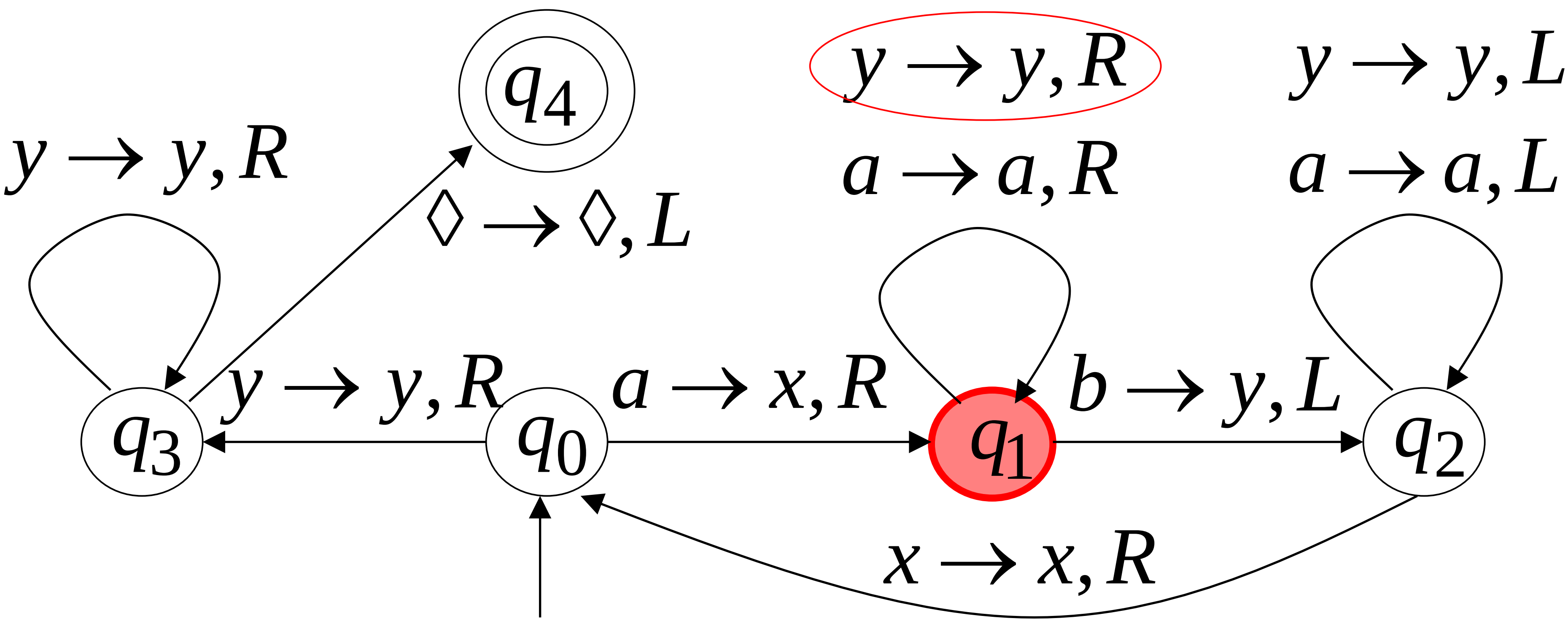




Another Turing machine example

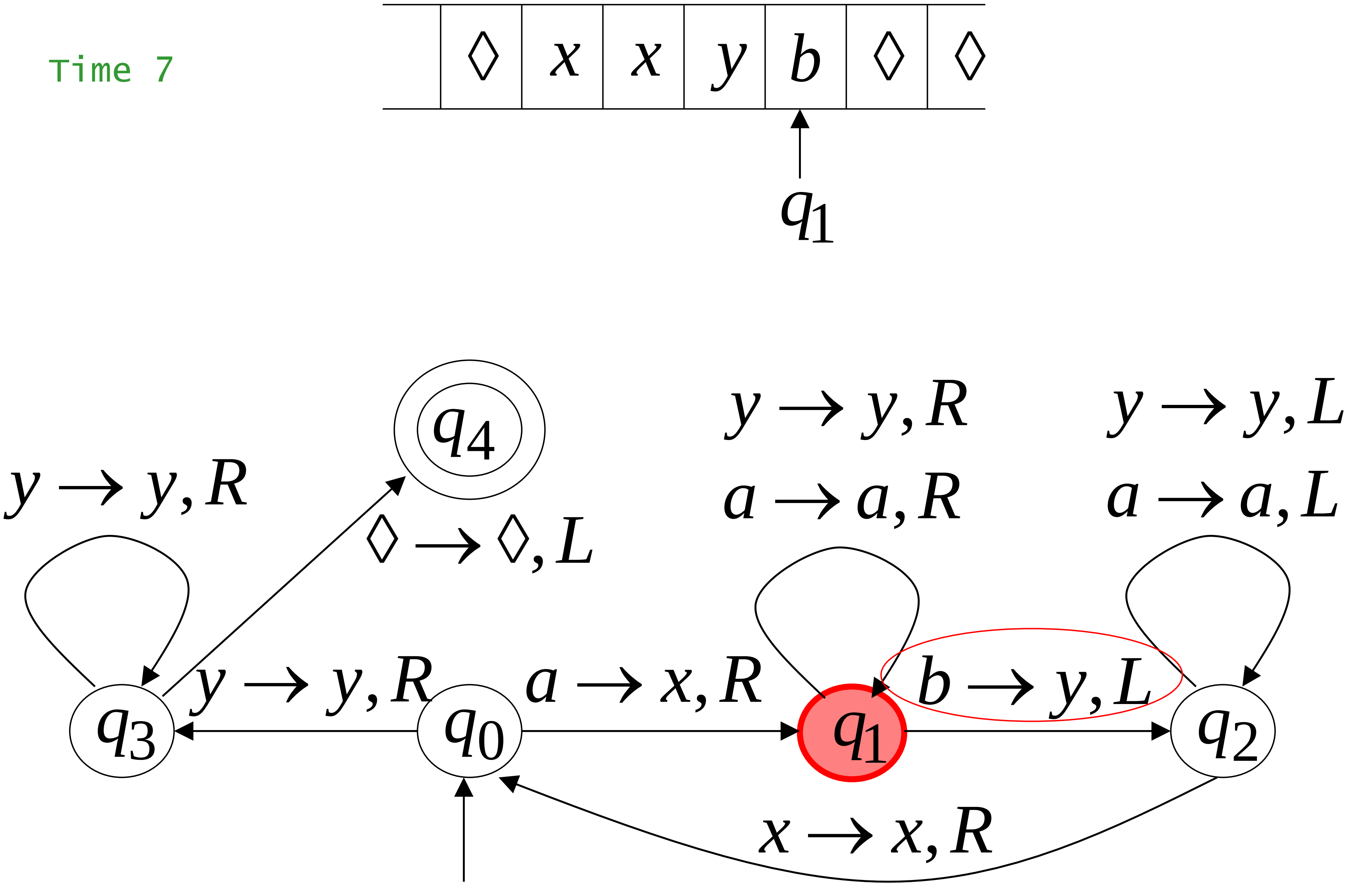
Time 6





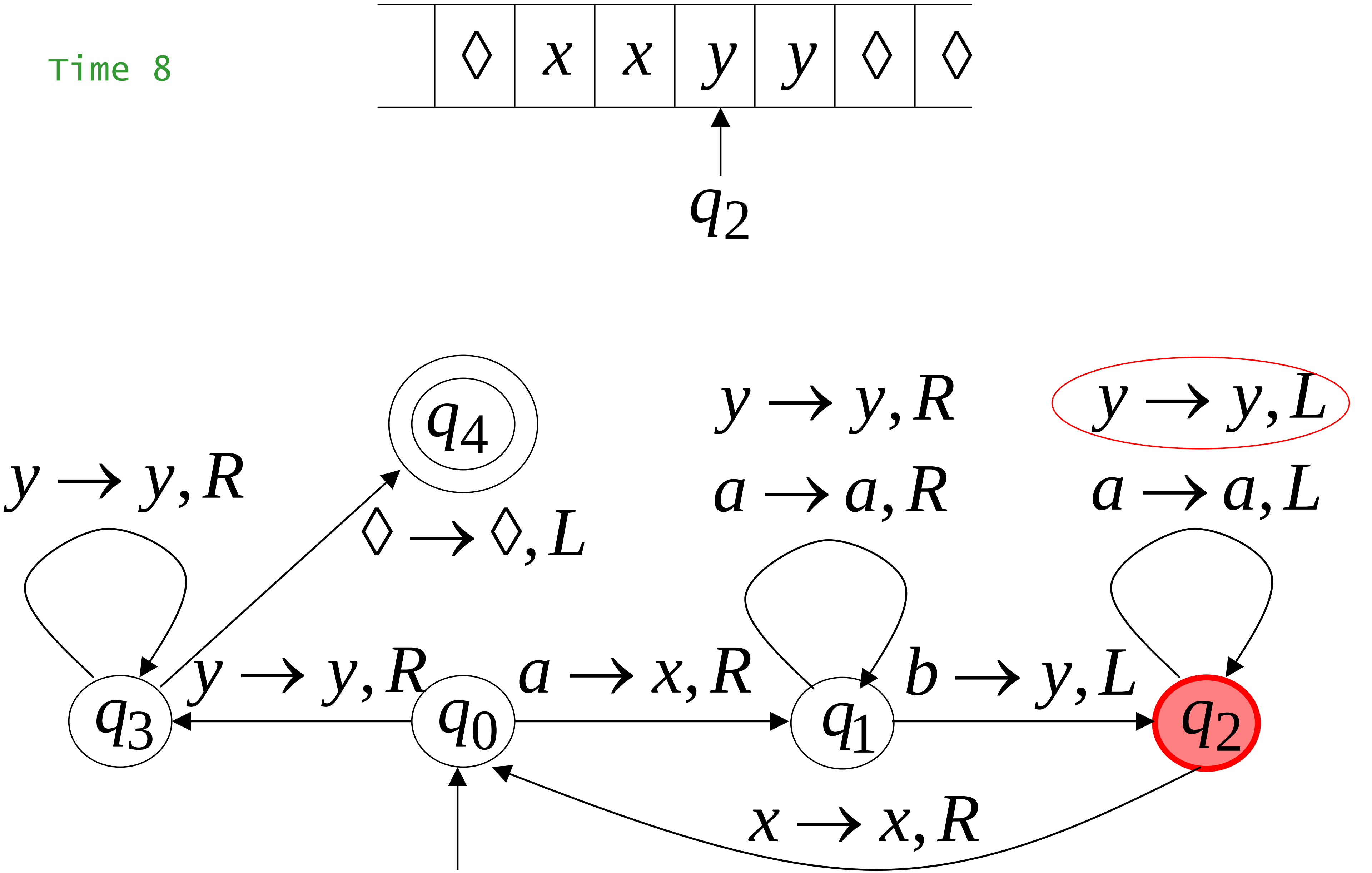
Another Turing machine example

Time 7



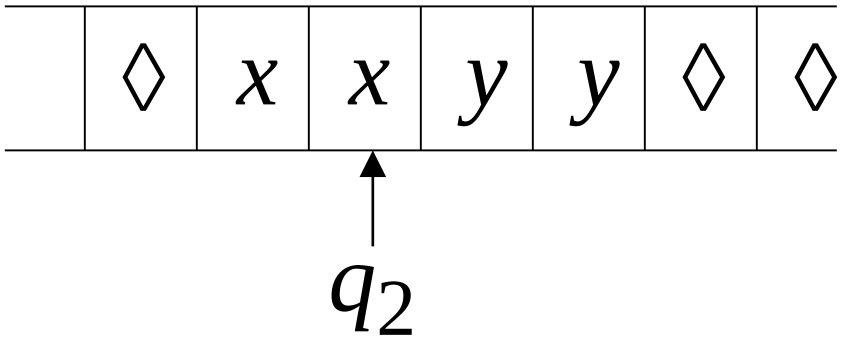
Another Turing machine example

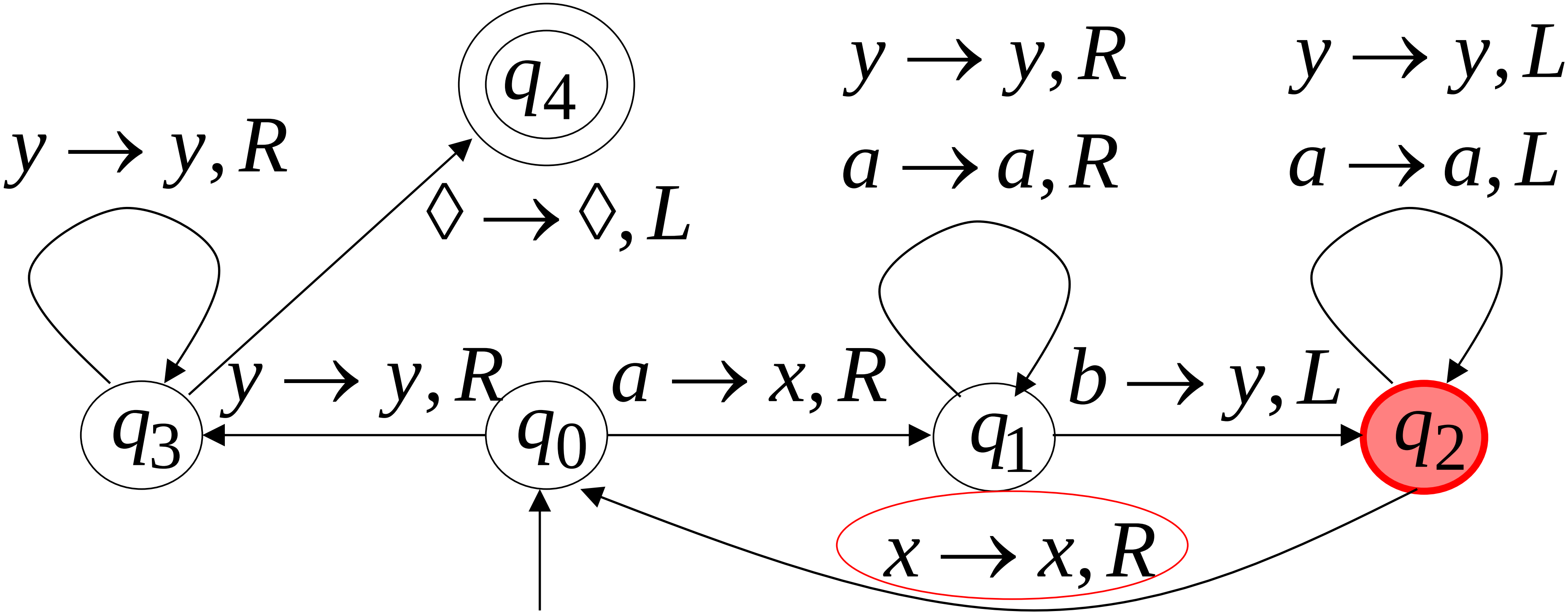
Time 8



Another Turing machine example

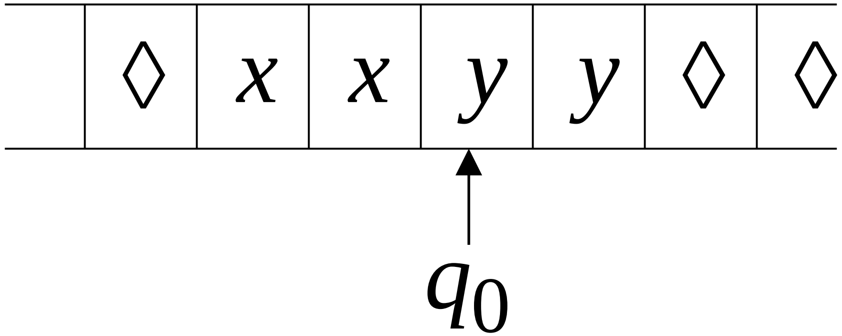
Time 9

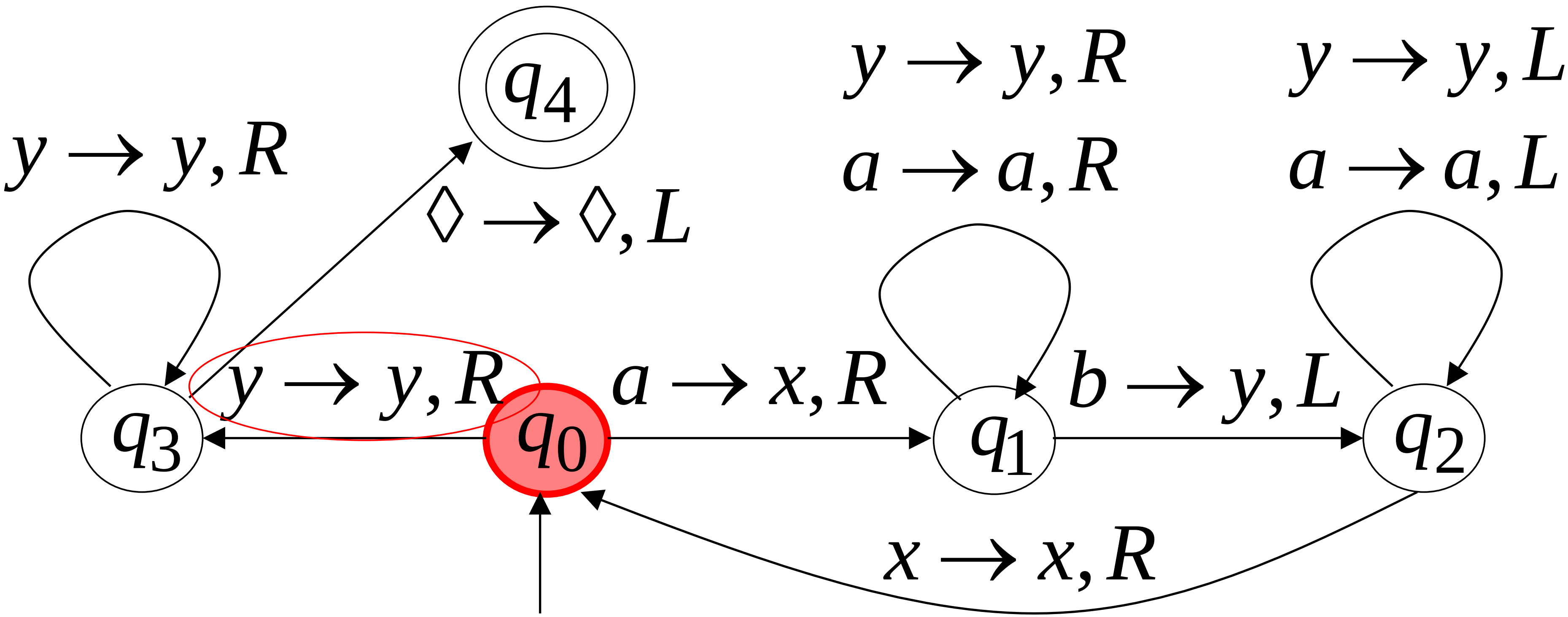




Another Turing machine example

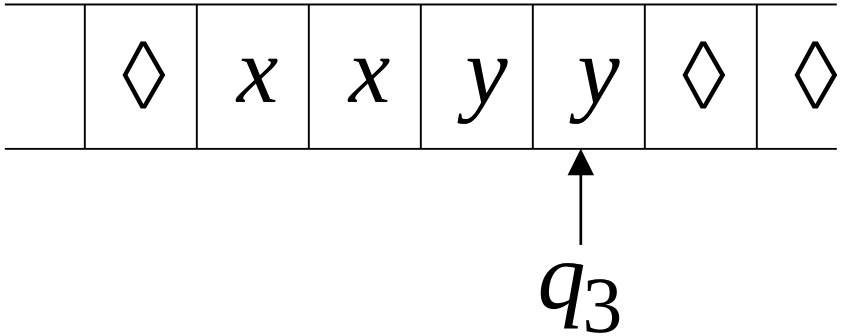
Time 10

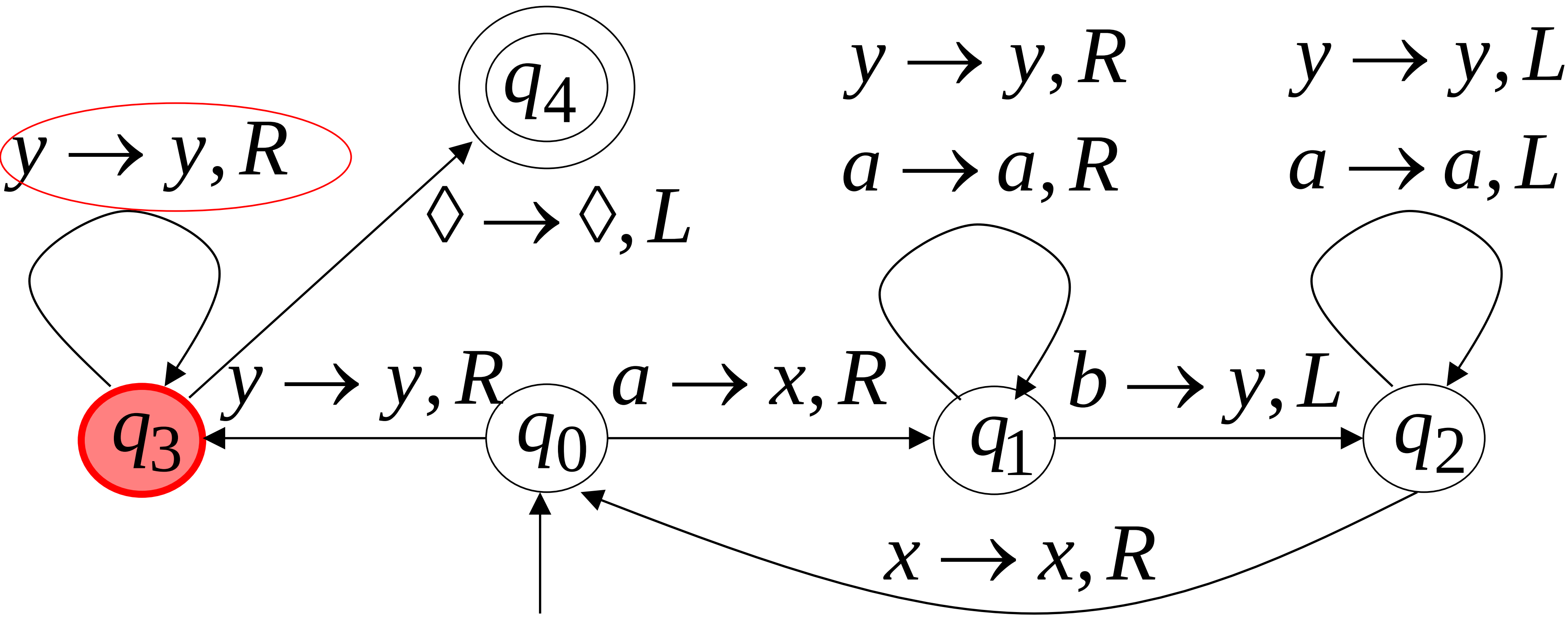




Another Turing machine example

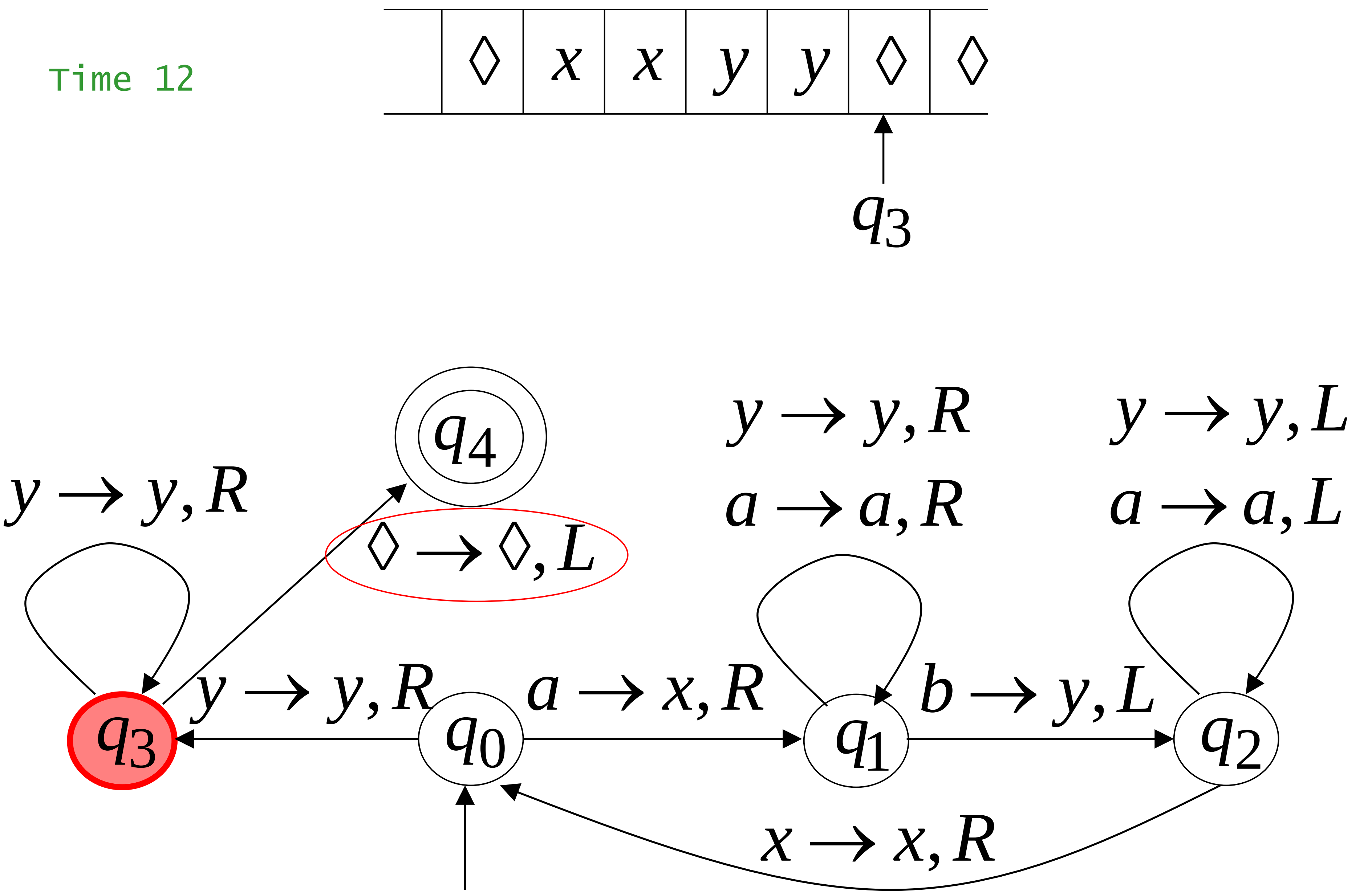
Time 11





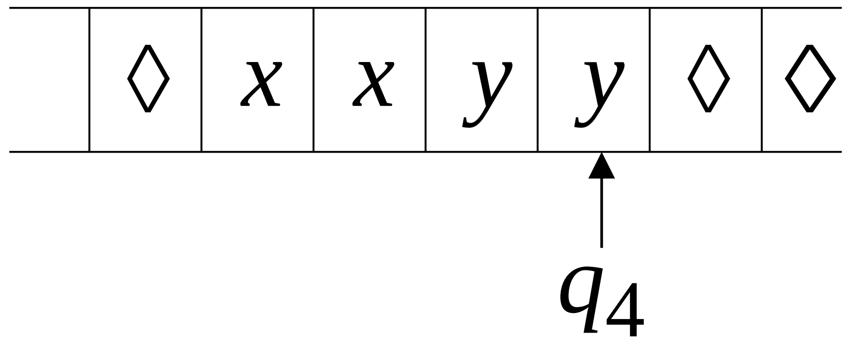
Another Turing machine example

Time 12

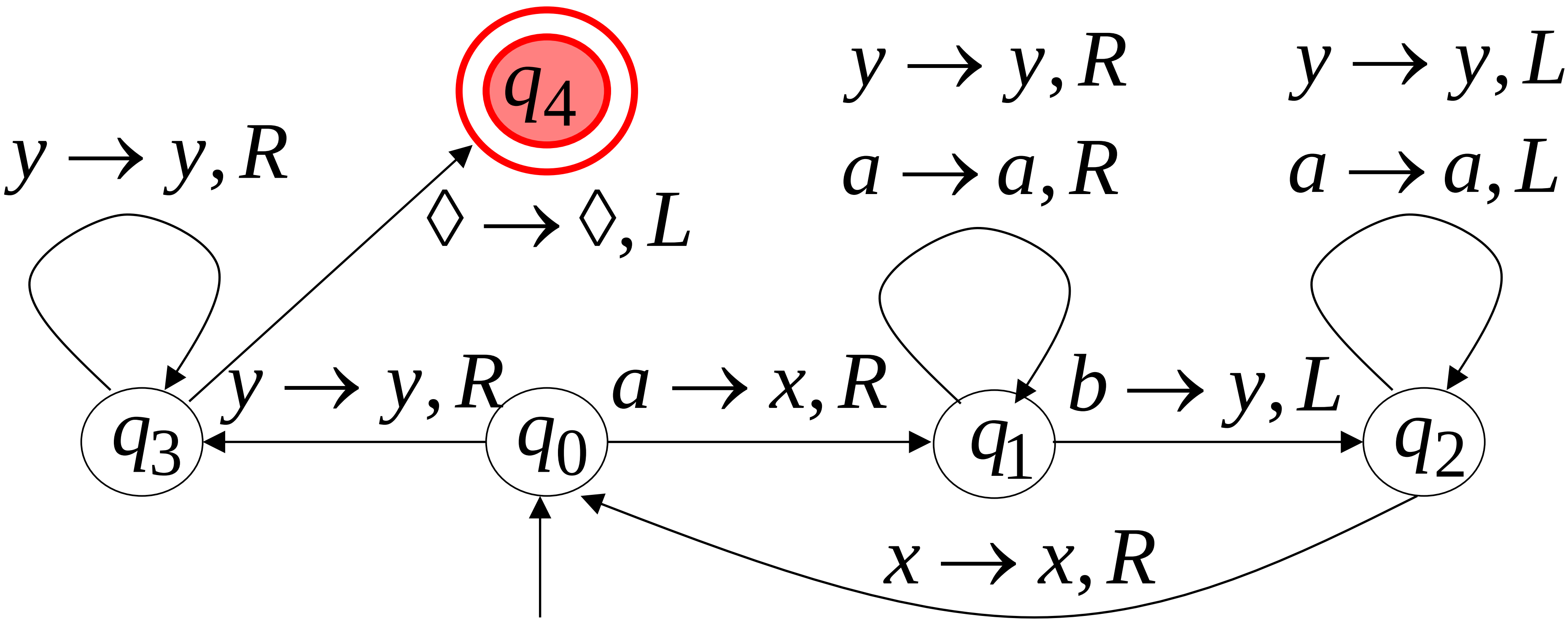


Another Turing machine example

Time 13



Halt & Accept



Observation

If we modify the machine for the language

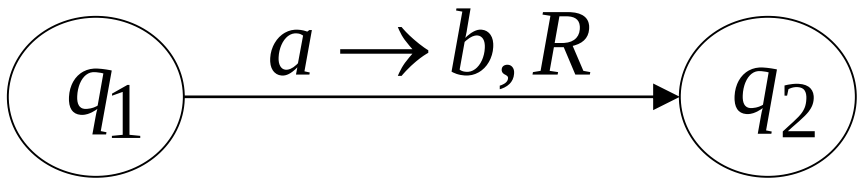
$$\{a^n b^n\}$$

we can easily construct a machine for the language

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

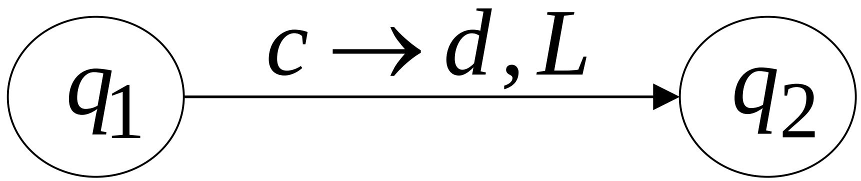
Formal definitions

Transition function



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

Transition function



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

Turing machine

Input alphabet Tape alphabet

States

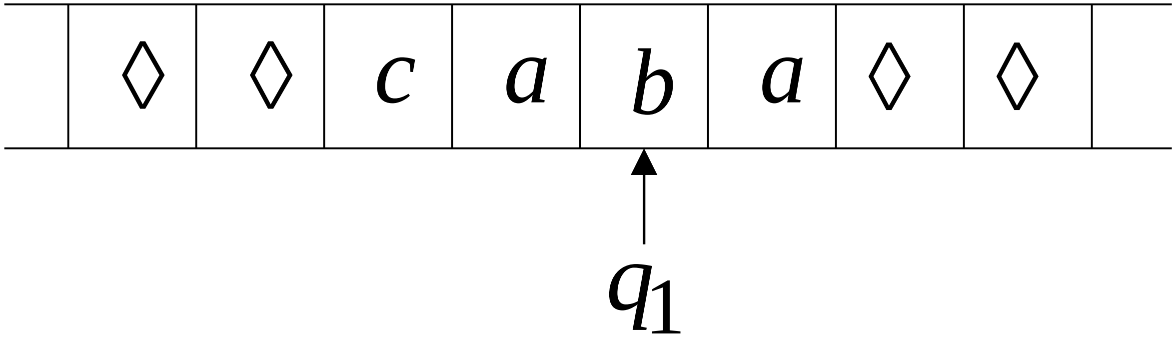
$$M = (Q, \Sigma, \Gamma, \delta, q_0, \Diamond, F)$$

Final states

Transition function

Initial state blank

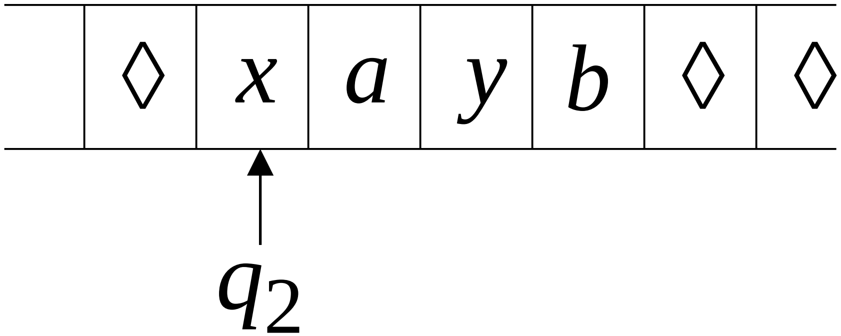
Configuration



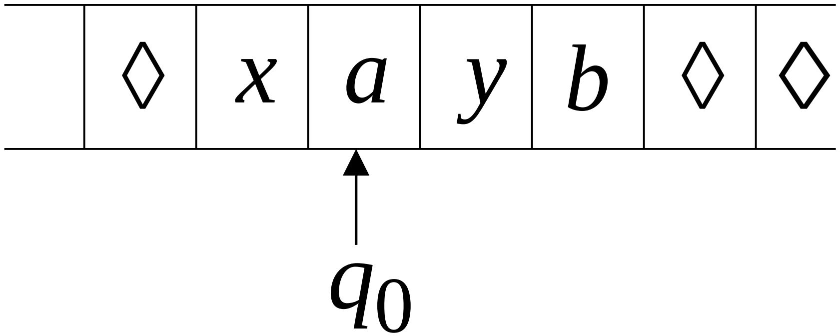
Instantaneous description: $ca\ q_1\ ba$

Configuration

Time 4

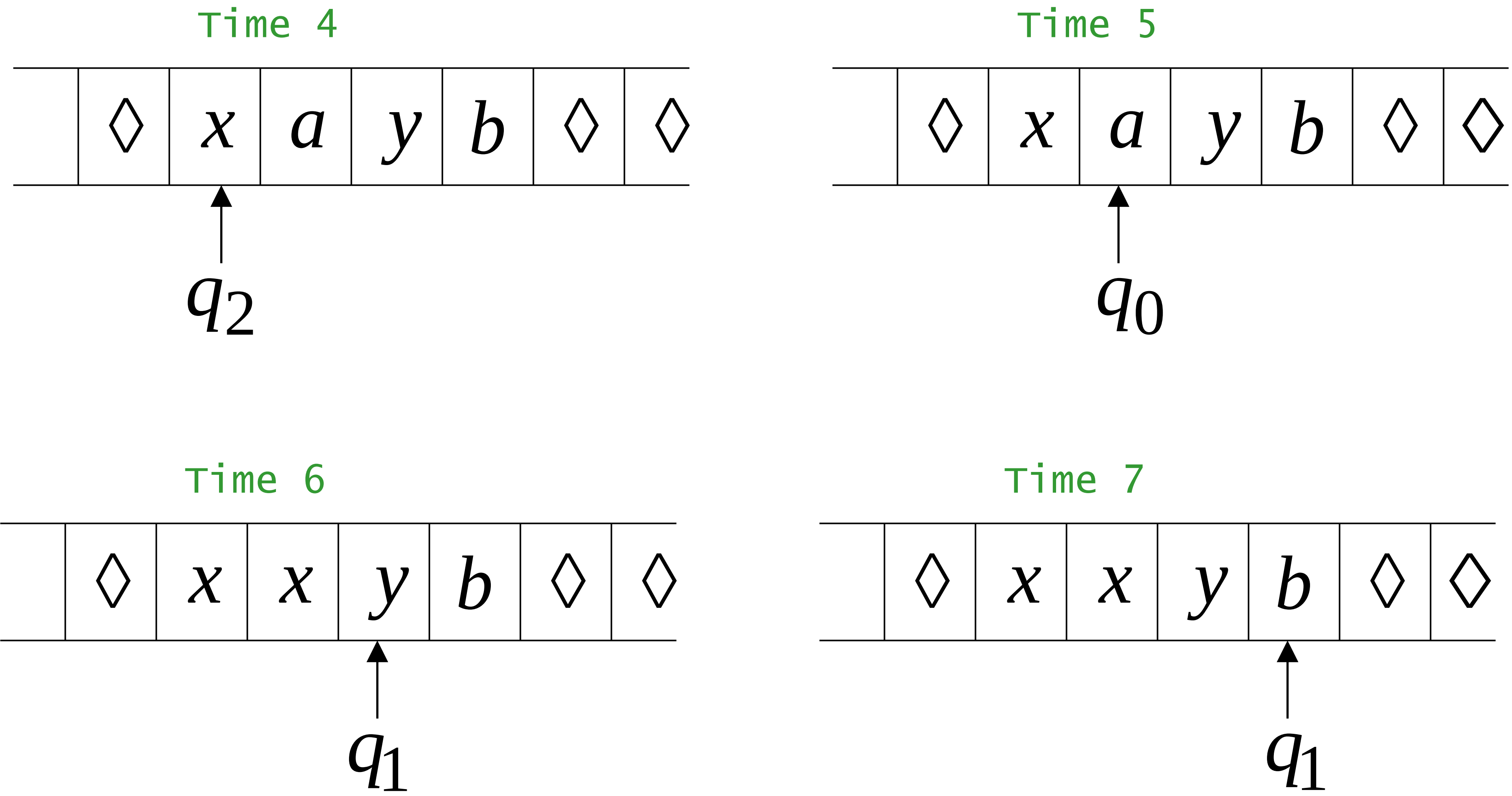


Time 5



A Move: $q_2 \; xayb \succ x \; q_0 \; ayb$

Configuration



$$q_2\ xayb \succ x\ q_0\ ayb \succ xx\ q_1\ yb \succ xxy\ q_1\ b$$

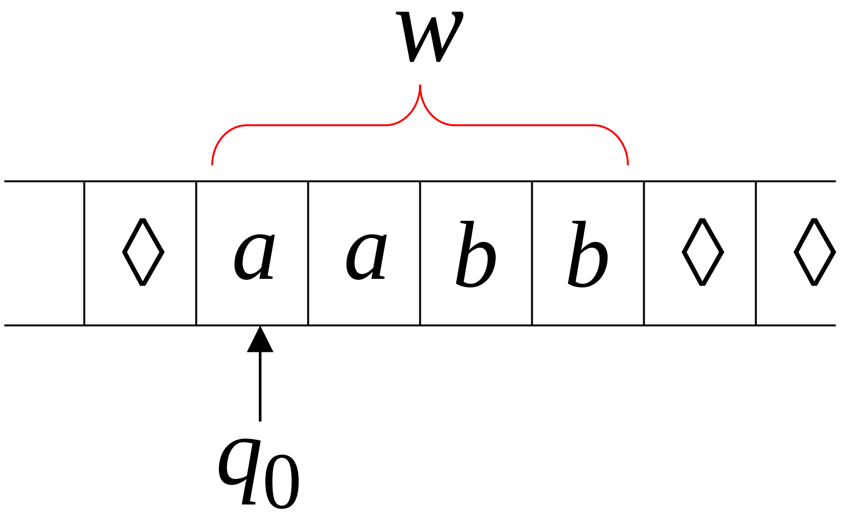
Configuration

$$q_2\, xayb \succ x\, q_0\, ayb \succ xx\, q_1\, yb \succ xxy\, q_1\, b$$

Equivalent notation: $q_2\, xayb \overset{*}{\succ} xxy\, q_1\, b$

Initial configuration $q_0 \, w$

Input string



The accepted language

For any Turing Machine M

$$L(M) = \{w : \quad q_0\, w \overset{*}{\succ} x_1\, q_f\, x_2\}$$

Initial state

Final state

Standard Turing machine

The machine we described is the standard

- Deterministic
- Infinite tape in both directions
- Tape is the input/output file

Computing functions

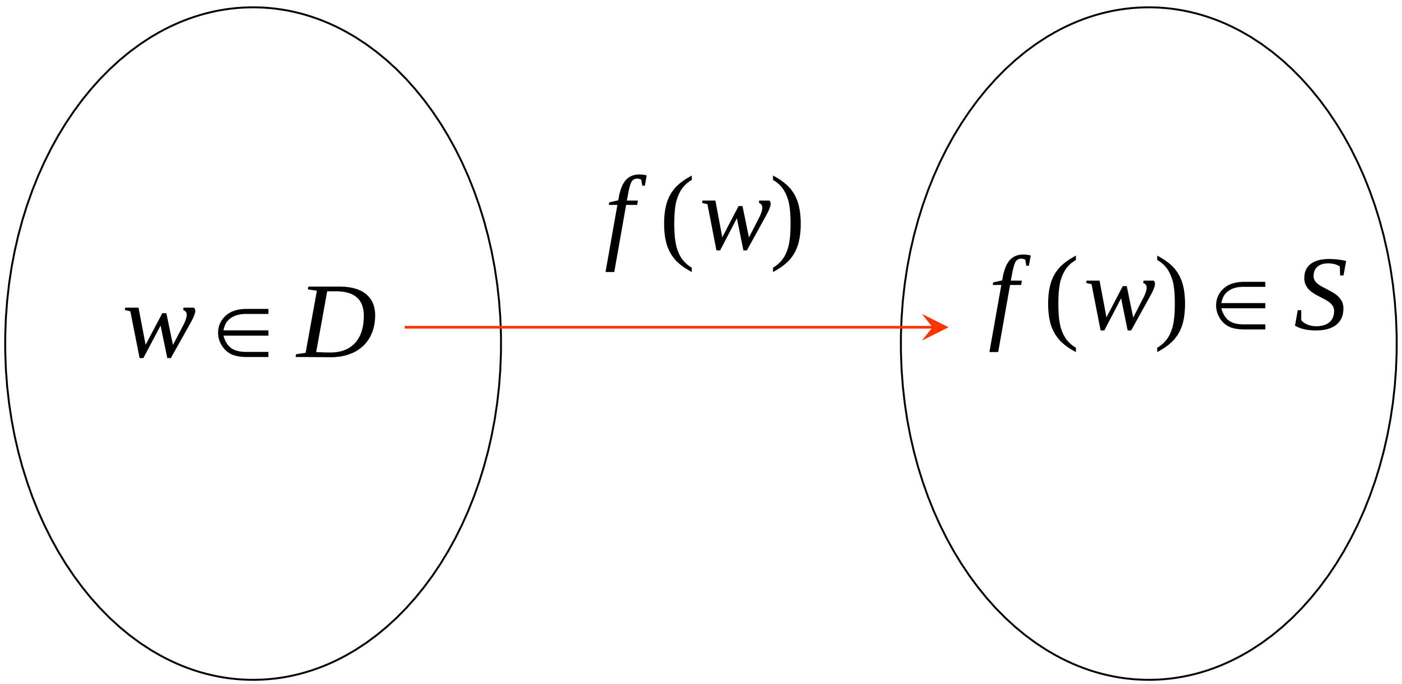
....

Functions

A function $f(w)$ has:

Domain: D

Result Region: S



Functions

A function may have many parameters

Example:

Addition function

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

Integer domain

Decimal: 5

Binary: 101

Unary: 11111

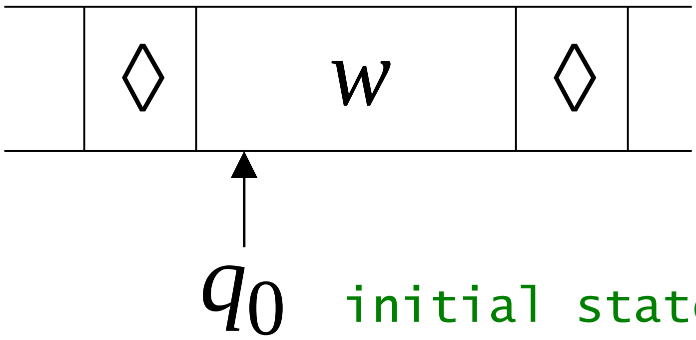
We prefer **unary** representation:

easier to manipulate with Turing machines

Functions definition

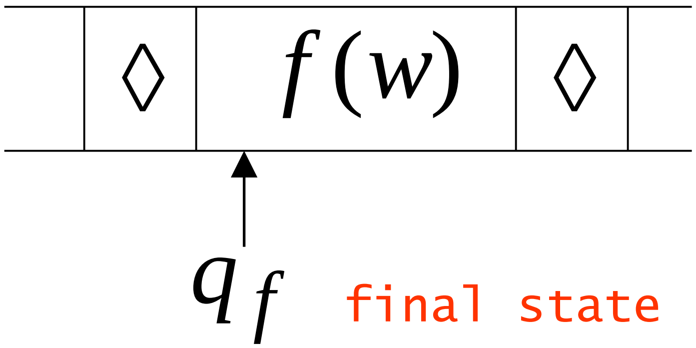
A function f is computable if
there is a Turing Machine M such that:

Initial configuration



initial state

Final configuration



final state

For all $w \in D$ Domain

Functions definition

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

Final
Configuration

For all $w \in D$ Domain

Example

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

x, y are integers

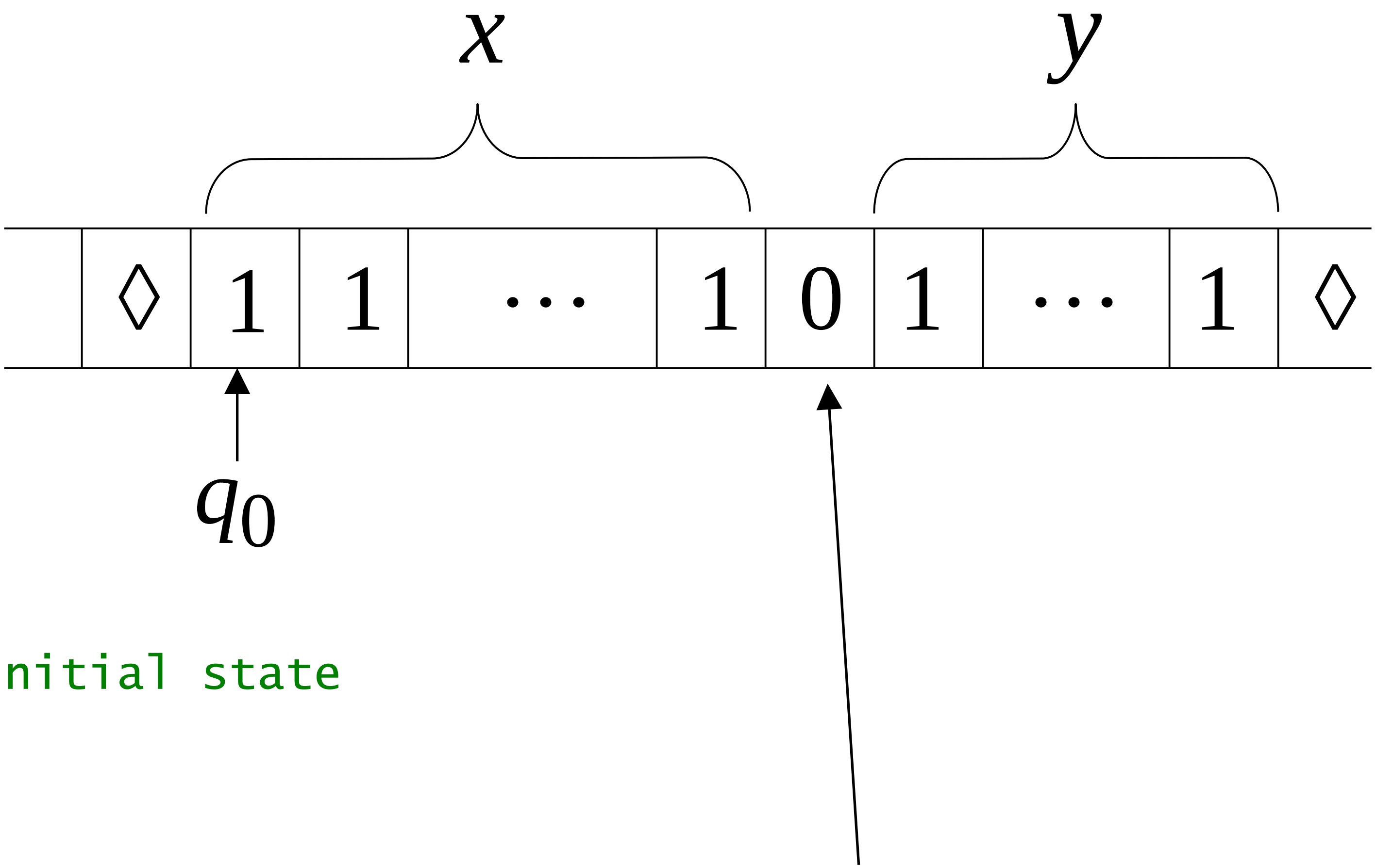
Turing Machine:

Input string: $x0y$ unary

Output string: $xy0$ unary

Example

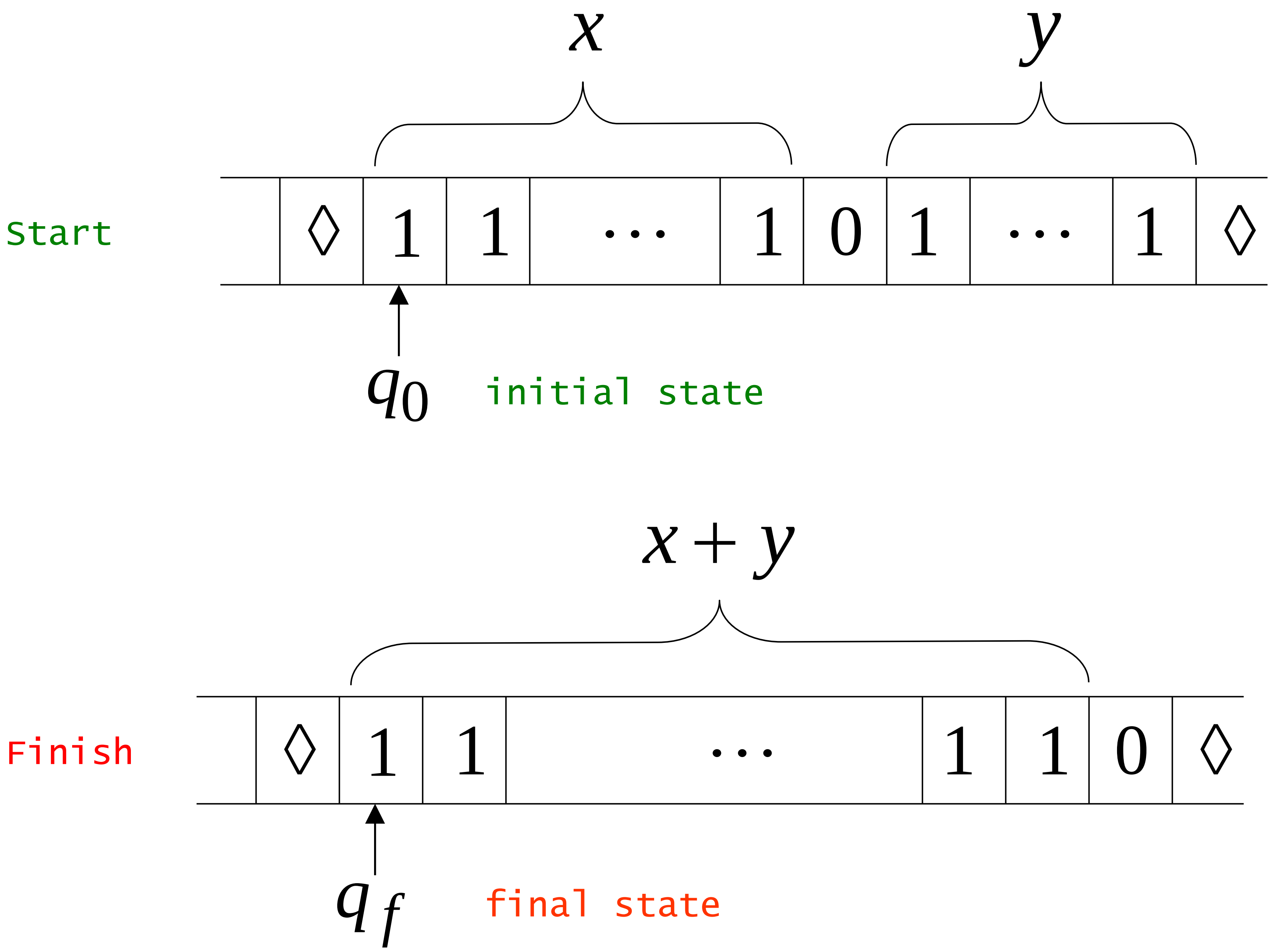
Start



initial state

The 0 is the delimiter that separates the two numbers

Example

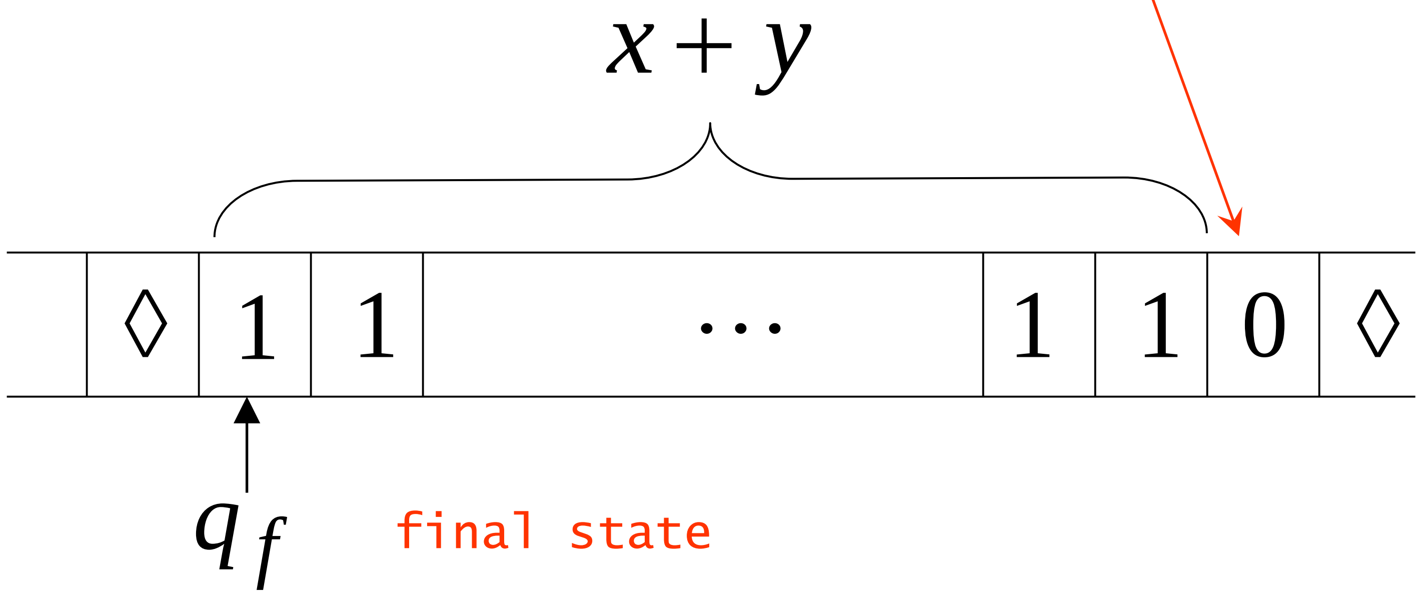


x
y
Start
◊ 1 1 ... 1 0 1 ... 1 ◊
q0
initial state
x + y
Finish
◊ 1 1 ... 1 1 0 ◊
qf
final state

Example

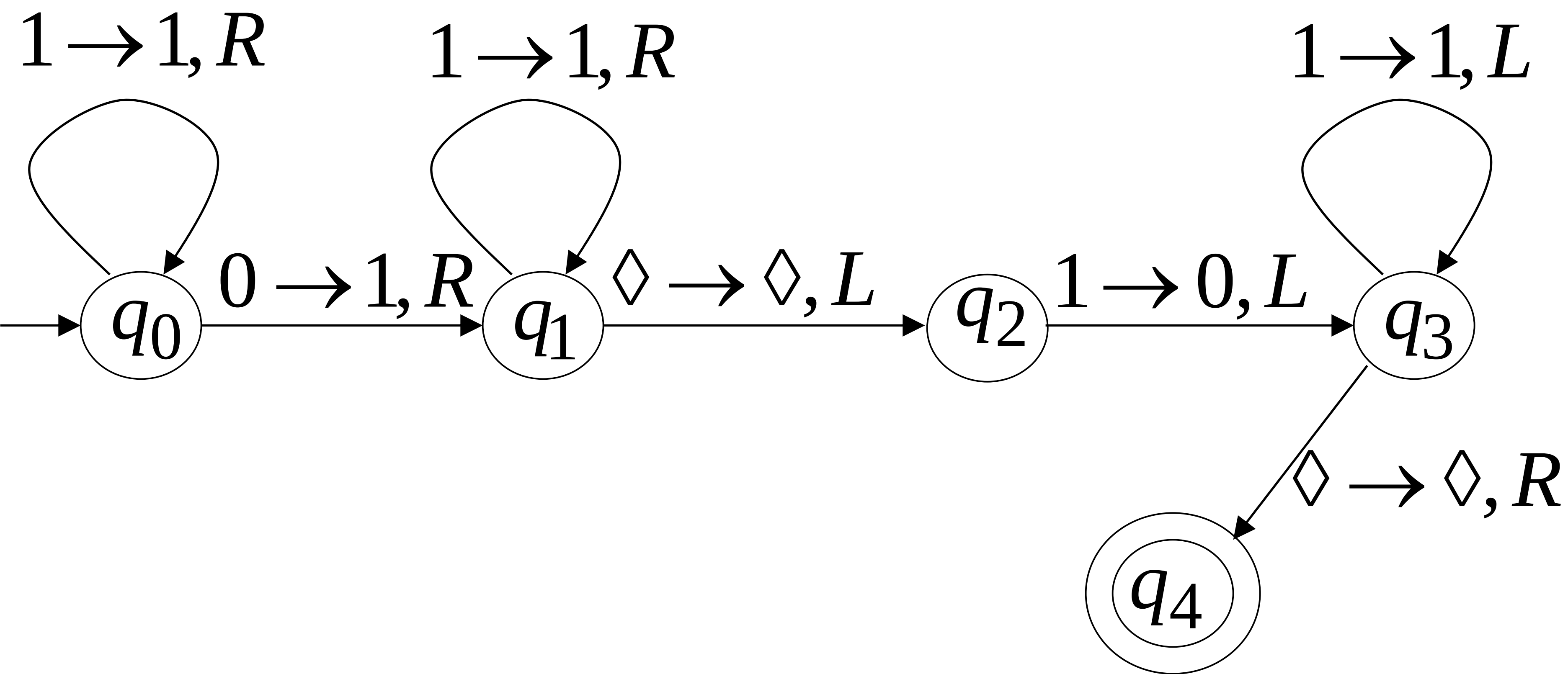
The 0 helps when we use the result for other operations

Finish



Turing machine example

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



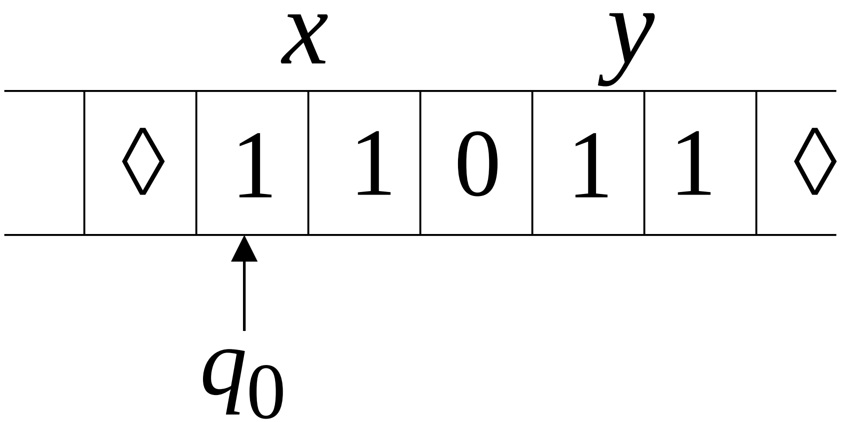
Turing machine example

Execution Example:

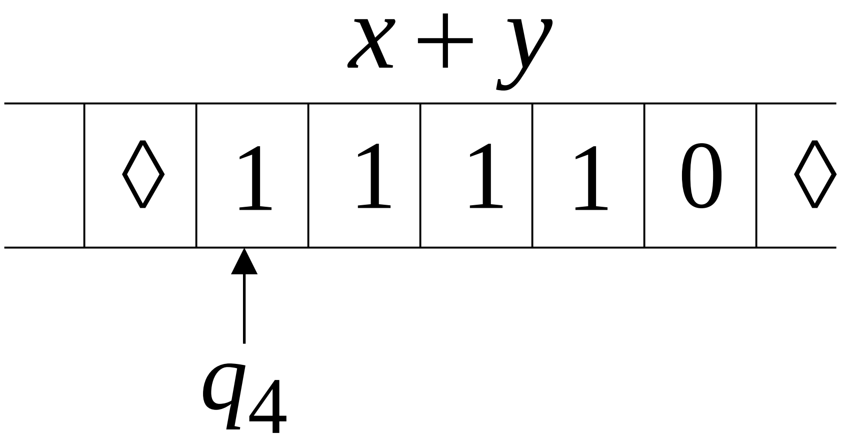
$$x = 11 \quad (2)$$

$$y = 11 \quad (2)$$

Time 0

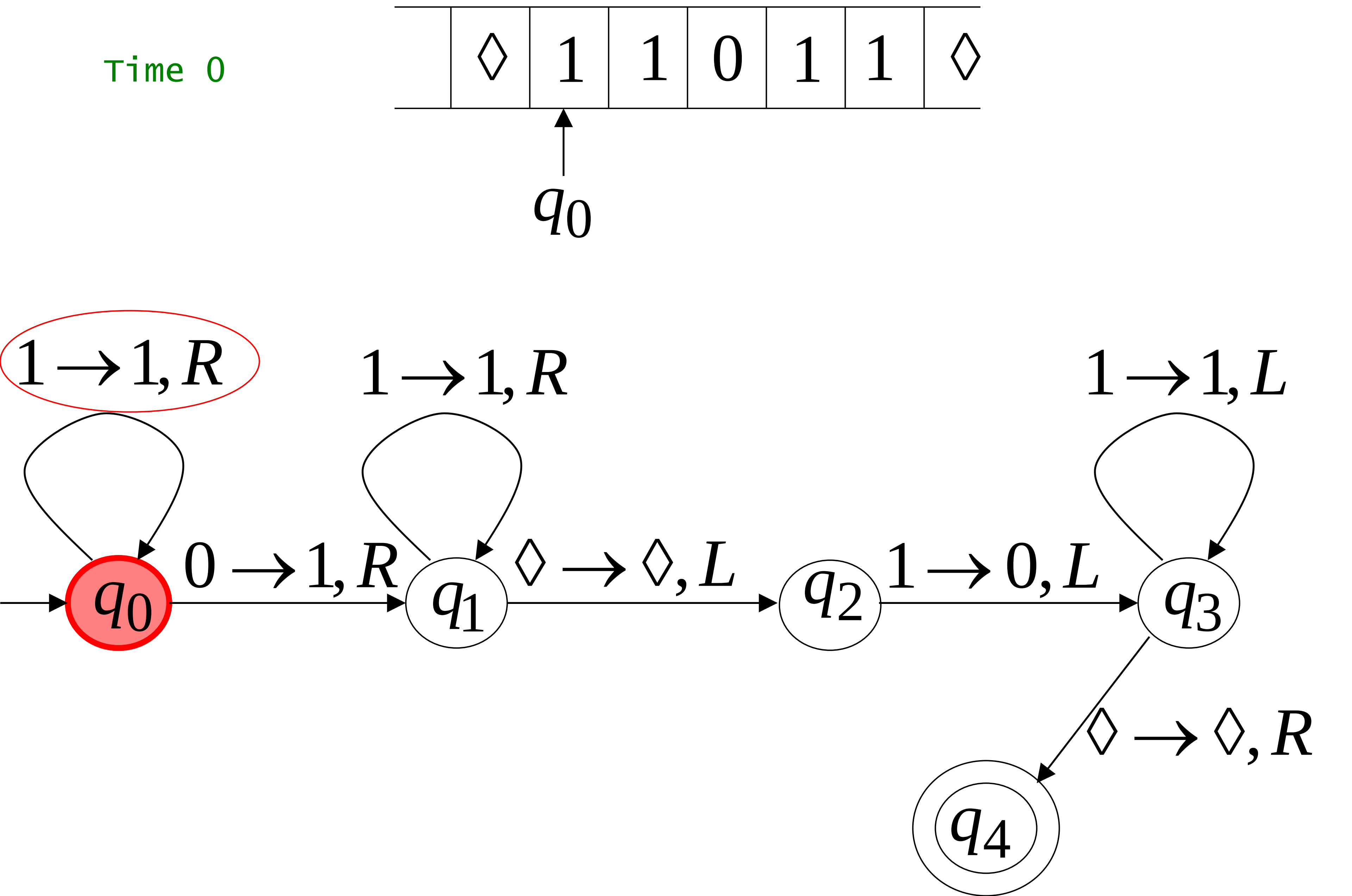


Final Result



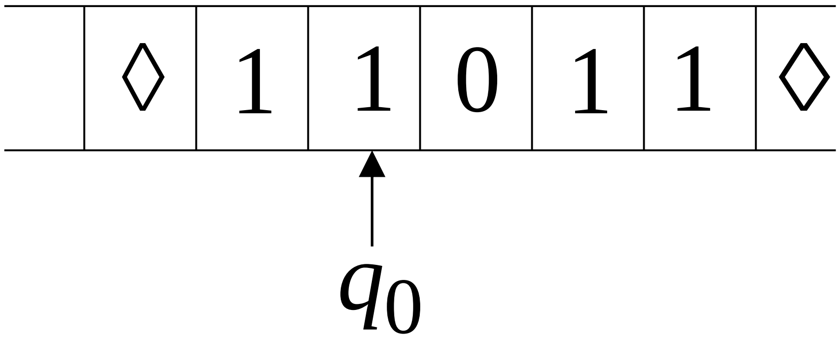
Turing machine example

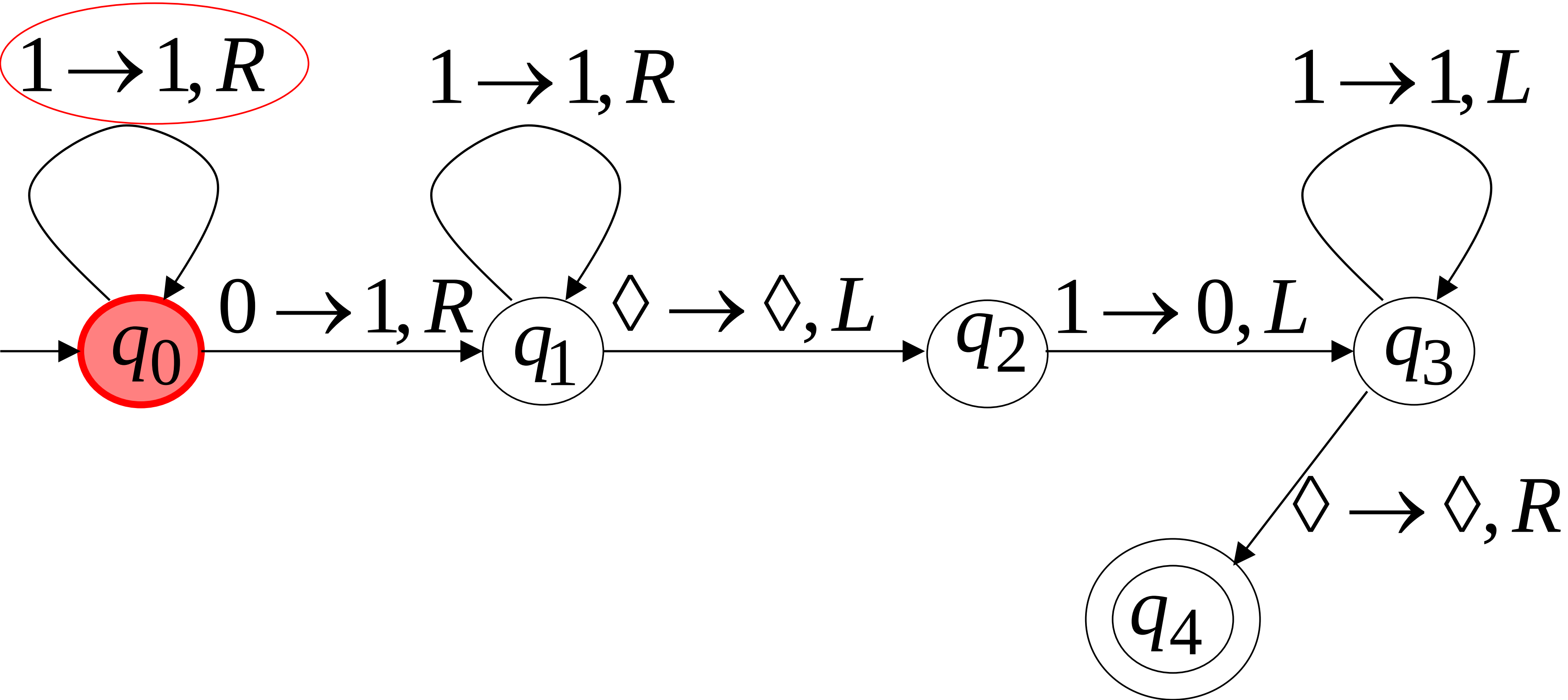
Time 0



Turing machine example

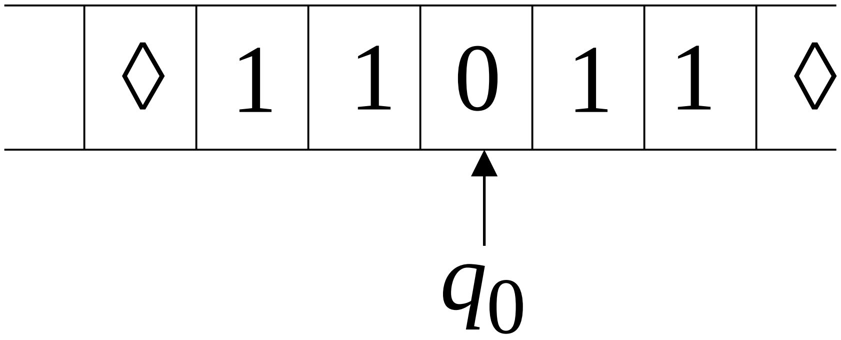
Time 1

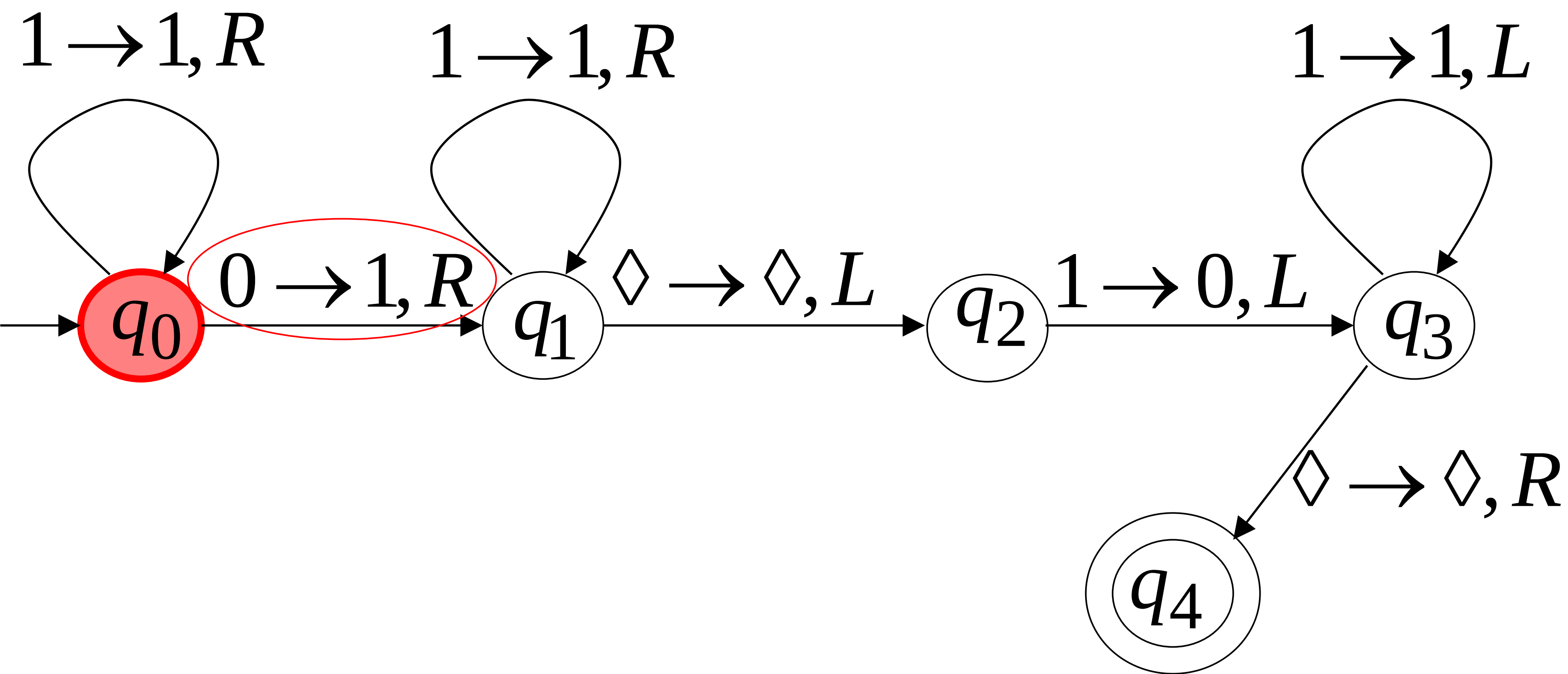




Turing machine example

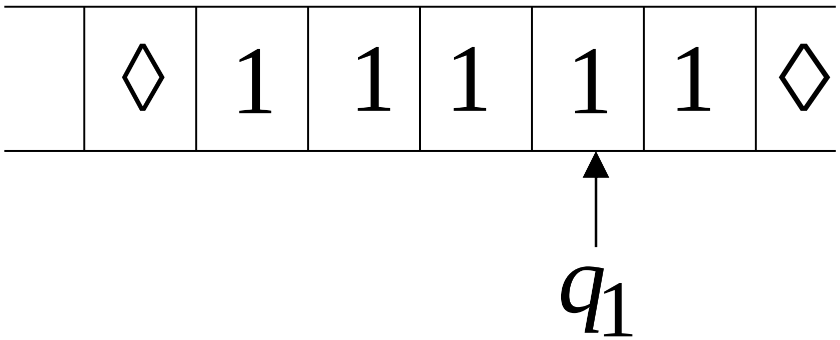
Time 2

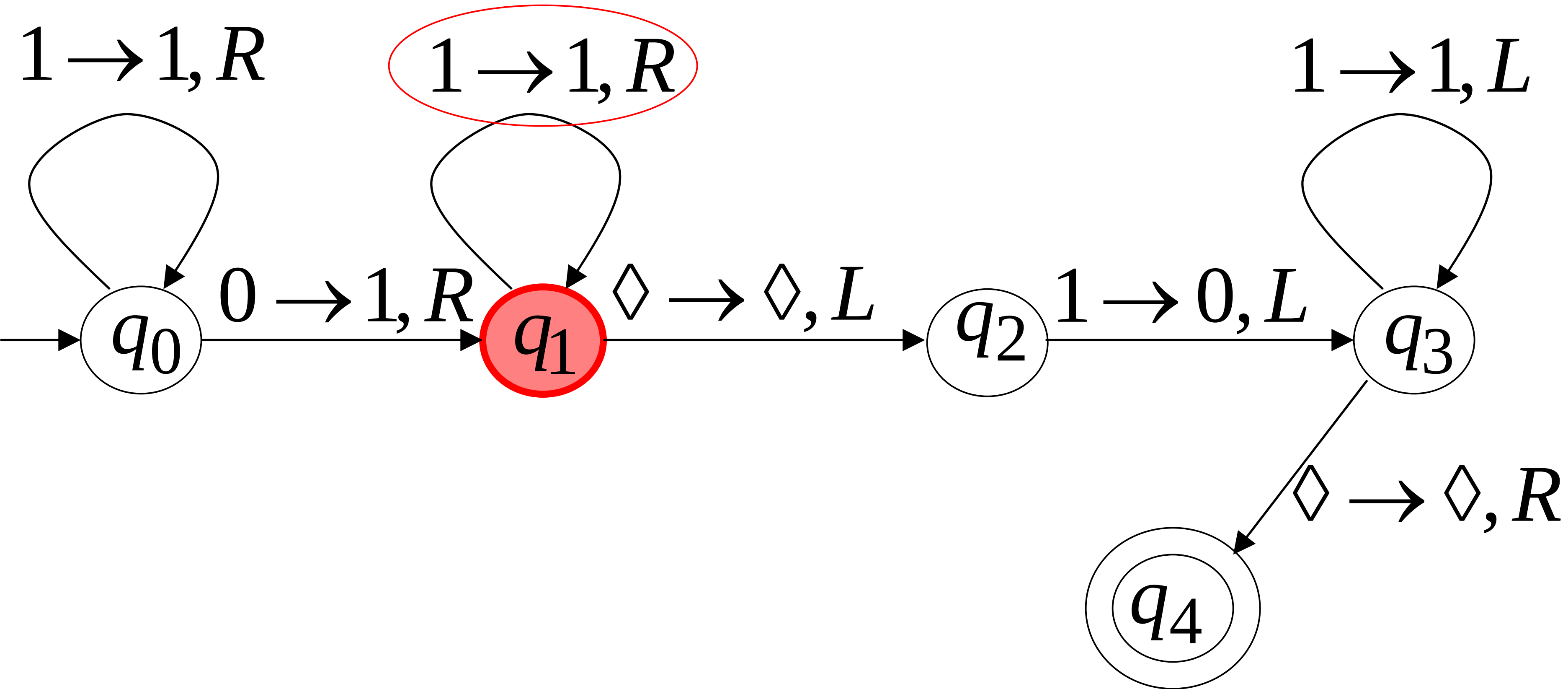




Turing machine example

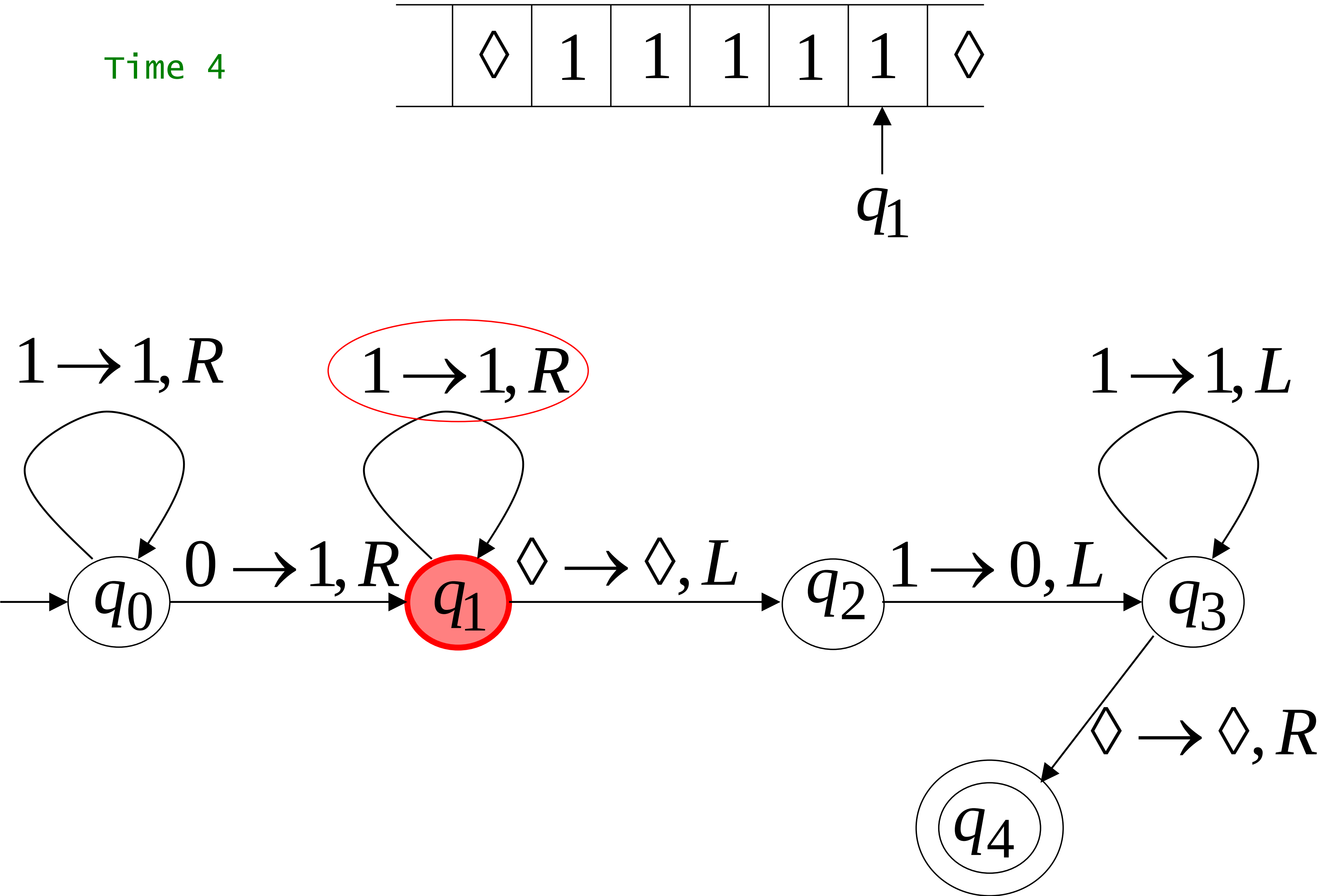
Time 3





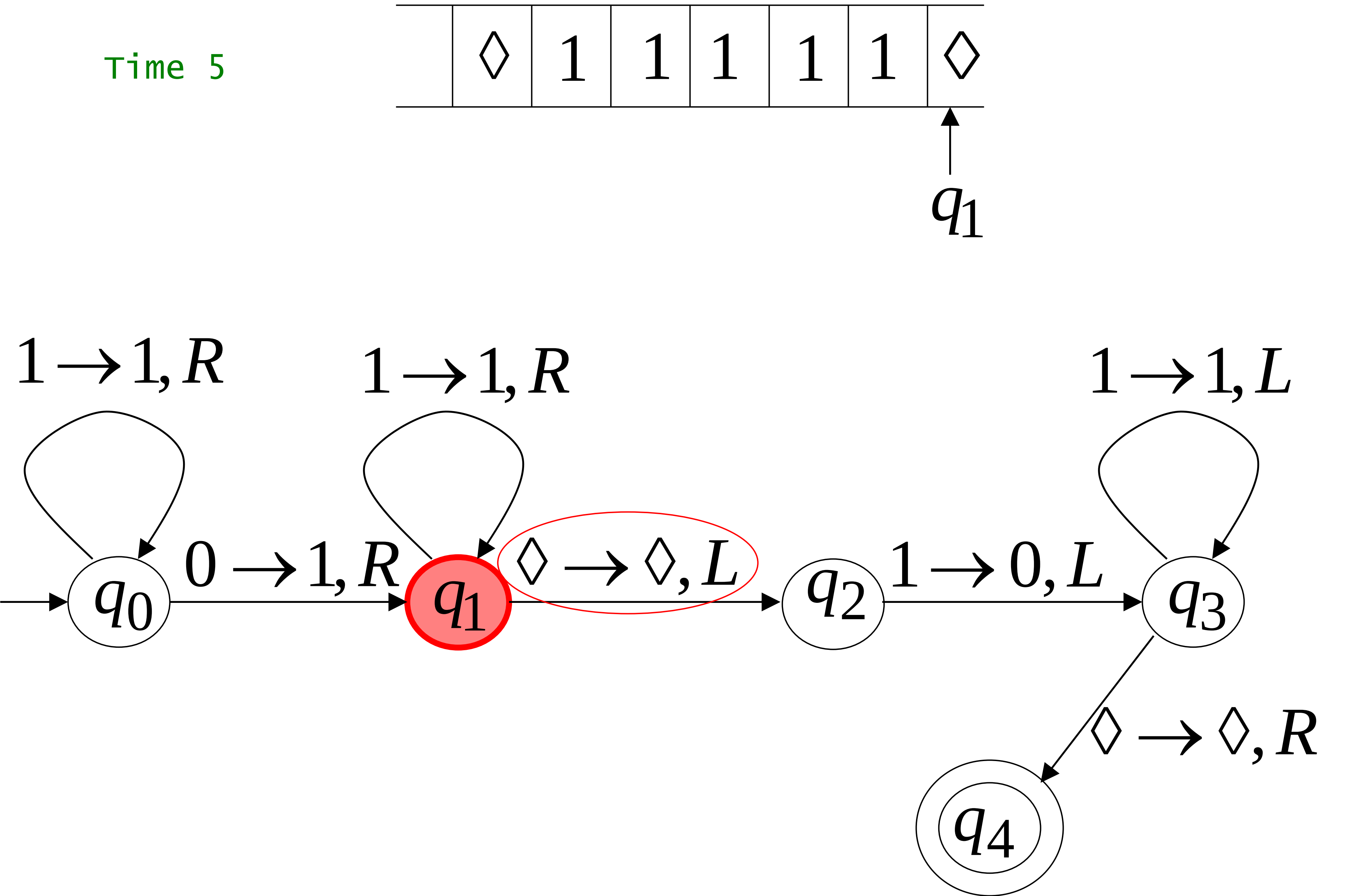
Turing machine example

Time 4



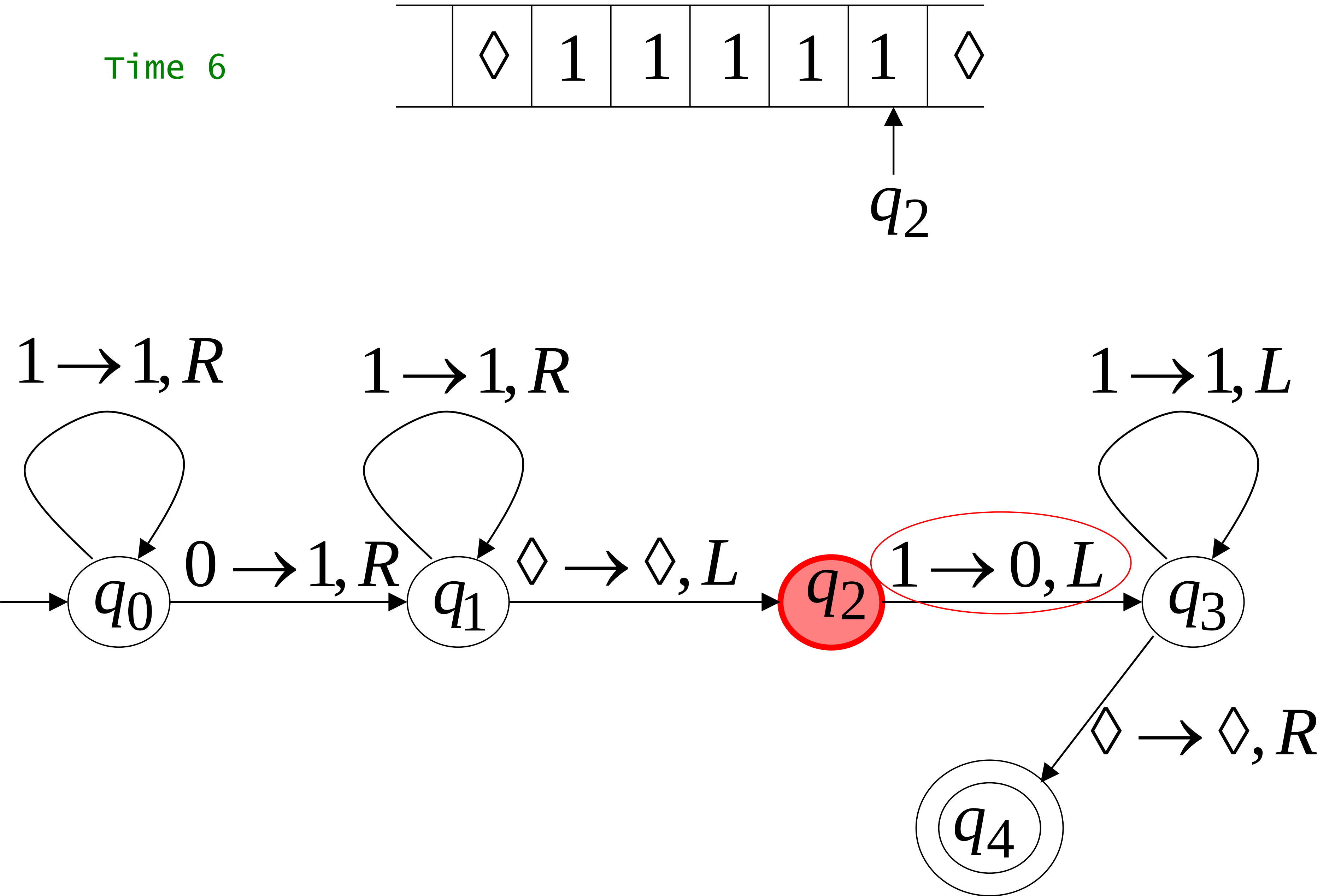
Turing machine example

Time 5

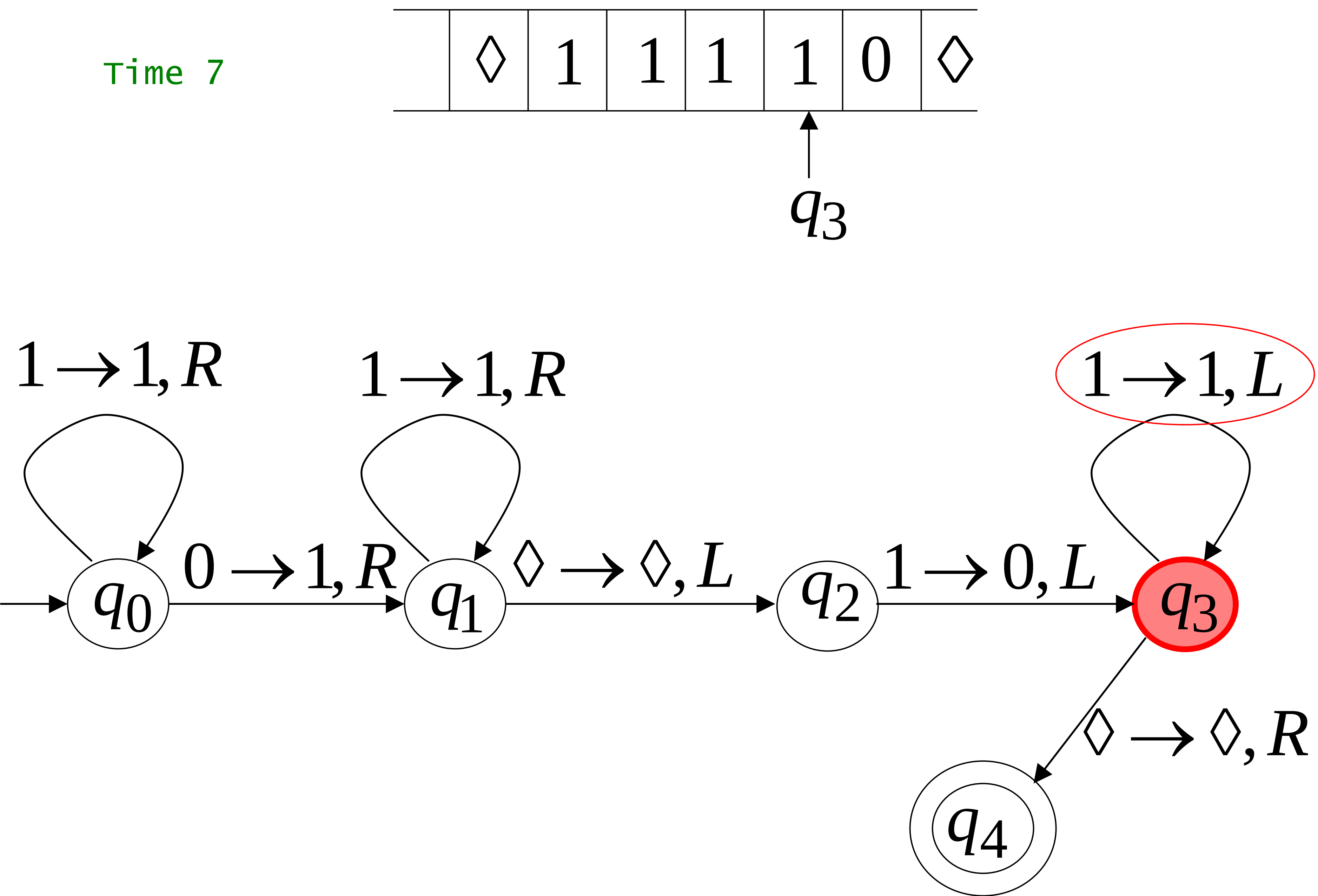


Turing machine example

Time 6

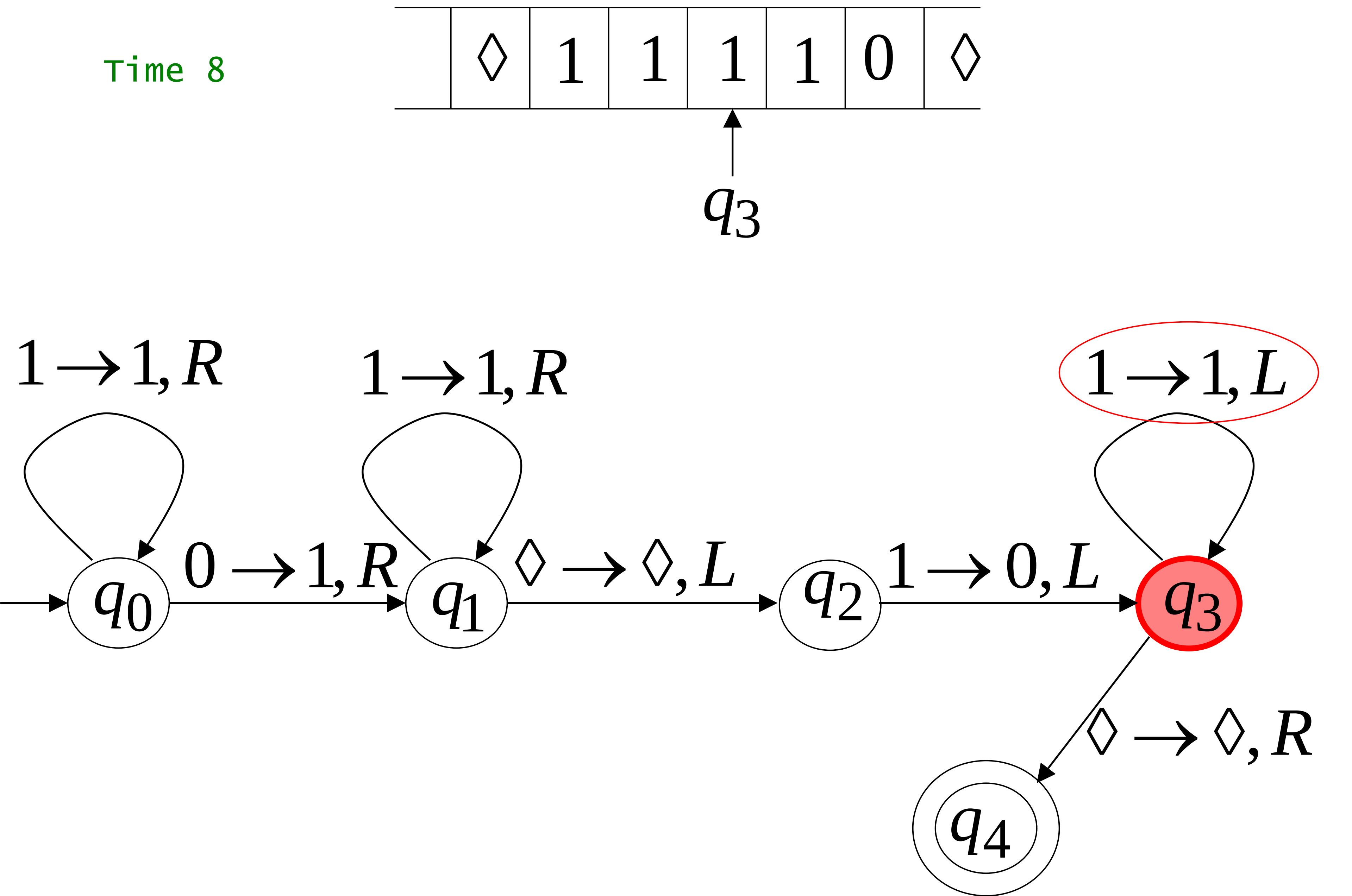


Turing machine example



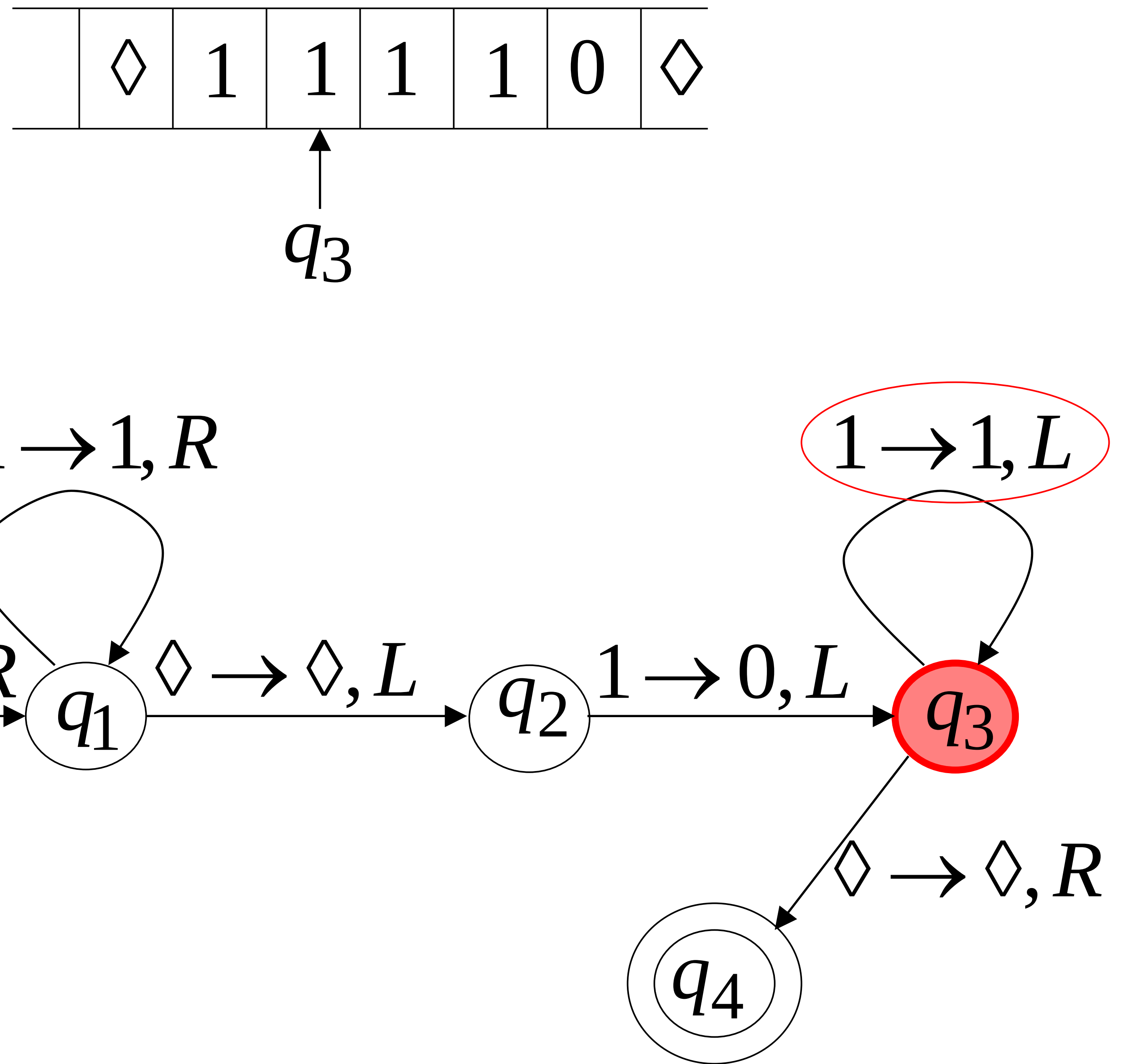
Turing machine example

Time 8



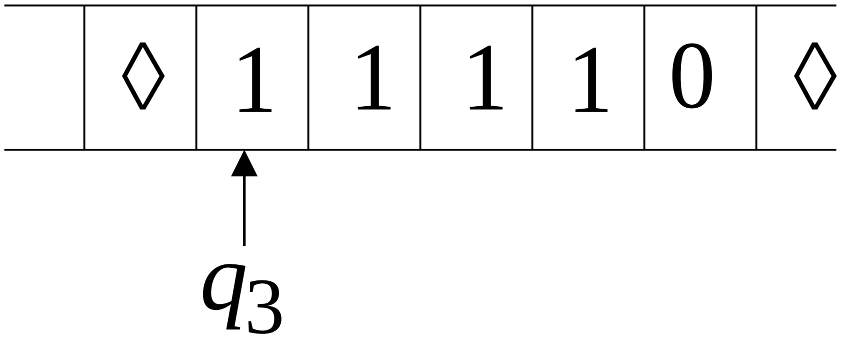
Turing machine example

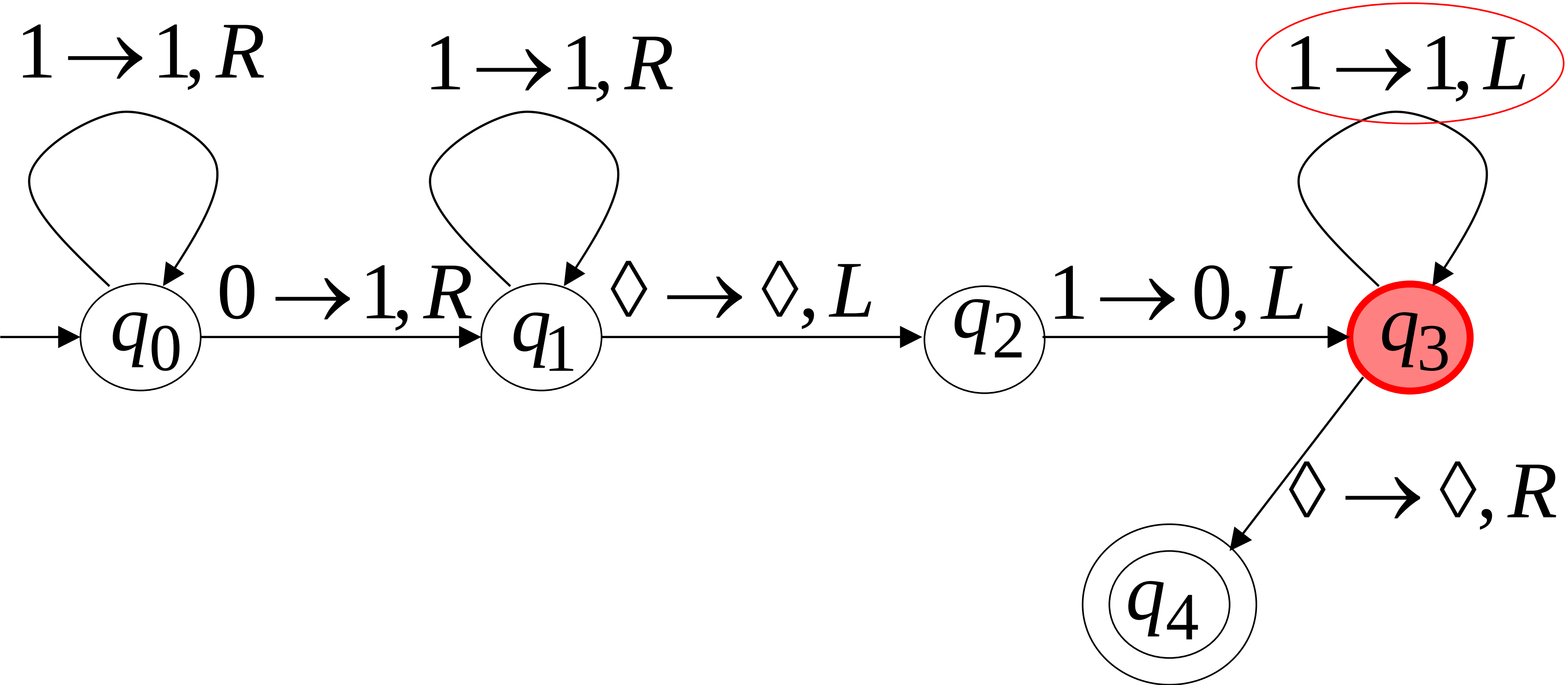
Time 9



Turing machine example

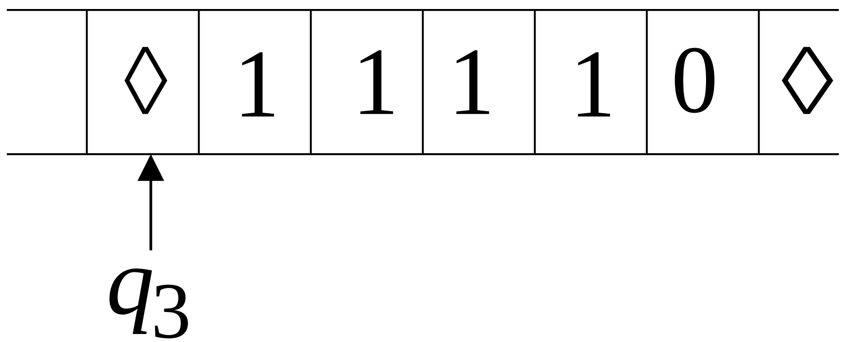
Time 10

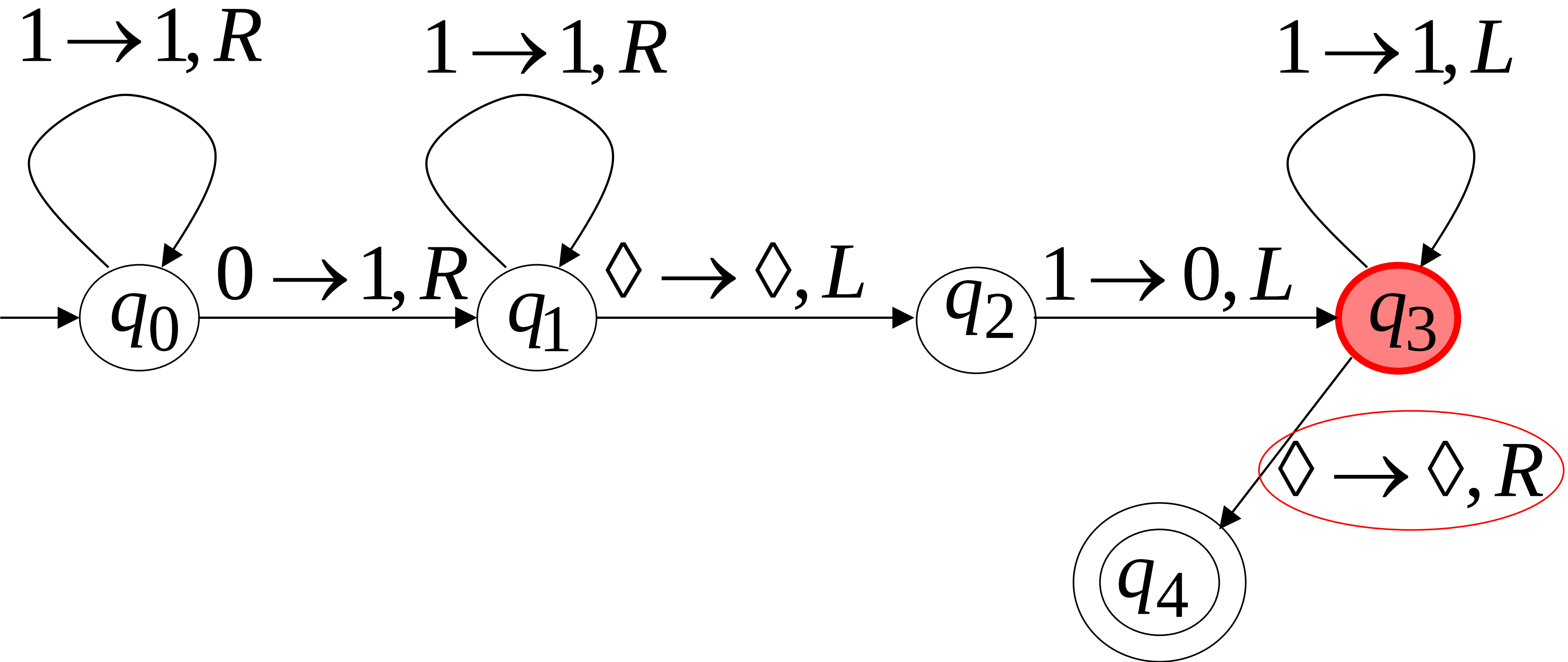




Turing machine example

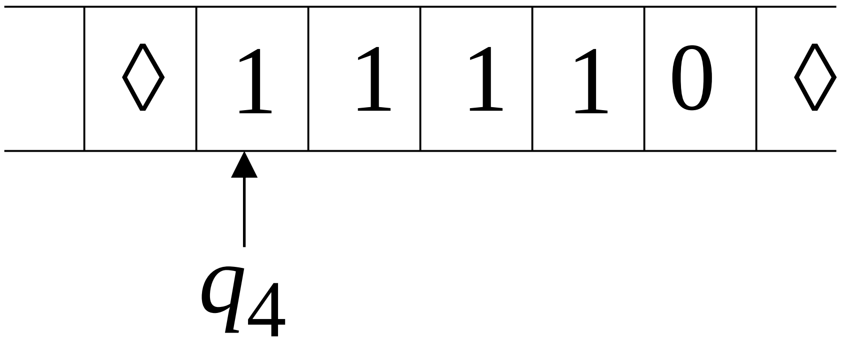
Time 11

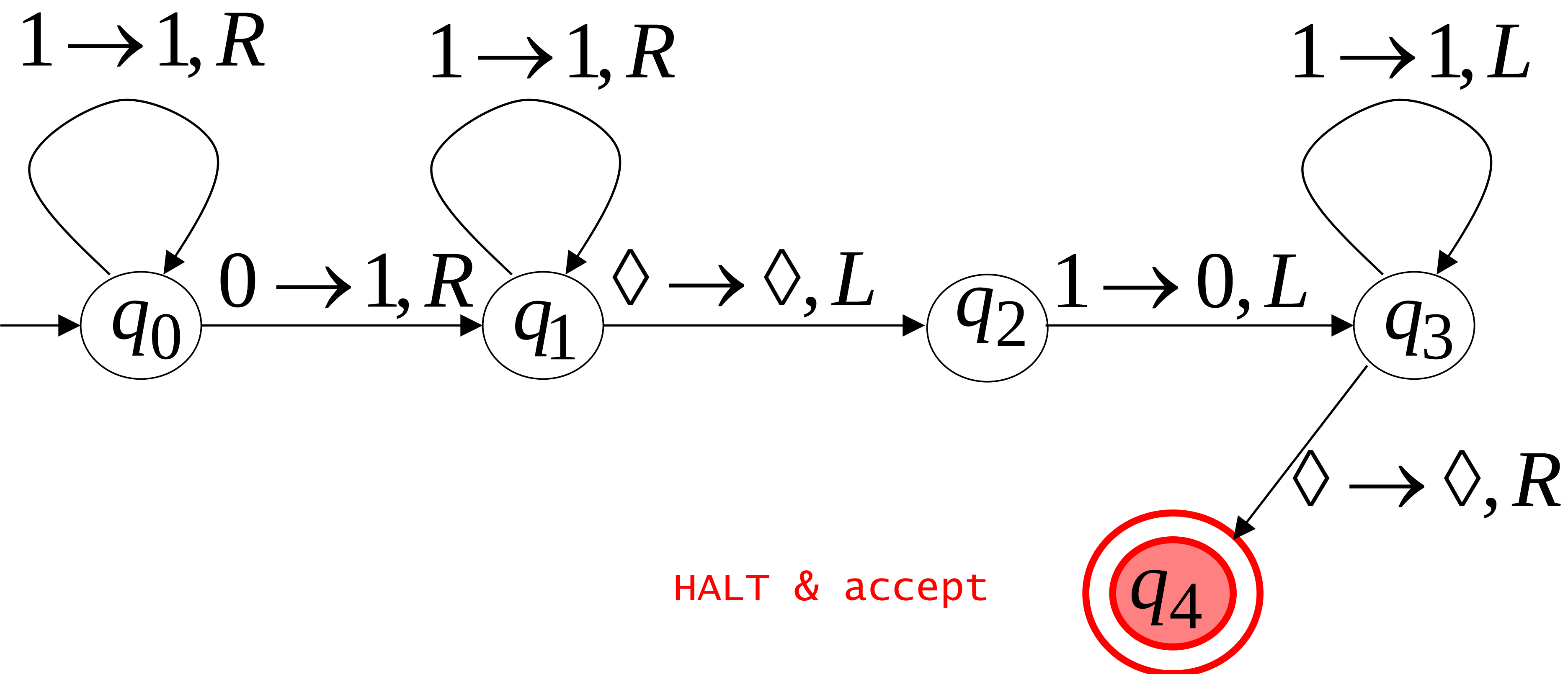




Turing machine example

Time 12





Multitape Turing Machine

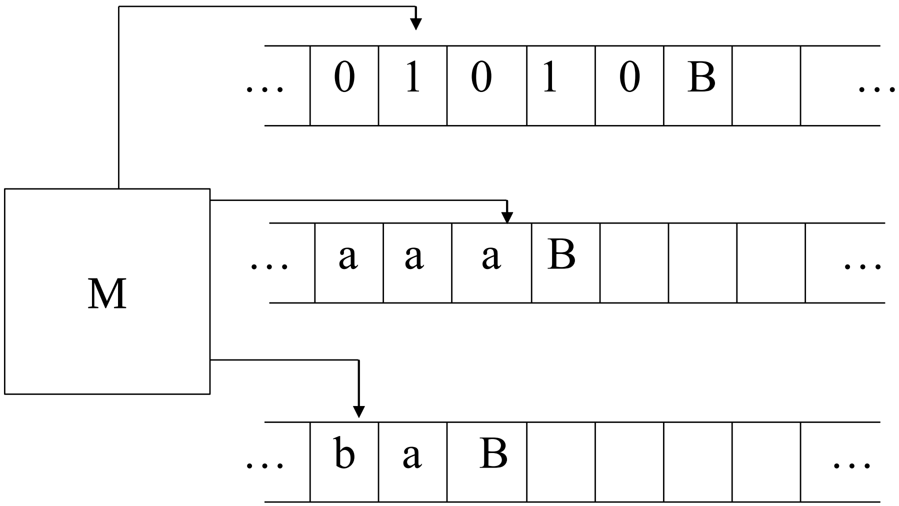
Multitape Turing Machines

- 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.

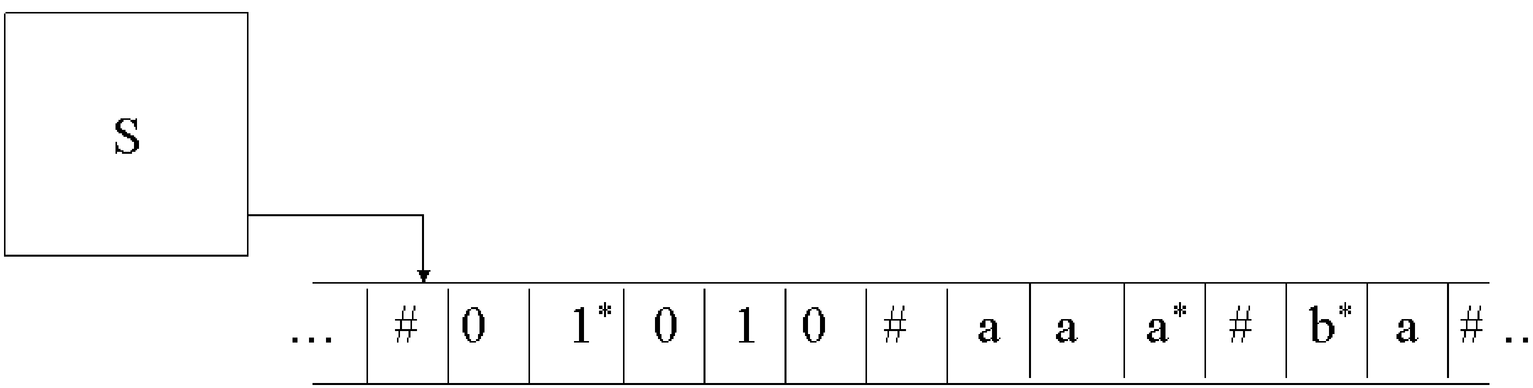
Multitape Turing Machine

- 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:



Single Tape Equivalent

- One final detail
  - If at any point S moves one of the virtual tape heads onto a #, then this action signifies that M has moved the corresponding head onto the previously unread blank portion of that tape.
  - To accommodate this situation, S writes a blank symbol on this tape cell and shifts the tape contents to the rightmost # by one, adds a new #, and then continues back where it left off

A Universal Turing Machine

A limitation of Turing Machines:

$$\text{Turing Machines are “}\underbrace{\text{hardwired}}_{\substack{\text{they execute}\\ \text{only one program}}}\text{“}$$

Real Computers are re-programmable

Solution: **Universal Turing Machine**

Attributes:

- Reprogrammable machine
- Simulates any other Turing Machine

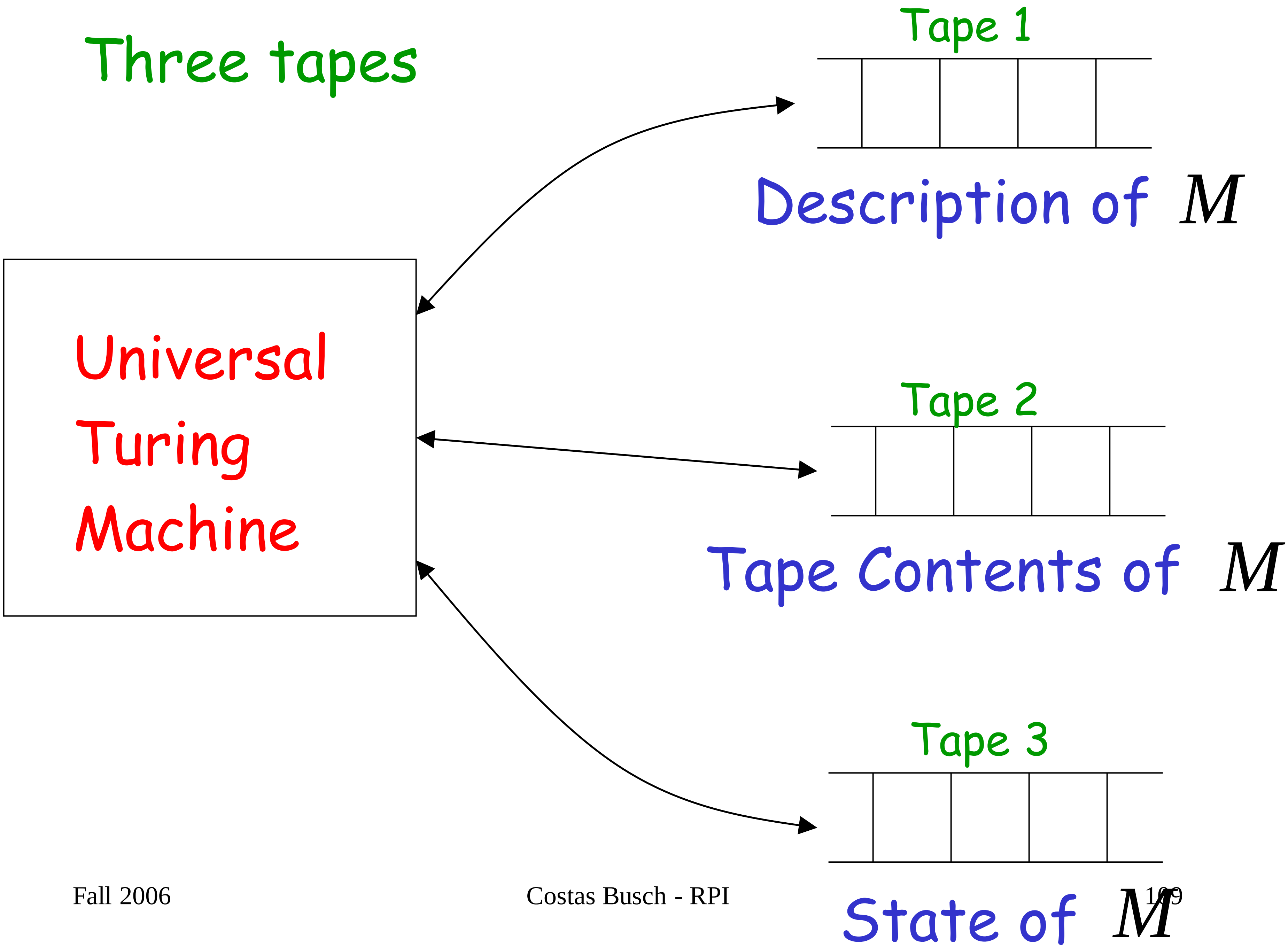
Universal Turing Machine

simulates any Turing Machine M

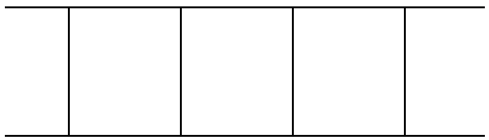
Input of Universal Turing Machine:

Description of transitions of M

Input string of M


Three tapes
Universal
Turing
Machine
Tape 1
Description of M
Tape 2
Tape Contents of M
Tape 3
State of M

Tape 1



Description of M

We describe Turing machine M
as a string of symbols:

We encode M as a string of symbols

Alphabet Encoding

Symbols:	a	b	c	d	$\cdots$
Encoding:	1	11	111	1111	

State Encoding

States:	q_1	q_2	q_3	q_4	$\cdots$
Encoding:	1	11	111	1111	

Head Move Encoding

Move:	L	R
Encoding:	1	11

Transition Encoding

Transition: $\delta(q_1, a) = (q_2, b, L)$

Encoding: $1\,0\,1\,0\,11\,0\,11\,0\,1$

separator

Turing Machine Encoding

Transitions:

$$\delta(q_1, a) = (q_2, b, L) \qquad \delta(q_2, b) = (q_3, c, R)$$

Encoding:

1 0 1 0 1 1 0 1 1 0 1 00 1 1 0 1 1 0 1 1 1 0 1 1 1 0 1 1

separator

Tape 1 contents of Universal Turing Machine:

binary encoding
of the simulated machine M

Tape 1

1 0 1 0 11 0 11 0 10011 0 1 10 111 0 111 0 1100...

A Turing Machine is described with a binary string of 0's and 1's

Therefore:

The set of Turing machines forms a language:

each string of this language is the binary encoding of a Turing Machine