Course- FORMAL LANGUAGE & AUTOMATA THEORY
ITUA31204
T.Y. B.Tech (NEP 2023)

Department of Information Technology Engineering, VIIT, Pune-48

BRACT'S, Vishwakarma Institute of Information Technology, Pune-48

(An Autonomous Institute affiliated to Savitribai Phule Pune University)
(NBA and NAAC accredited, ISO 9001:2015 certified)

Course Objectives:

- Understand fundamentals of computer mathematics.
- Study the various abstract computing models.
- Study the Regular expression.
- Study the different types of languages & their relationships.
- Study the different types of grammar & ambiguity in the grammar
- Study the recursive & recursively enumerable languages..

Pre-requisites: Basic Mathematics, Discrete Structure, Data structures

Course Outcomes:

After completion of the course, student will be able to

1. Use the mathematical preliminaries with the help of proofs and lemmas for language derivation.

2. Understand Regular expression, its conversion to automata and its applications.

3. Understand different types of languages, grammars and removal of ambiguity in the grammar.

4. Construct pushdown automata for formal language and its applications.

5. Understand the concept of language acceptability by Turing machine and types of Turing machine.

6. Understand the recursive & recursively enumerable languages.

Text Books & Reference Books

- **Text Books**

•John C. Martin, Introduction to Language and Theory of Computation, TMH, 3rdEdition, ISBN: 978-0-07-066048-9.

•K.L.P Mishra,N. Chandrasekaran,Theory of Computer Science (Automata, Languages and Computation), Prentice Hall India, 2nd Edition.

- **Reference Books**

•Hopcroft J., Motwani R., Ullman J., "Introduction to Automata Theory, Languages and Computations", Third edition, Pearson Education Asia

•Michael Sipser, Introduction to the Theory of Computation,CENGAGE Learning, 3rd Edition, ISBN:13:978-81-315-2529-6.

•Daniel Cohen, Introduction to Computer Theory, Wiley India, 2nd Edition, ISBN: 9788126513345.

•Kavi Mahesh, Theory of Computation: A Problem Solving Approach, 1st Edition, Wiley-India, ISBN: 978-81-265-3311-4.

•Vivek Kulkarni, Theory of computation, Oxford University Press, ISBN-13: 978-0-19-808458-7.

Important Instructions

- Be ready Be Attentive
- Separate Notebook for TOC
- Join Google classroom -keep your profile picture there and use only VIIT email-id for FLAT communications and submissions.

Unit I Theory of automata

- Basic Mathematical Objects: Sets, Logic, functions, Relations, Strings.
- Introduction to Formal language, Basic concepts: Symbol, Alphabet, String
- Introduction to Finite Automata, State transition graph, Transition table, Acceptance of a string, Acceptance of a Language
- Deterministic finite Automata (DFA)-Formal Definition
- NonDeterministic finite Automata (NFA)-Formal Definition
- NonDeterministic finite Automata (NFA) with epsilon transition
- Equivalence of NFA and DFA, Conversion from NFA to DFA, Conversion from NFA with epsilon transition to DFA
- Finite Automata with output: Moore and Mealy Machine, Moore to Mealy conversion, Mealy to Moore conversion

FLAT Covers

•Course is about models of computation, their power, and relationships.

•Hierarchy of models of increasing power:

-DFA < PDA < TM.

For each model, we study:

-What can be computed.

-What cannot be computed.

- Automata theory is the study of abstract computing devices or "machines"

Automata theory

- The study of **abstract 'mathematical' machines** or systems and the computational problems that can be solved using these machines.
- These abstract machines are called automata.
- Automata theory is also closely related to **formal language** theory, as the automata are often classified by the class of formal languages they are able to recognize.
- An automaton can be a finite representation of a formal language that may be an infinite set.
- *Automata are used as theoretical models for computing*

History of TOC

- 1930-45: Alan Turing studied Turing machine

  Describe boundary between what a computing machine could do and what it could not do
- 1940-50: Researchers studies Finite Automata
- Late 1950s: N. Chomsky studied formal "Grammars"
- 1969-70: S. Cook extended Turing's study

  Separated problems that can be solved efficiently by computers and the problems that can be solved theoretically, but takes so much time by computers to solve

Application of TOC

- Finite Automata:
  - Software for designing and checking behavior of digital circuits
  - The “Lexical Analyzer” of a compiler
  - Software for scanning large bodies of text

    E.g. to find occurrences of words in collections of web pages
  - Software for verifying systems of all types that have a finite number of distinct states

    E.g. communication Protocols

Application of TOC (cont…)

- Turing Machine:
  - Prototype/Abstract of modern computer
  - To find out what a computer can do and what it can not do
  - Help us understand what we can expect from our software
  - To decide, whether we can solve the problem, which requires more time to solve, in less amount of time.

Basic Definitions

- **Alphabet** - a finite set of symbols.

  Notation: Σ .

  Examples: Binary alphabet {0,1},

  English alphabet {a,...,z,!,?,...}

- **String over an alphabet** Σ - a finite sequence of symbols from Σ.

  Notation: (a) Letters u, v, w, x, y, and z denote strings.

  (b) Convention: concatenate the symbols.

  E.g:

  No parentheses or commas used.

- **Language over alphabet Σ - a set of strings over Σ.**

  Notation: L. 0000 is a string over the binary alphabet.

  E.g: a!? is a string over the English alphabet.

  {0, 00, 000, ...} is an "infinite" language over the binary alphabet.

  {a, b, c} is a "finite" language over the English alphabet.

Basic Definitions (cont…)

- **Empty string:** e or ε denotes the empty sequence of symbols.

- **Empty Language:** Empty set of strings.

  Notation: Φ

- **Regular Language-** a finite set of symbols.

  L is called regular Language if there is some automaton that accepts L.

  E.g.:

  Σ* , Even length of strings, strings containing abba are regular Languages.

Basic Definitions (cont…)

- **Difference Between Formal Language and Natural Language**

Formal Language	Natural Language
1. All the rules are stated explicitly	1. Not necessarily rules to be stated explicitly. It is a medium of communication.
2. Considered as Symbols on papers	2. Expressions of ideas in Natural way.
3. In formal Languages we are interested in the form of strings of symbols and not meaning	3. We are interested in Meaning than only strings.

Basic Machine and Finite State Machine

- **Basic Machine:**
  - A machine that recognizes input set I and produces output set O, where I & O are finite.
  - Machine Function(MAF): I-> O
  - Deals with **what** output is generated, not **how** output is generated

- **Finite State Machine (FSM):**
  - A machine that has a finite number of internal states. This state in a machine alters suitably when the machine receives an input and generates the required output such a model is called a Finite State Machine(FSM).
  - Deals with what internal state the machine has, and how each state behaves on receiving different intermediate inputs
  - It consists of a pair of functions as
  - Machine Function (MAF): I × S-> O
  - State Function (STF): I × S-> S

Behavior of such machines can be completely determined.

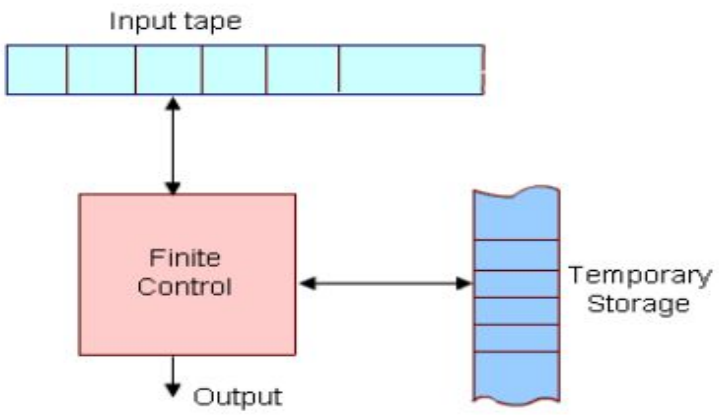
Finite Automata

- 5-tuple:
- M= (Q,Σ, δ, q_o,

  F) where,
  - Q: Finite set of states
  - Σ: Finite input alphabet
  - δ: STF that maps Q X Σ to Q, i.e. Q X Σ $\rightarrow$ Q
  - q_o: Initial state of FA, $q_o \in Q$
  - F: Set of final states, $F \subseteq Q$

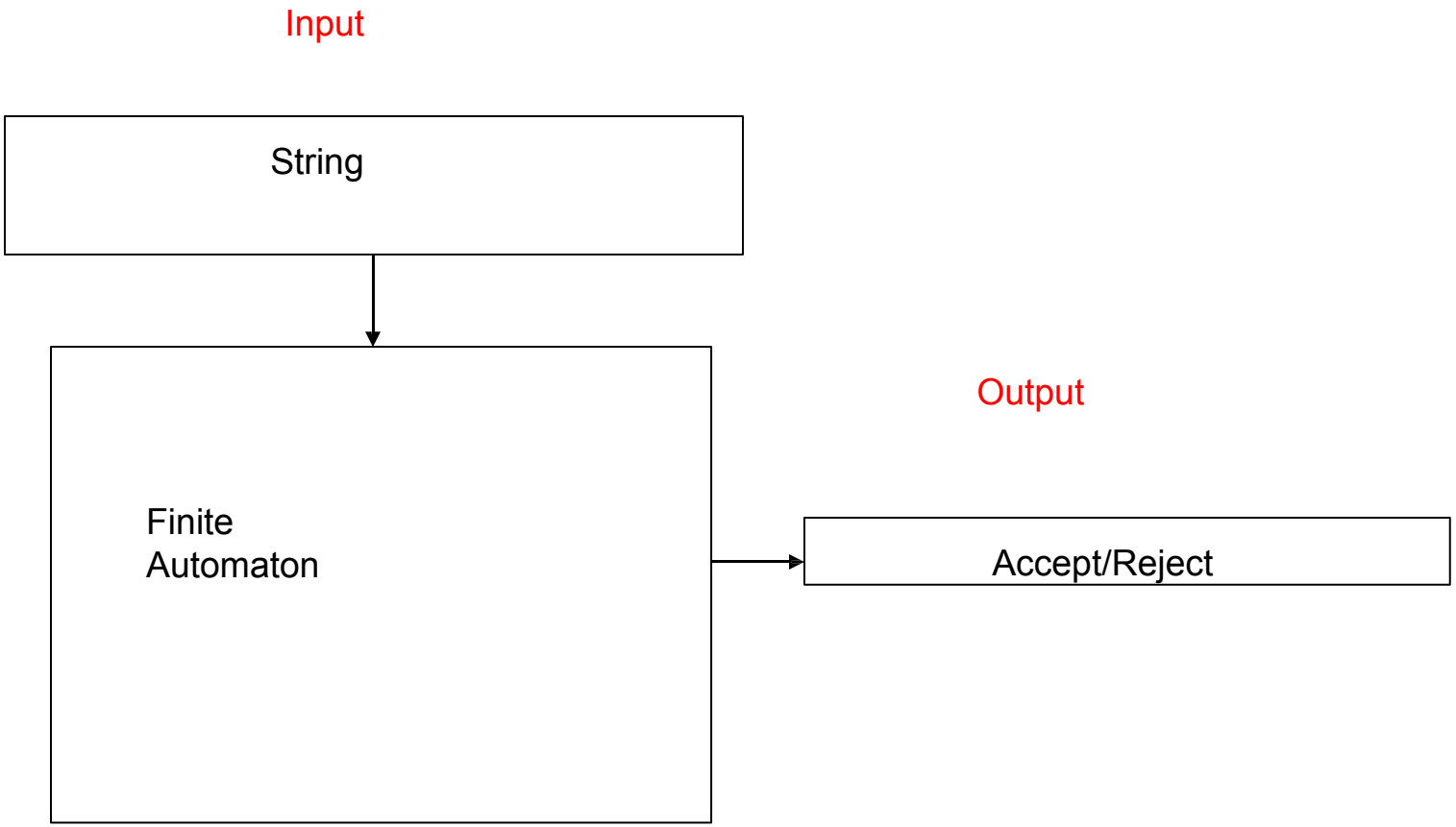
Finite Automata (cont…)

- Finite number of states
- No memory to store intermediate results
- Limited power due to lack of memory
- Input string is considered to be accepted, if the FA resides in any of the final states
- Input string is considered to be rejected, if the FA resides in any of the non-final states

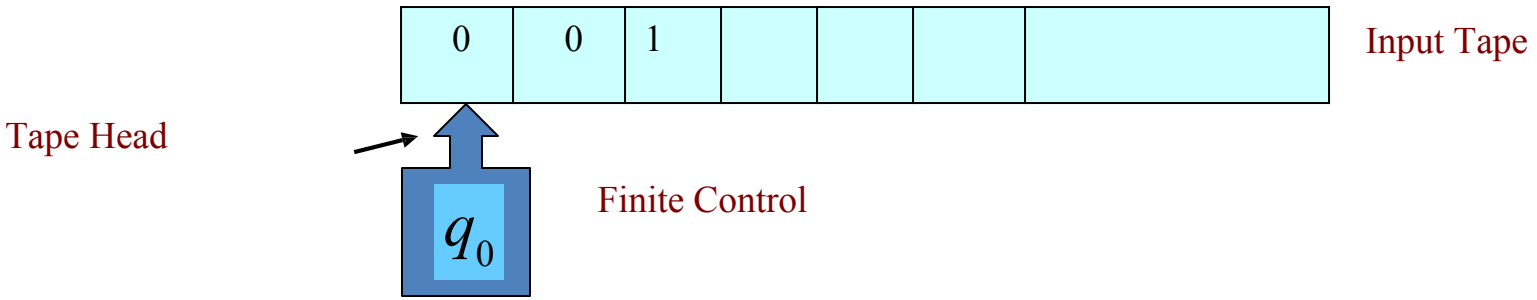


a diagrammatic representation of a generic automation.

Finite Automata (cont…)



Block diagram of FA



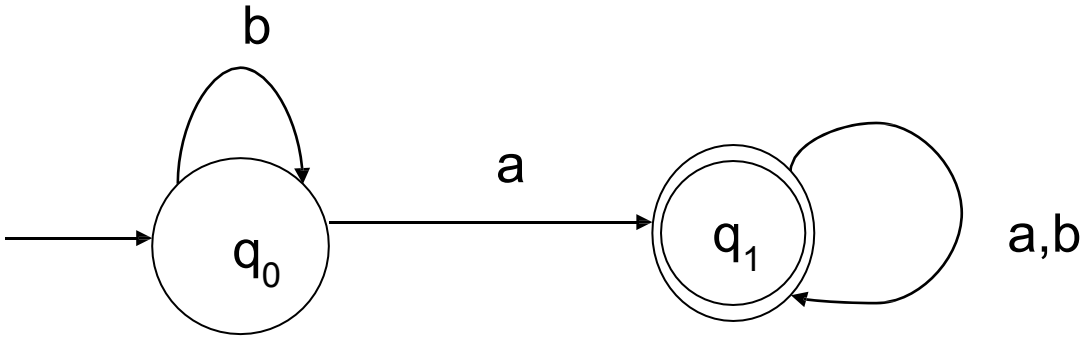
- Tape is divided into units ,each containing one symbol of the i/p string from Σ.
- Read the current letter of input under the tape head.
- Transit to a new state depending on the current input and the current state, as dictated by the transition function $\delta(q_i,a)=q_j$.
- Halt after consuming the entire input.

Designing FA

- Accept strings consisting of a & b, that has at least one ‘a’

i)Min string :a

ii)$\sum = \{a,b\}$



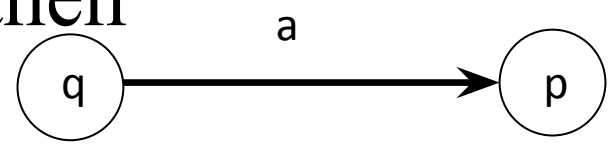
Preferred Notation for FA

- Transition Diagram
- Transition Table

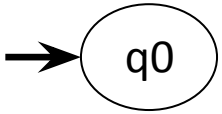
Transition Diagram

Transition diagram for FA(Q,Σ,q_0,F,δ) is a graph defined as:

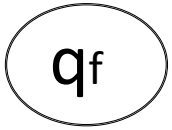
- For each state there is a node.
- For each state q in Q and for input symbol in Σ let $\delta(q,a)=p$ then



- There is arrow into start state.



- Final states marked by double circle.



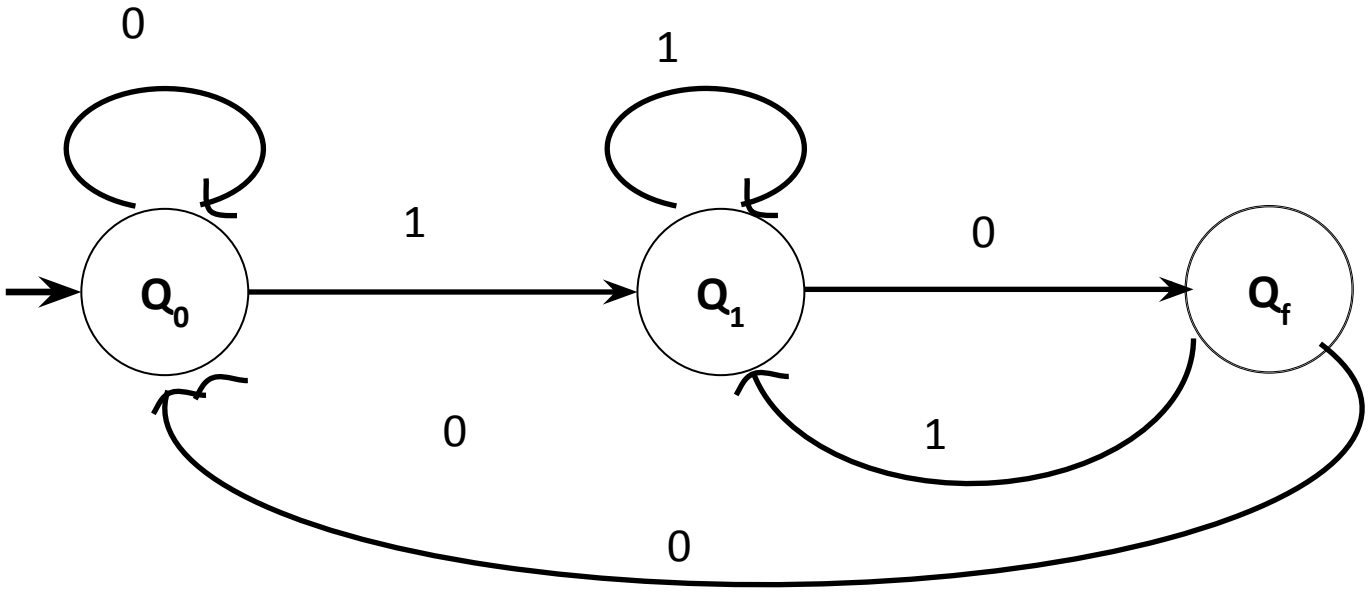
Transition Table

- A Transition table is a conventional, tabular representation of a function like δ that takes two arguments and return a state.
- The rows of the table correspond to the states, and the columns correspond to the input.
- The entry for one row corresponding to state q and the column corresponding to input a is the state $\delta(q, a)$. For Example:

	0	1
q_0	q_2	q_0
q_1	q_1	q_1
q_2	q_2	q_1

Finite Automata Example

- FA for language over {1,0} in which every string ends with 10.



FA Examples

1. Construct a FA to recognize language containing strings starting with ‘10’
2. Construct a FA to recognize language containing strings ending with ‘101’
3. Construct a FA to recognize language of strings with exactly two 0s anywhere
4. Construct a FA to recognize language containing strings starting with ‘a’
5. Construct a FA to identify string containing even number of ‘a’s over $\sum = \{a\}$

Acceptance of a String

- A string ‘x’ is said to be accepted by an FA (given by):

$M = (Q, \Sigma, \delta, q_o, F)$

if $\delta\ (q_o, x) = p$,

for some $p \in F$ (i.e. p is a member of F)

Acceptance of a Language

- If there is a language L such that:

$$L = \{x \mid \delta(q_o, x) = p, \text{ for some } p \in F\},$$

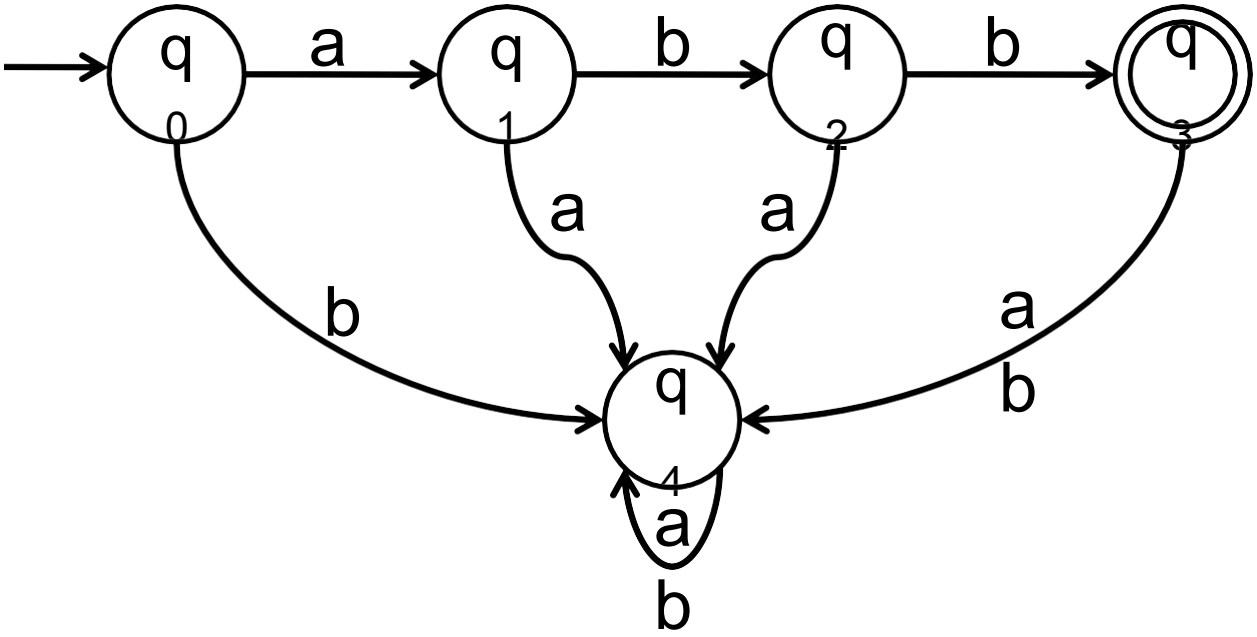
Then it is said to be accepted by the FA, M

Deterministic Finite Automata

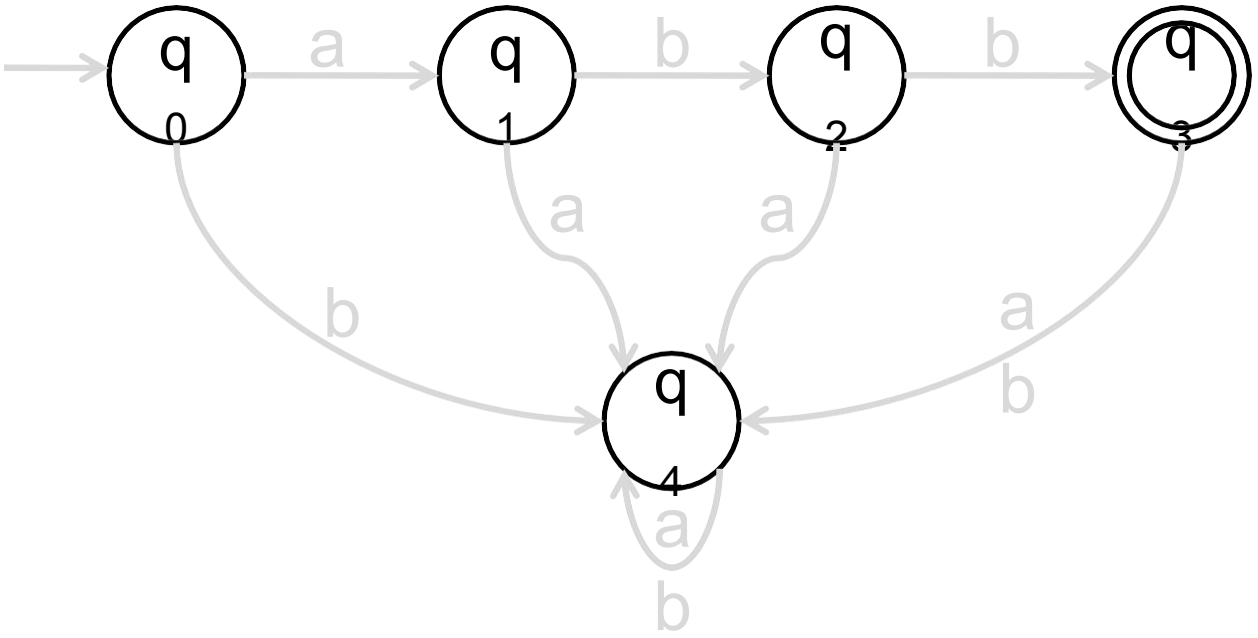
- An FA is said to be deterministic, if for every state there is a unique input symbol, which takes the state to the required next unique state
- Given a state Sj, the same input symbol does not cause the FA to move into more than one state-there is always a unique next state for an input symbol

DFA parts

TRANSITION GRAPH

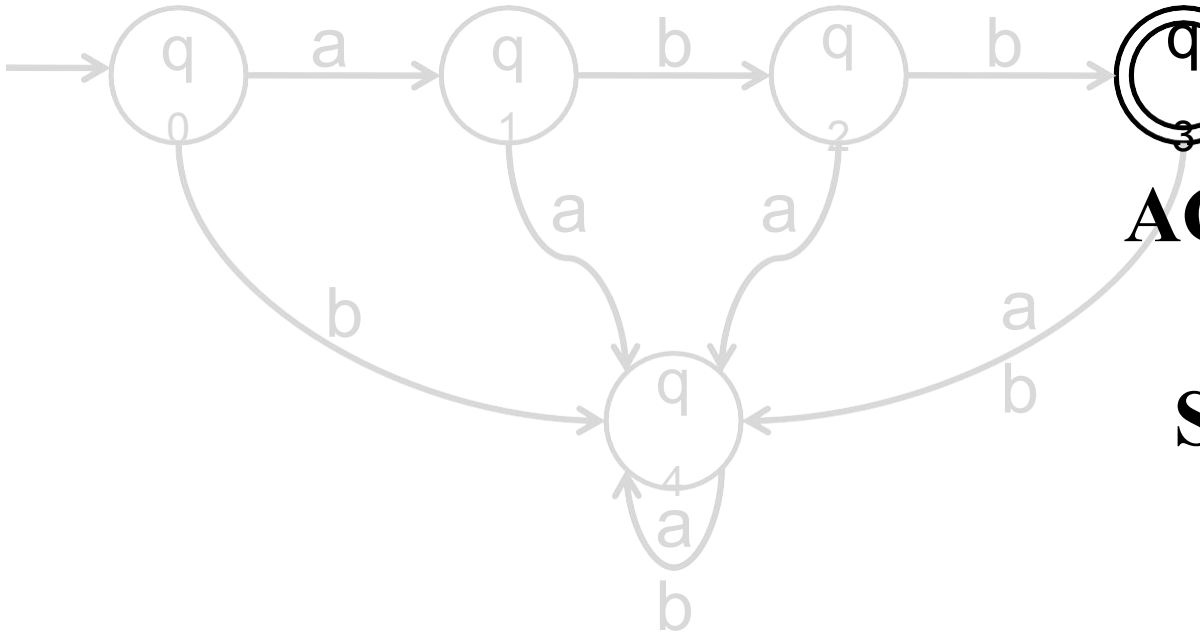


DFA parts



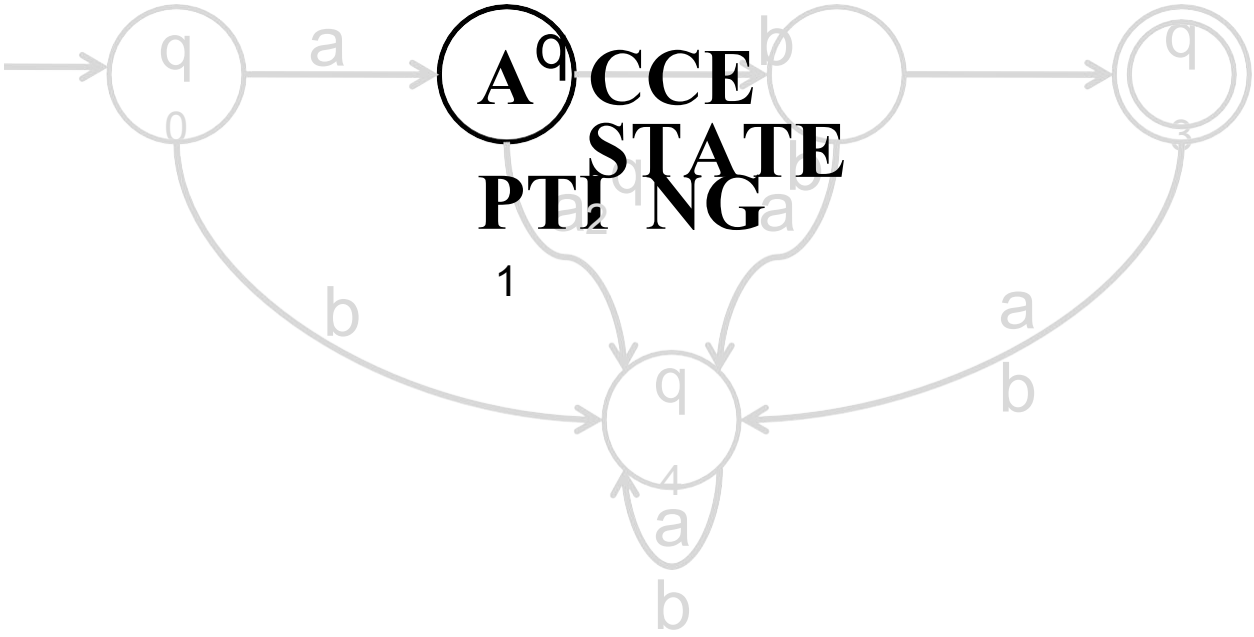
STATES : $\{q_0, q_1, q_2, q_3, q_4\}$

DFA parts

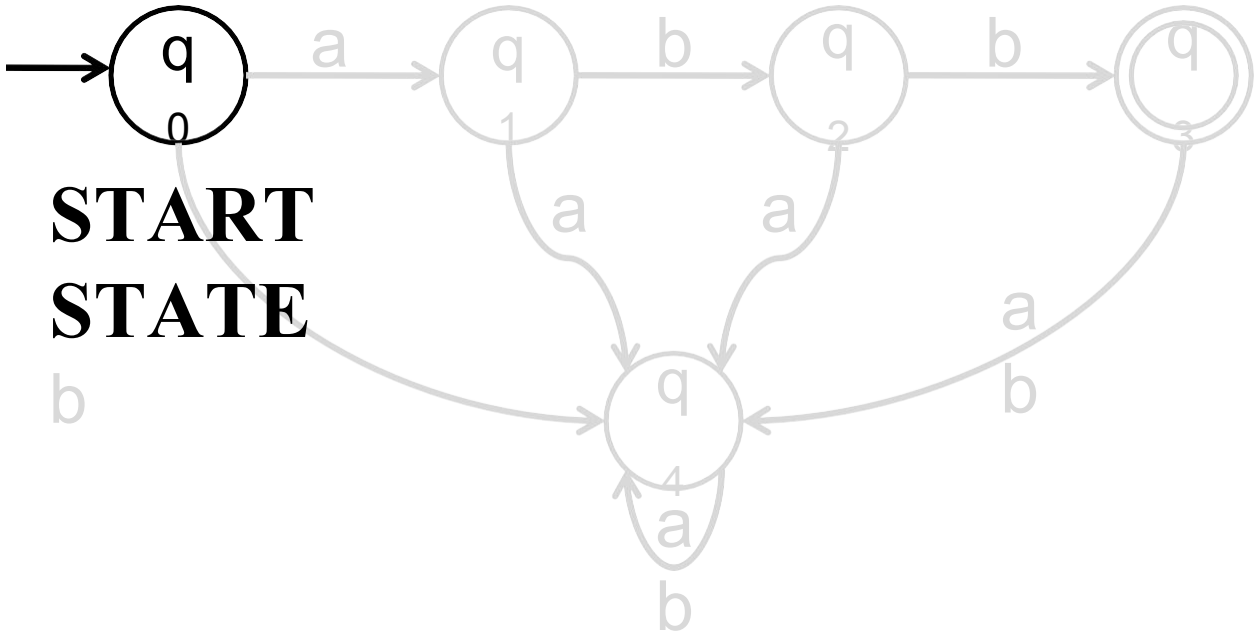


ACCEPTI NG STATE

DFA parts



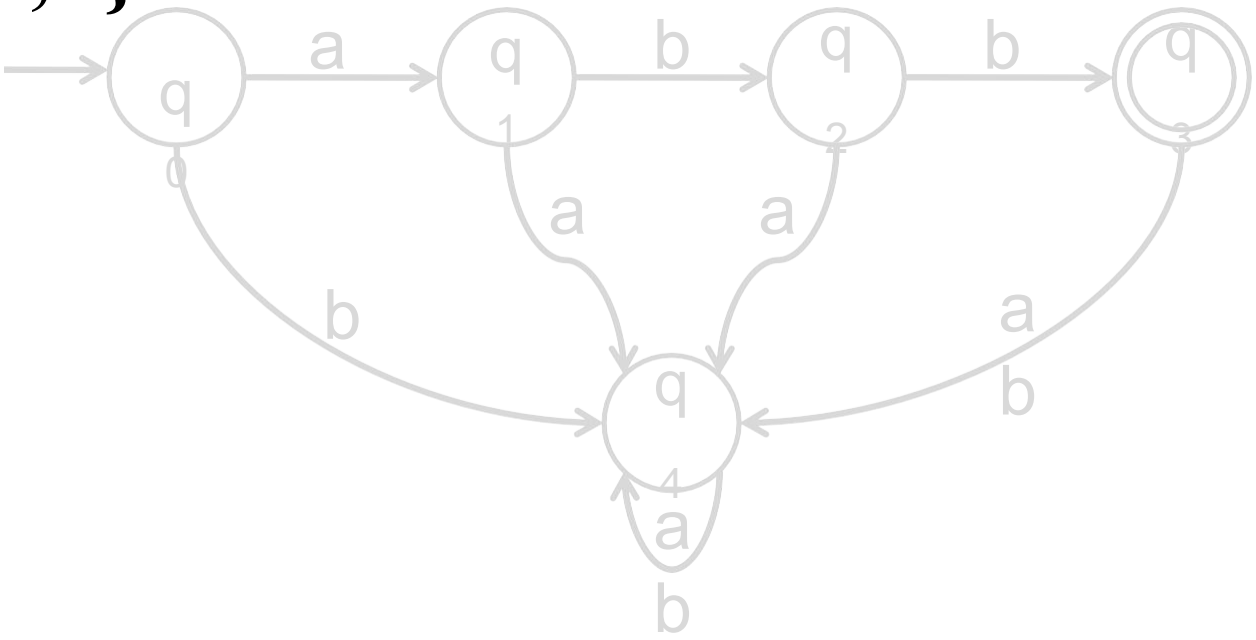
DFA parts



DFA parts

ALPHABET:

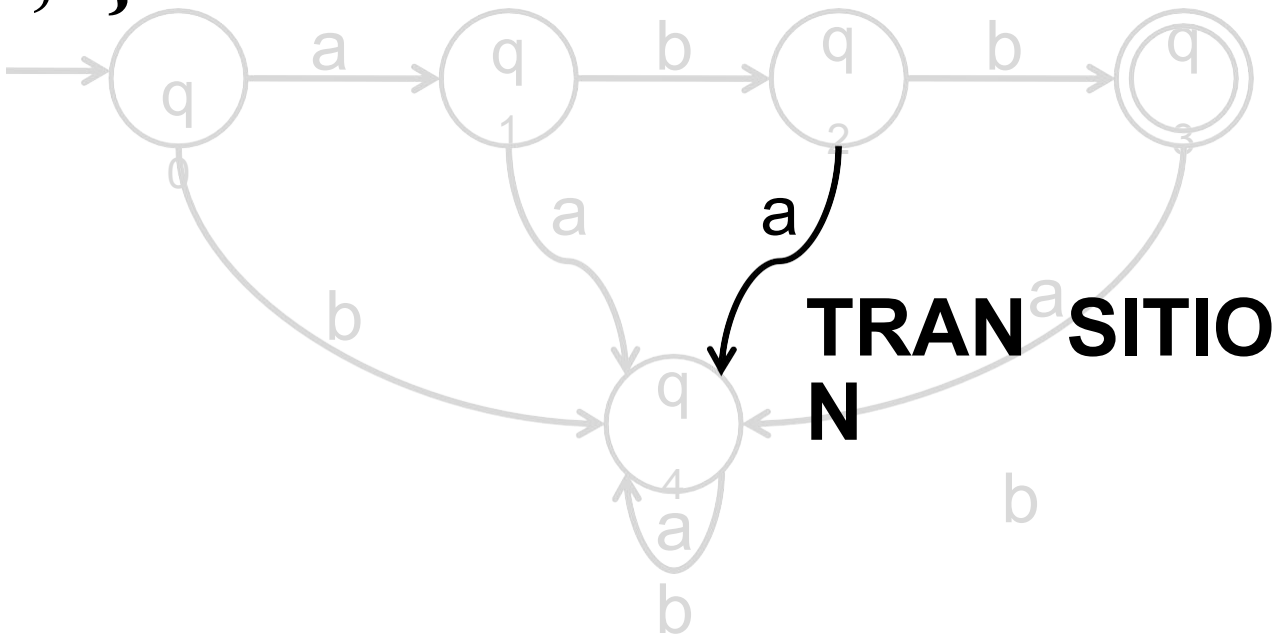
Σ = {a,b}



DFA parts

ALPHABET:

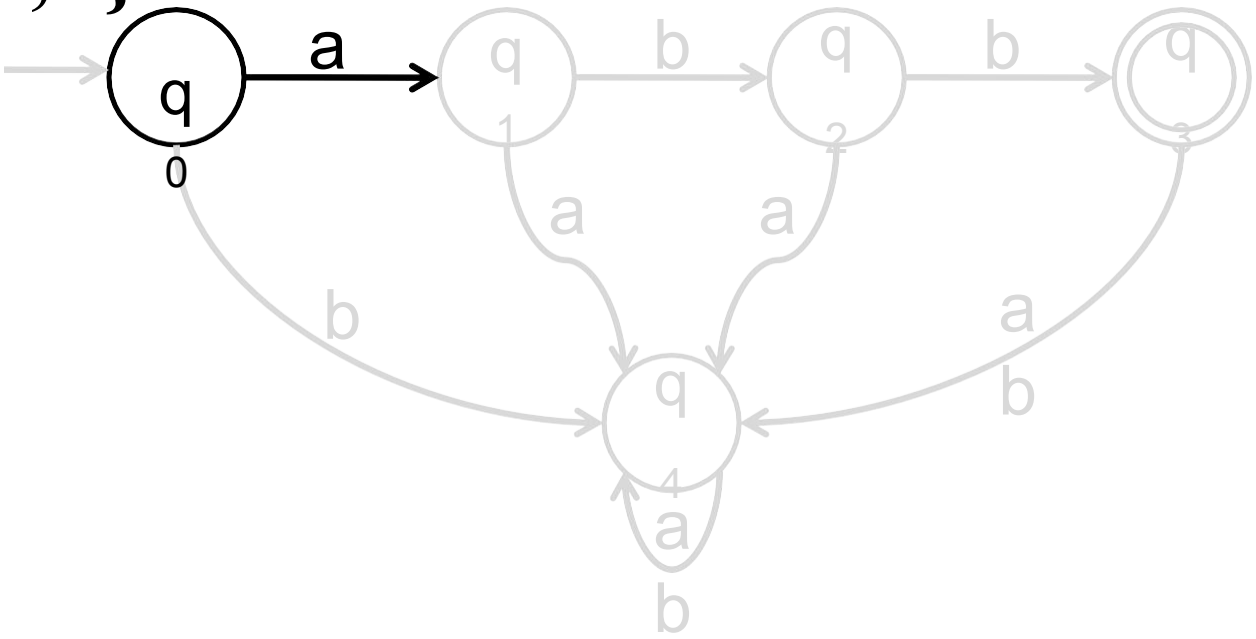
Σ = {a,b}



DFA parts

ALPHABET:

Σ = {a,b}

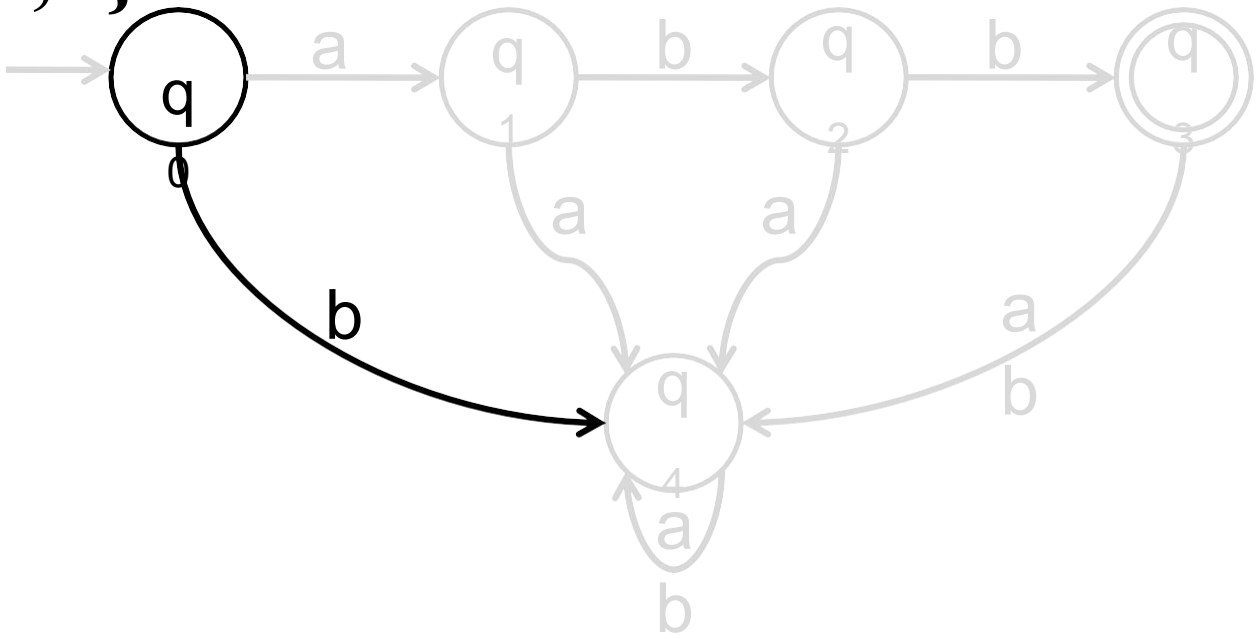


***Exactly one** transition from every state under each symbol in the alphabet.*

DFA parts

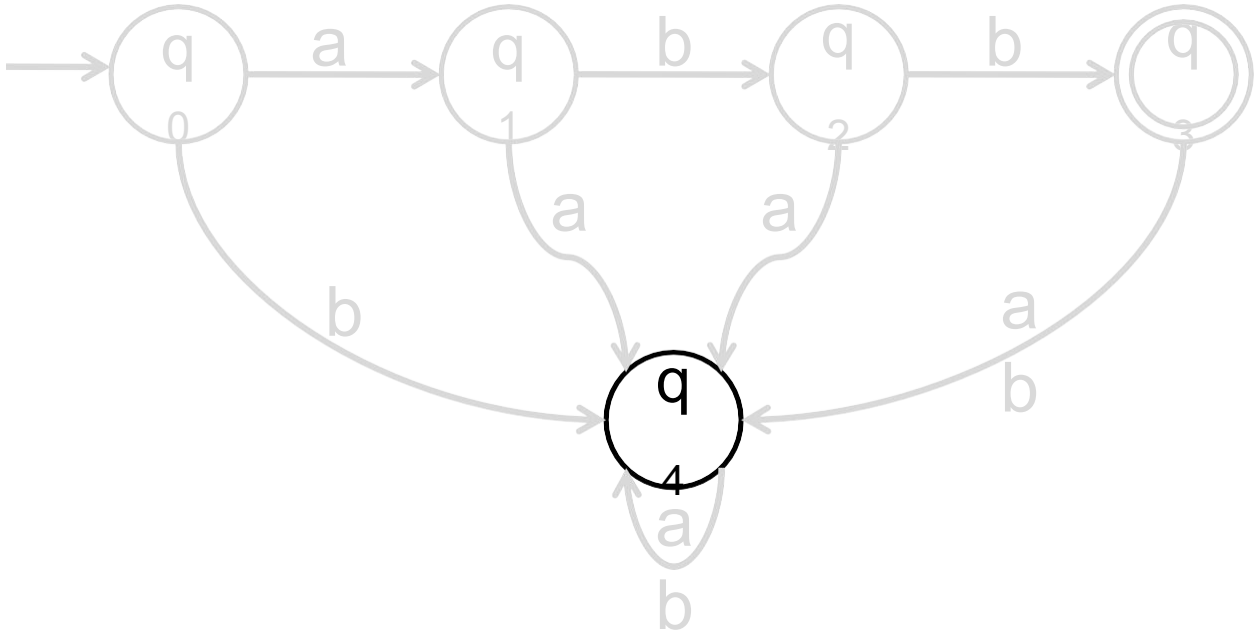
ALPHABET:

Σ = {a,b}

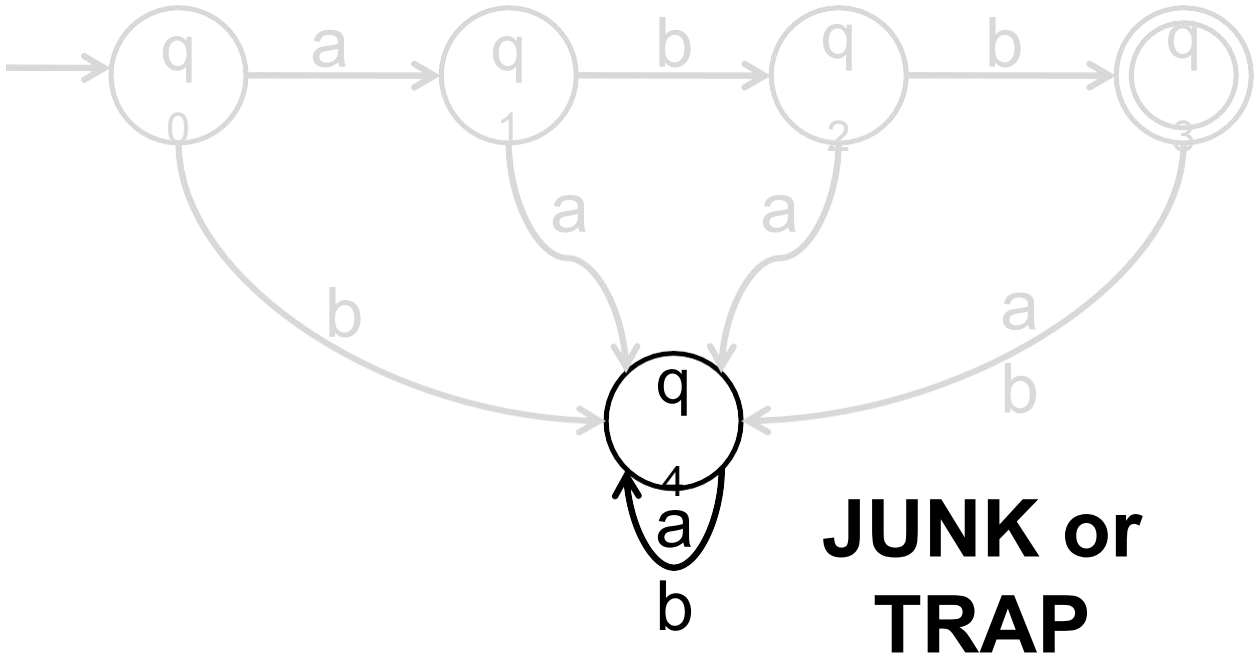


***Exactly one** transition from every state under each symbol in the alphabet.*

DFA parts

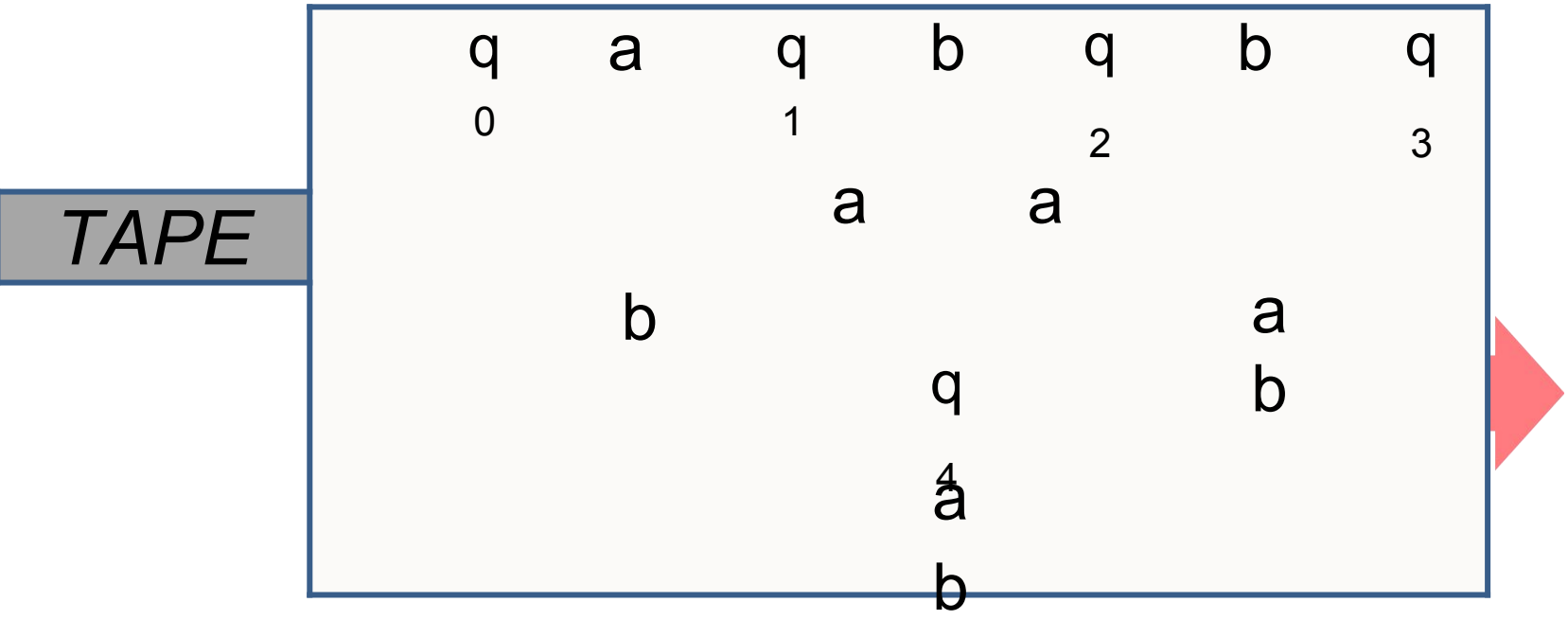


DFA transition graph



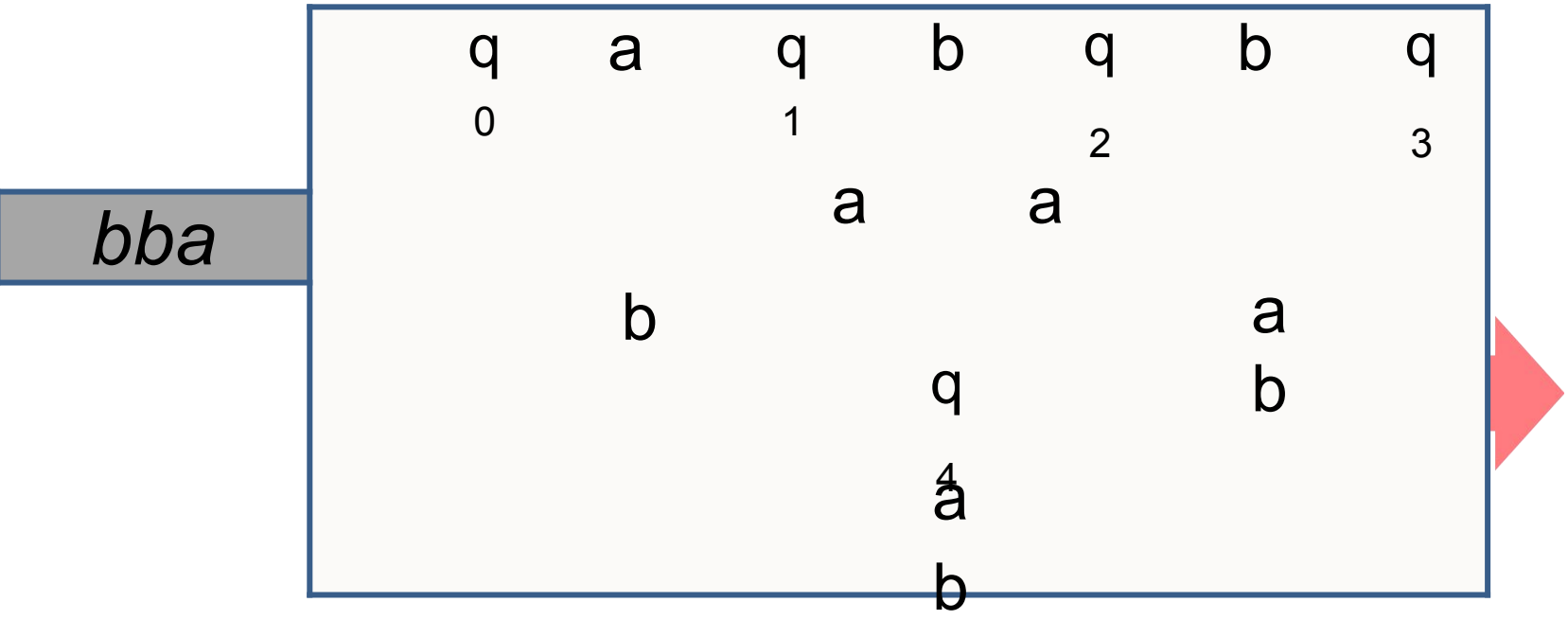
JUNK or TRAP STATE

DFA (cont…)



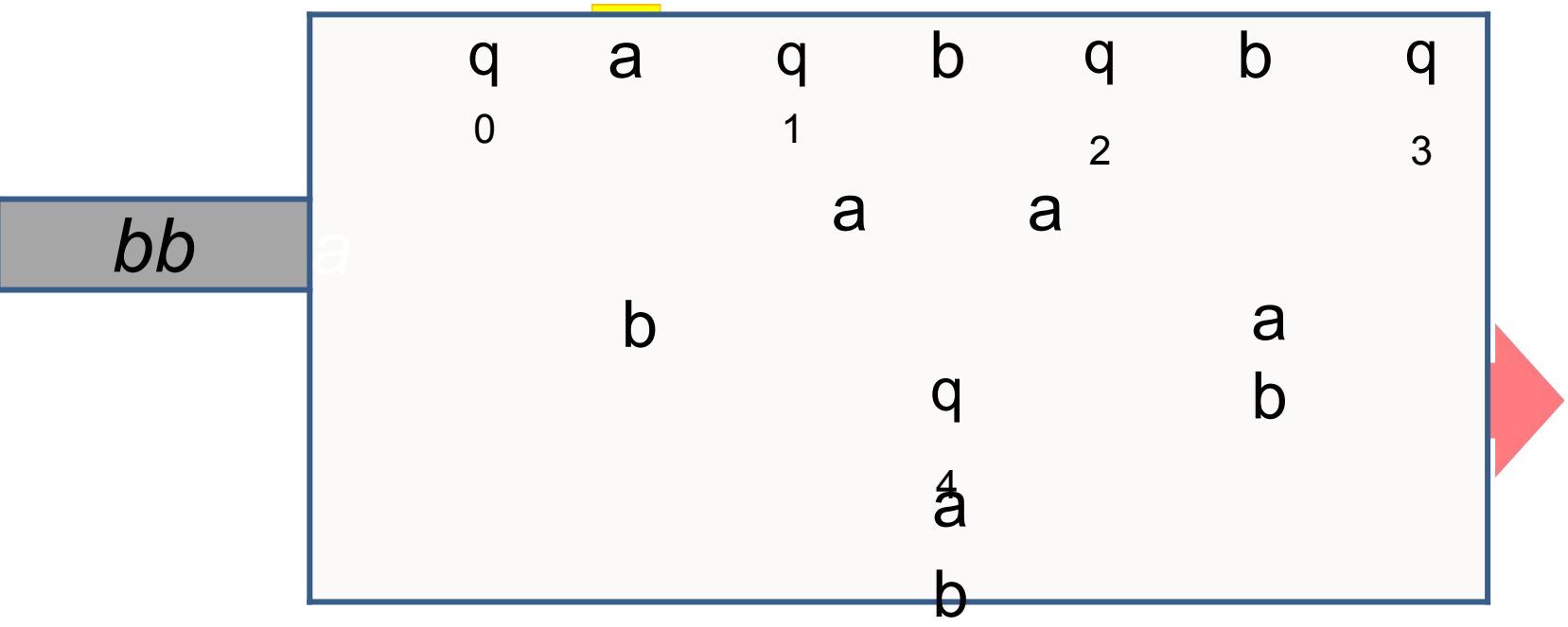
DFA (cont…)

Input string: *abb*



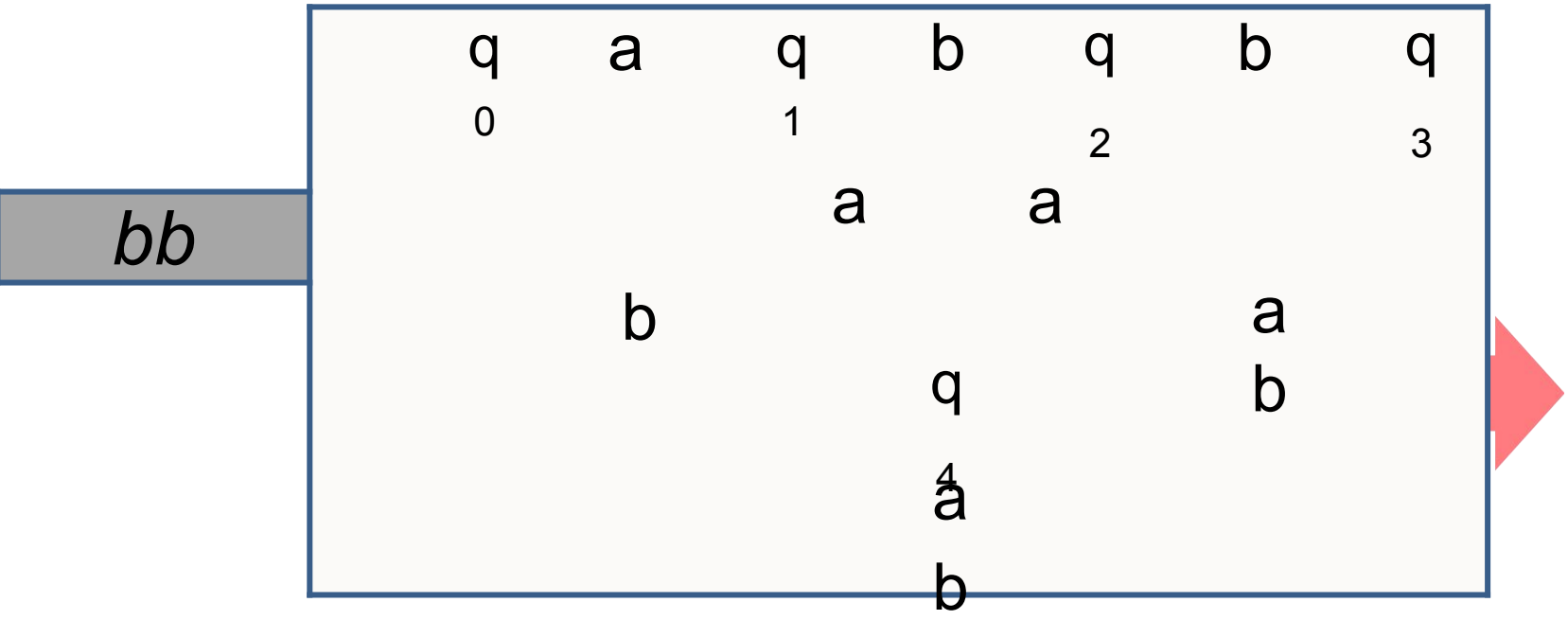
DFA (cont…)

Input string: *abb*



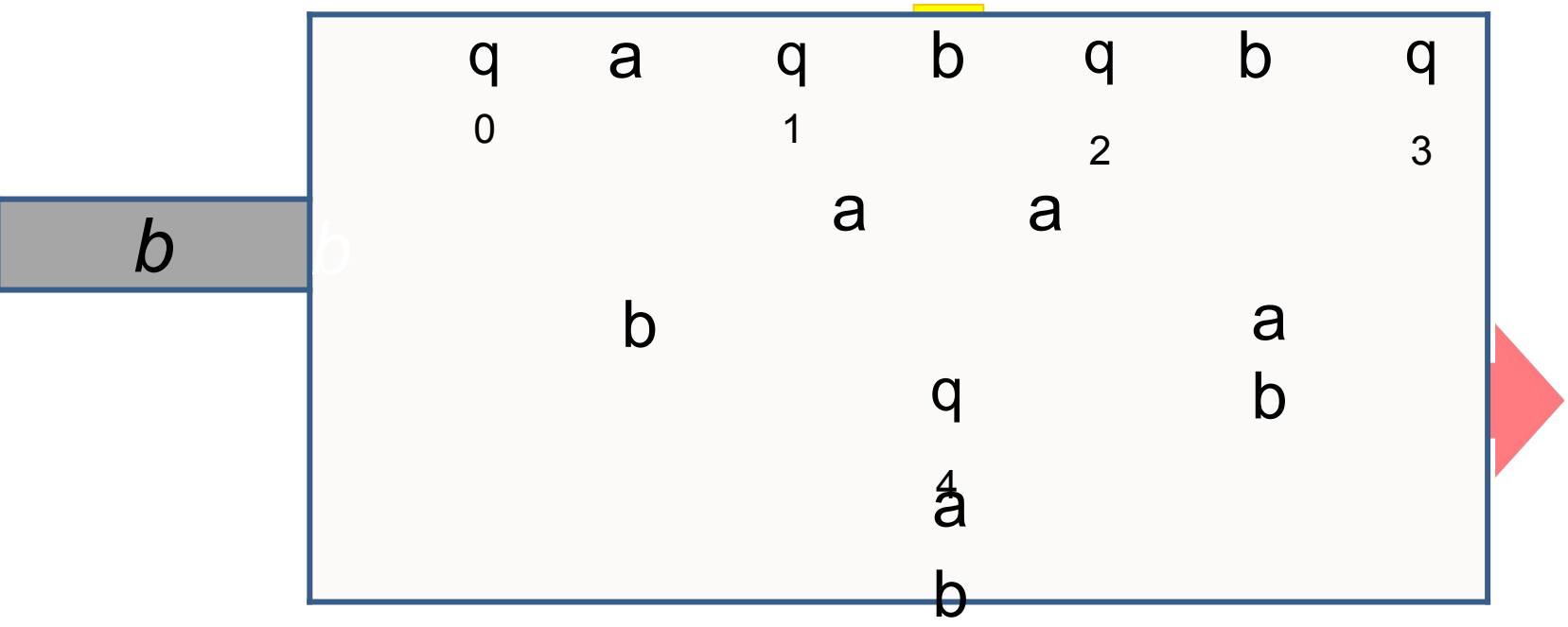
DFA (cont…)

Input string: *abb*



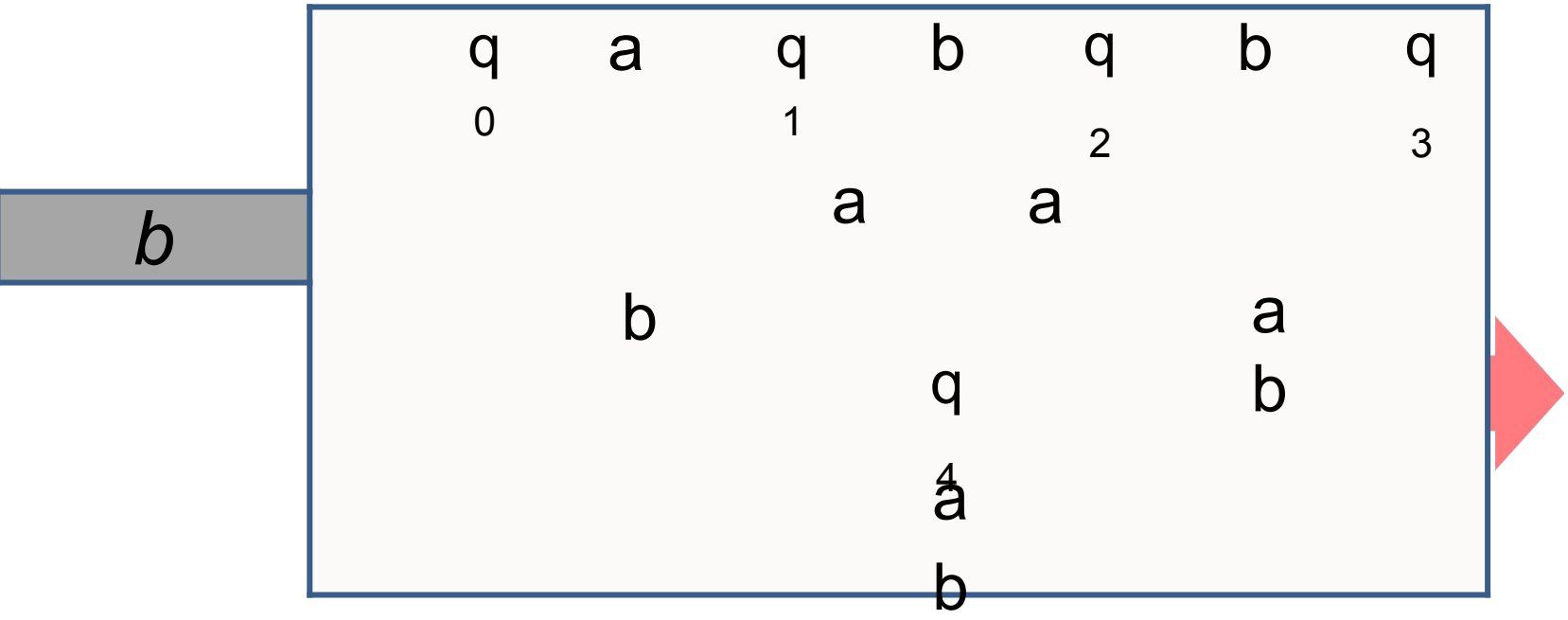
DFA (cont…)

Input string: *abb*



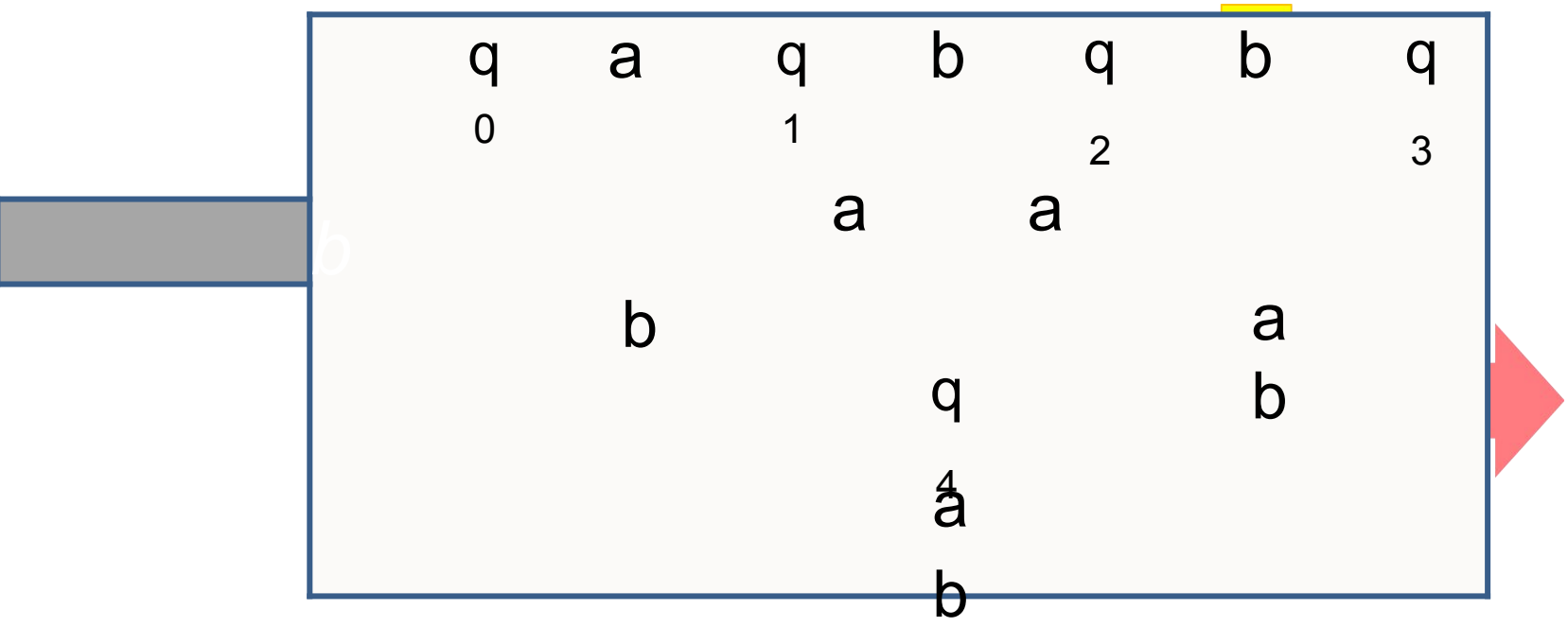
DFA (cont…)

Input string: *abb*



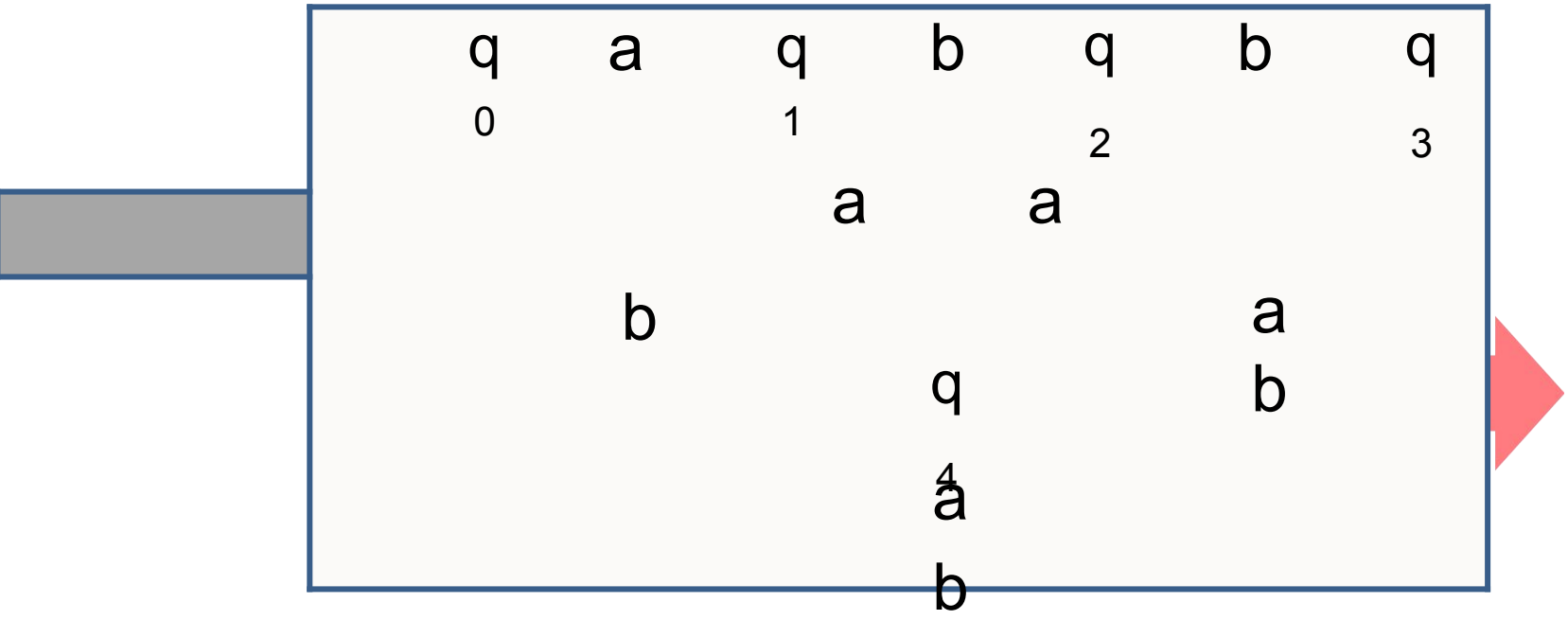
DFA (cont…)

Input string: *abb*



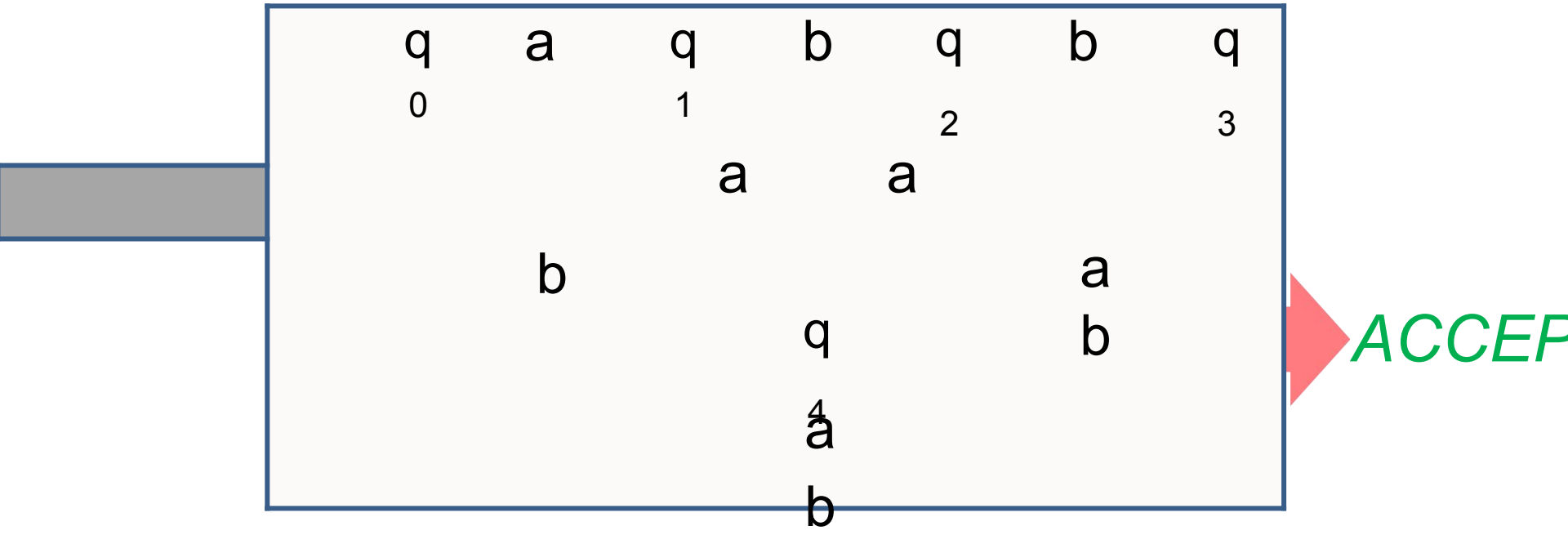
DFA (cont…)

Input string: *abb*



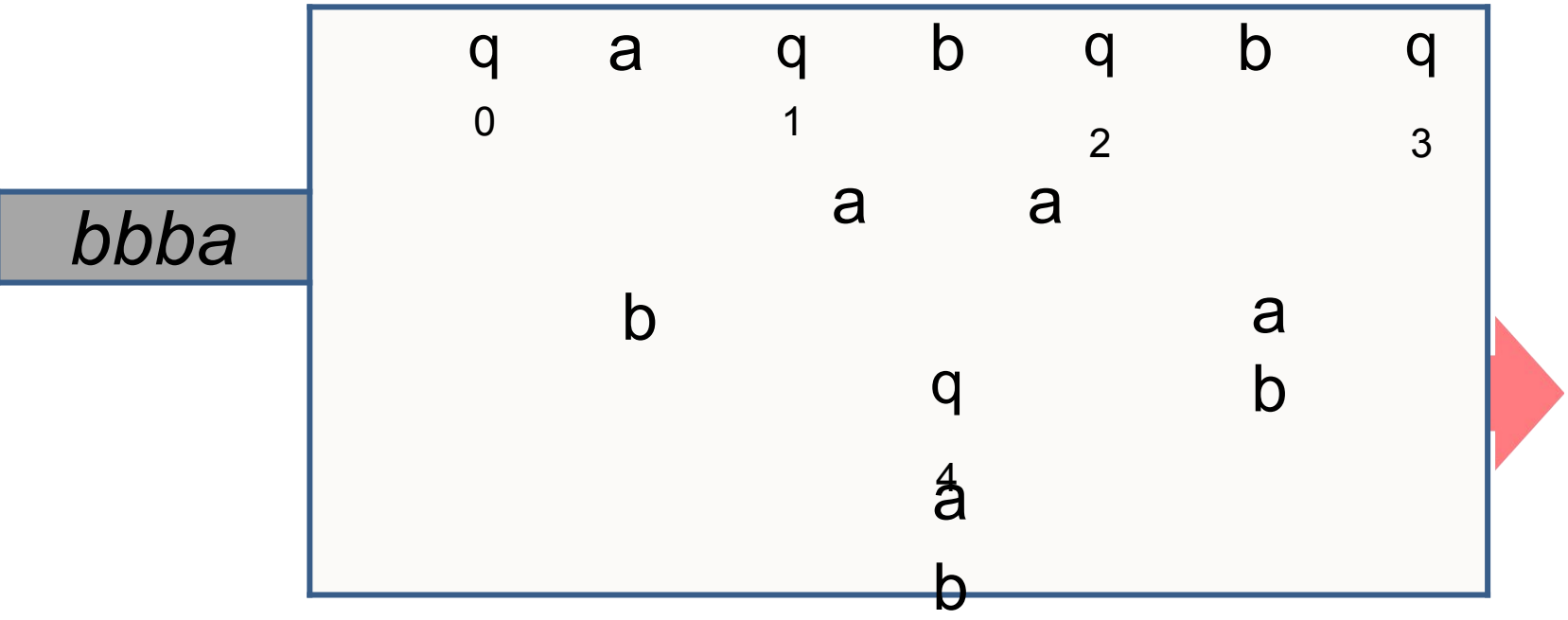
DFA (cont…)

Input string: *abb*



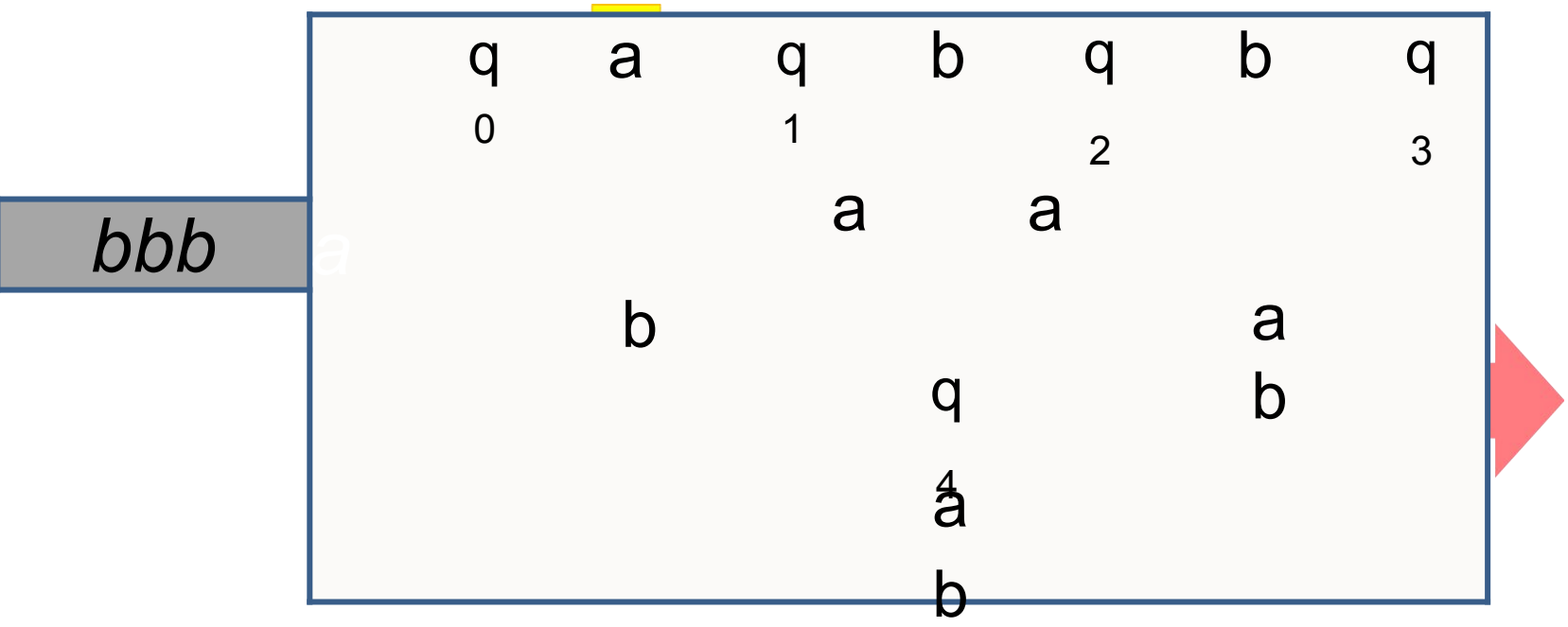
DFA (cont…)

Input string: *abbb*



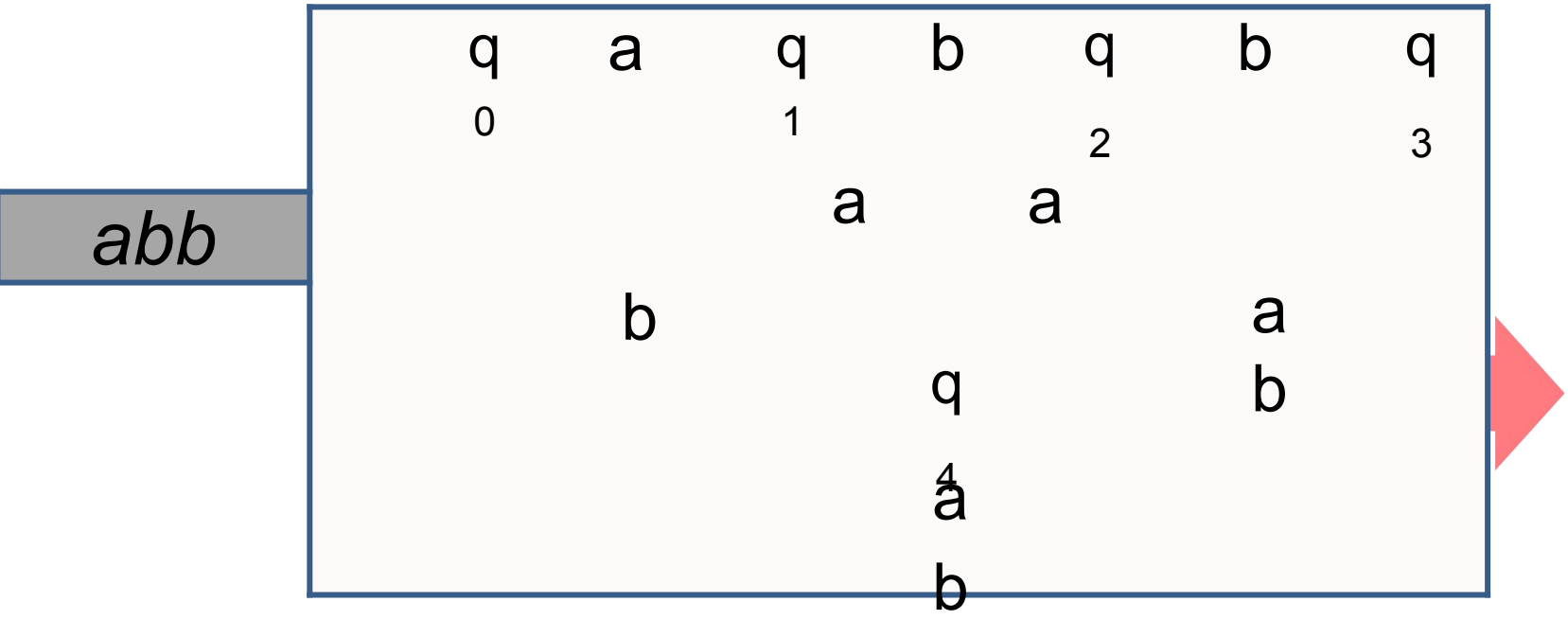
DFA (cont…)

Input string: *abbb*



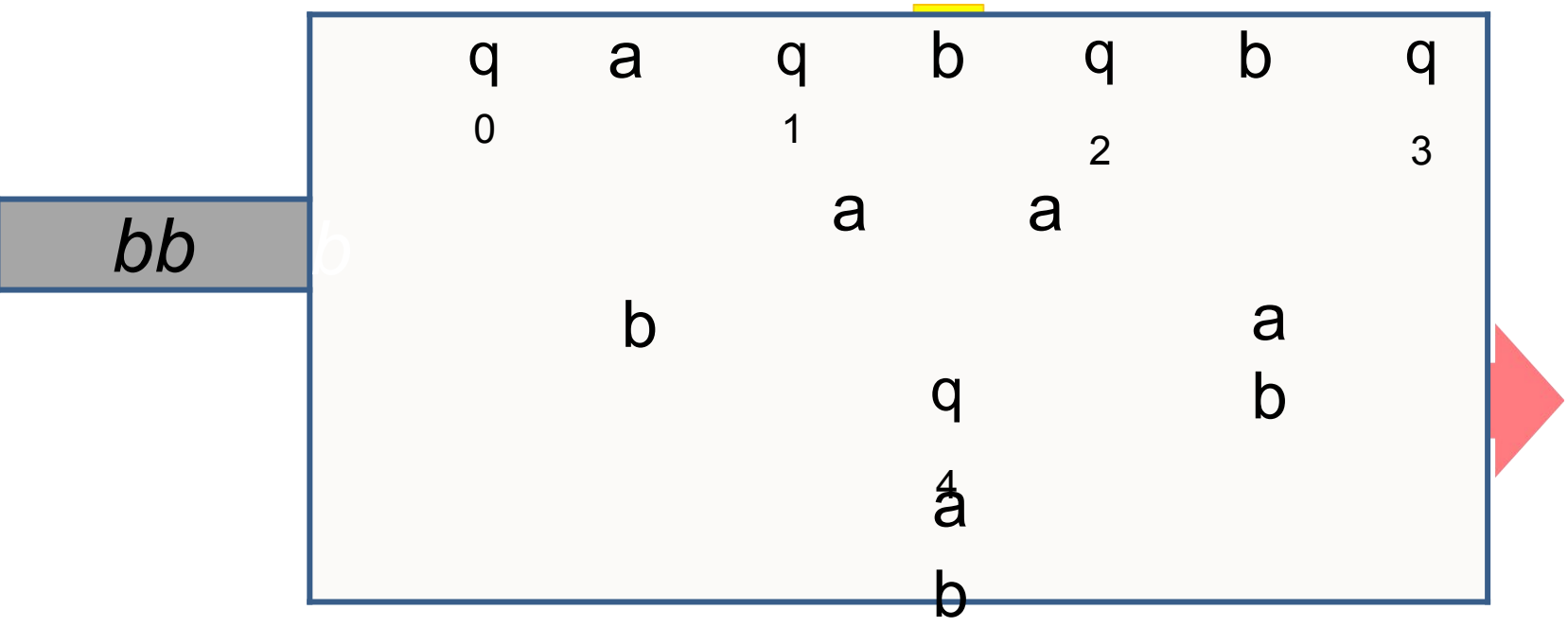
DFA (cont…)

Input string: *abbb*



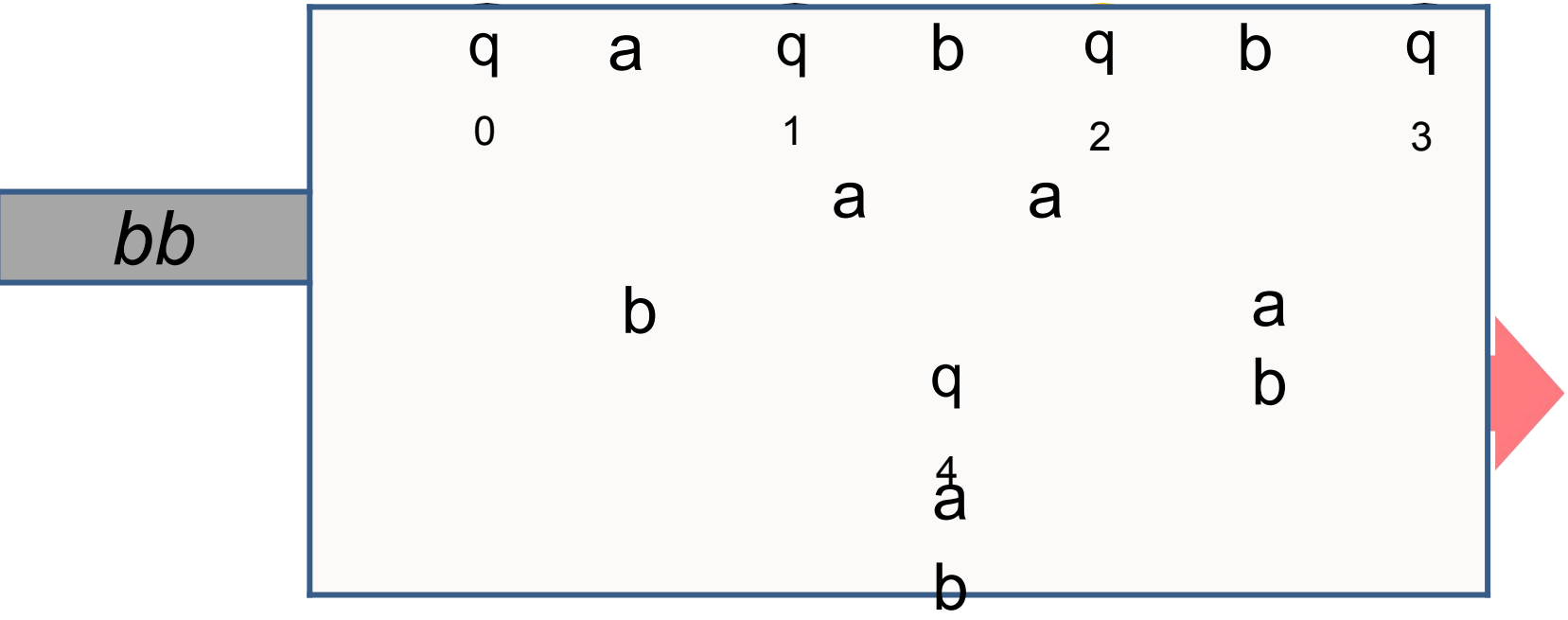
DFA (cont…)

Input string: *abbb*



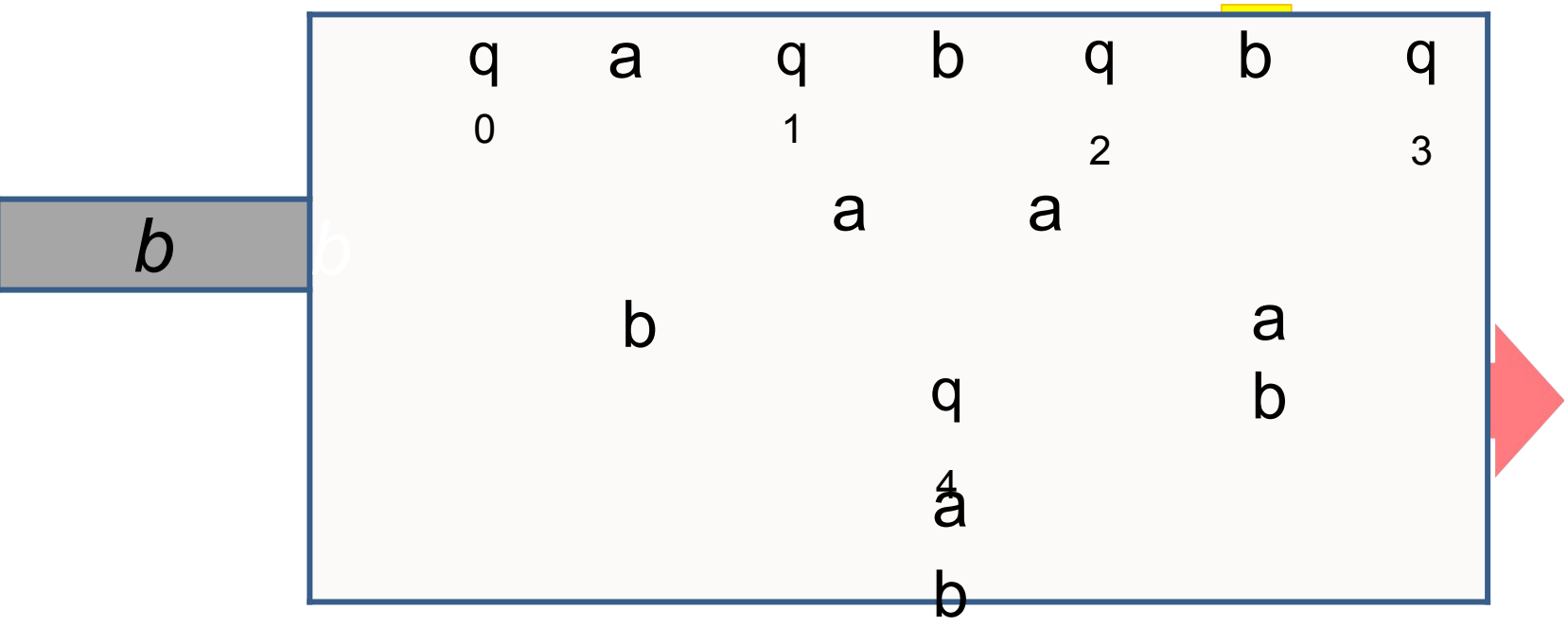
DFA (cont…)

Input string: *abbb*



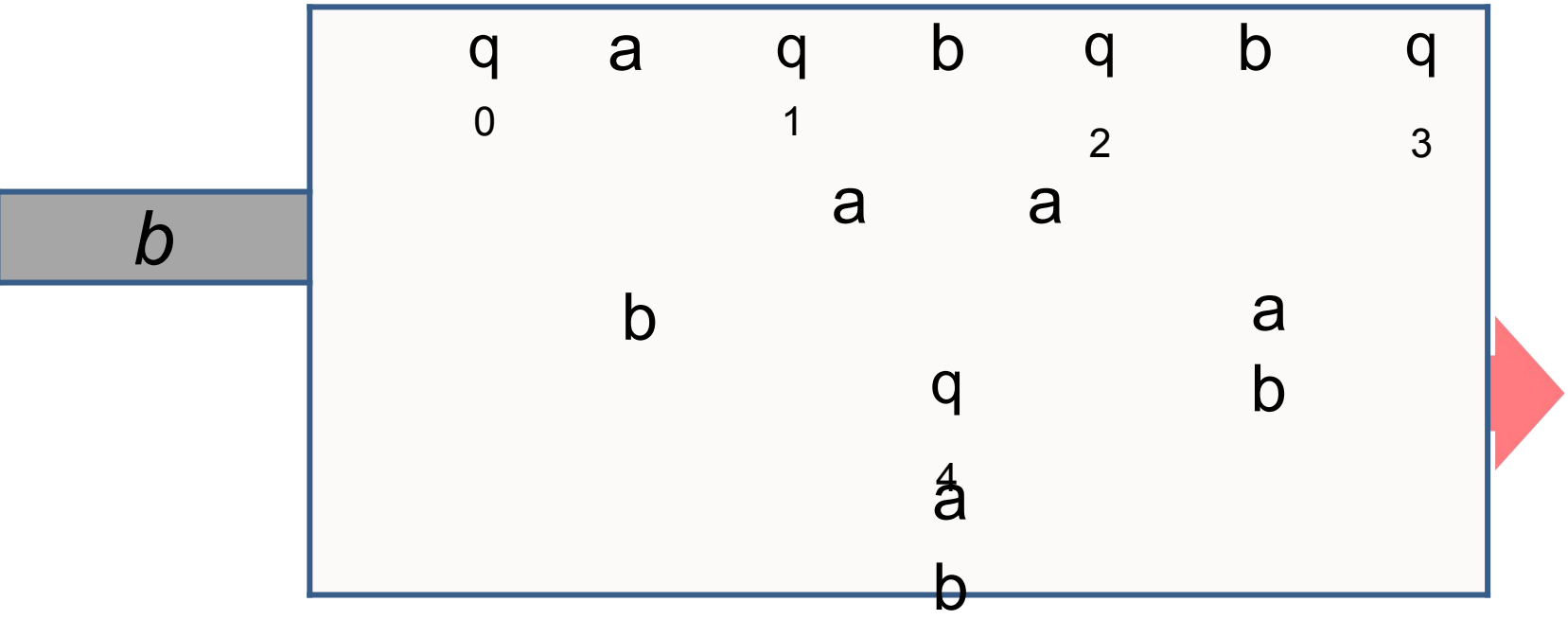
DFA (cont…)

Input string: *abbb*



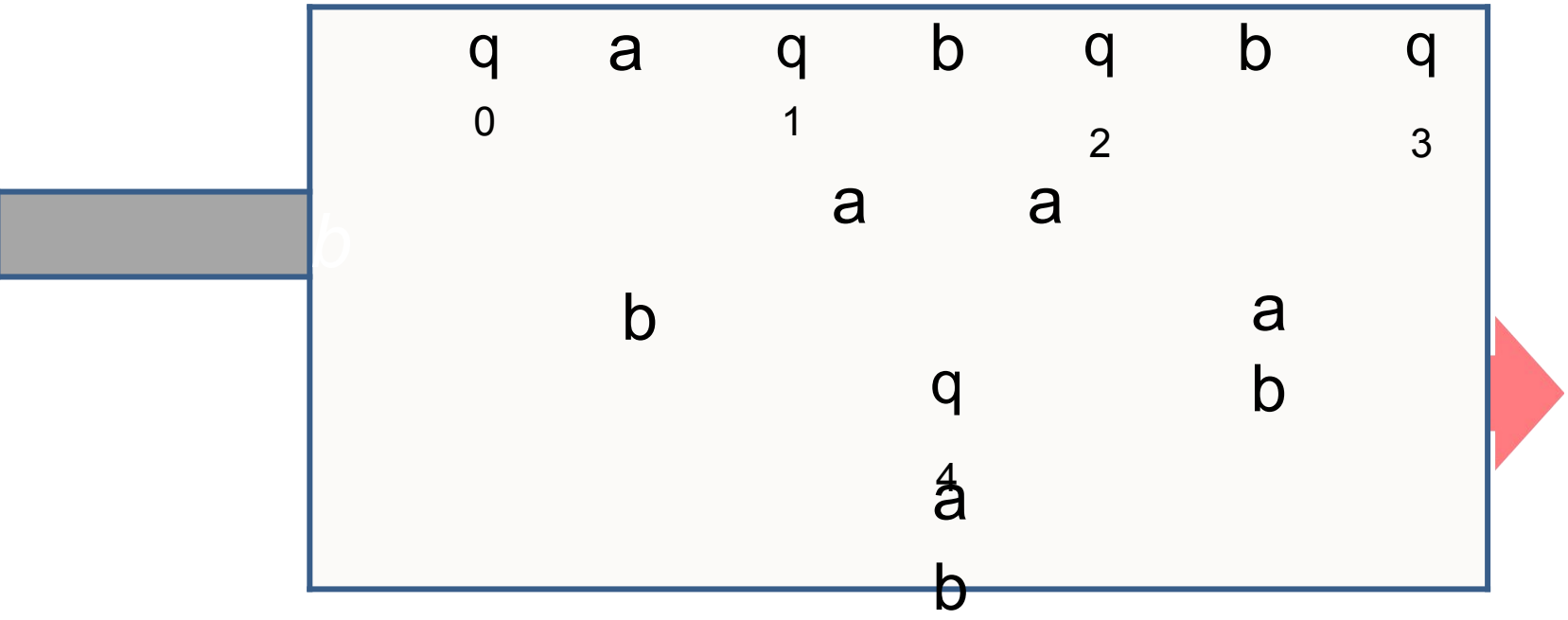
DFA (cont…)

Input string: *abbb*



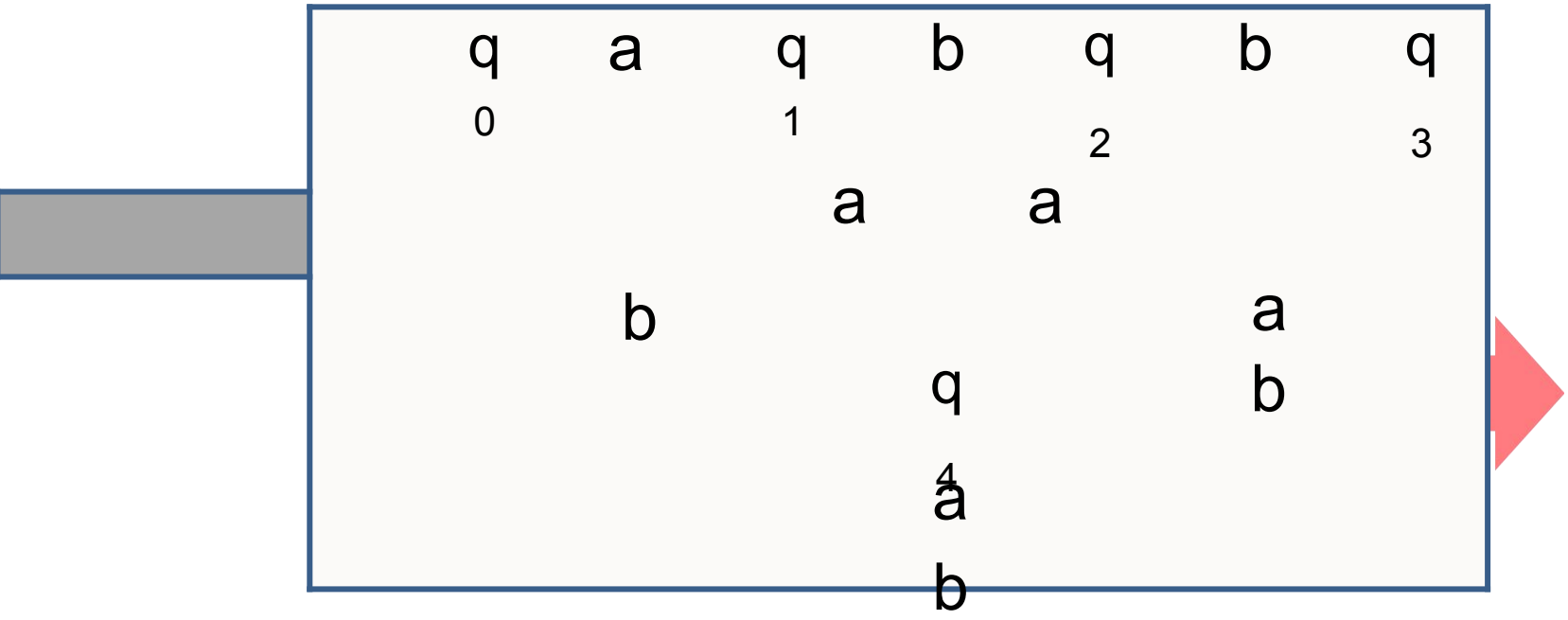
DFA (cont…)

Input string: *abbb*



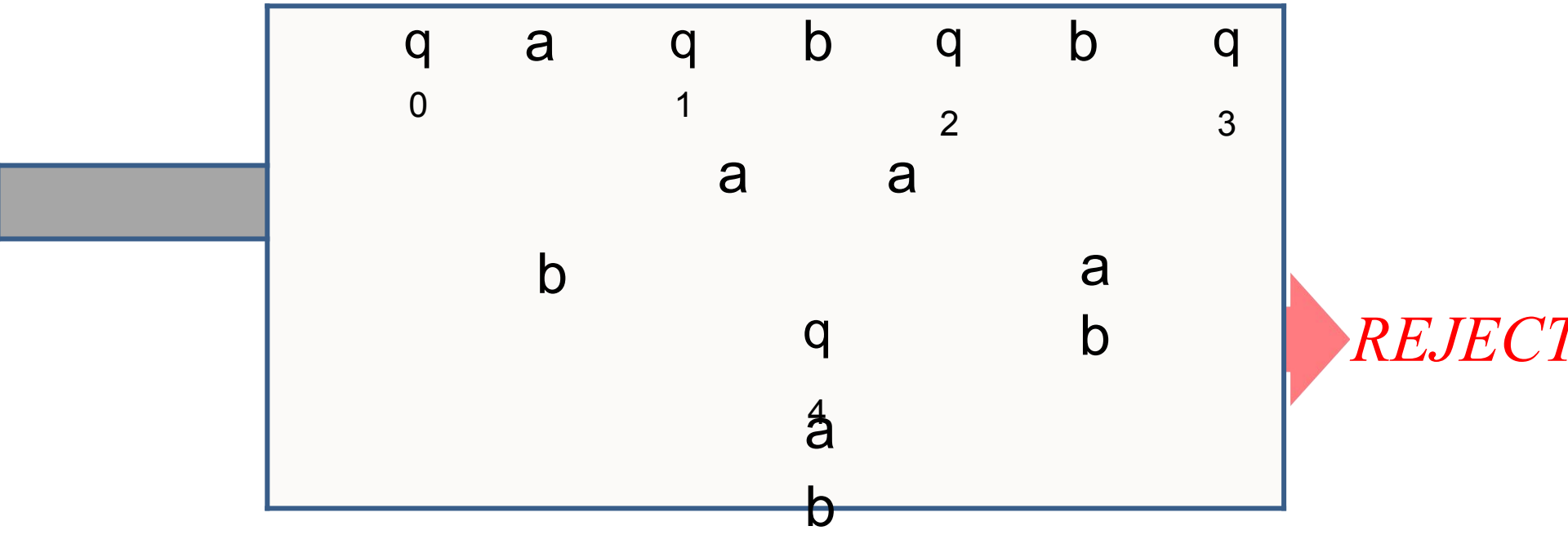
DFA (cont…)

Input string: *abbb*



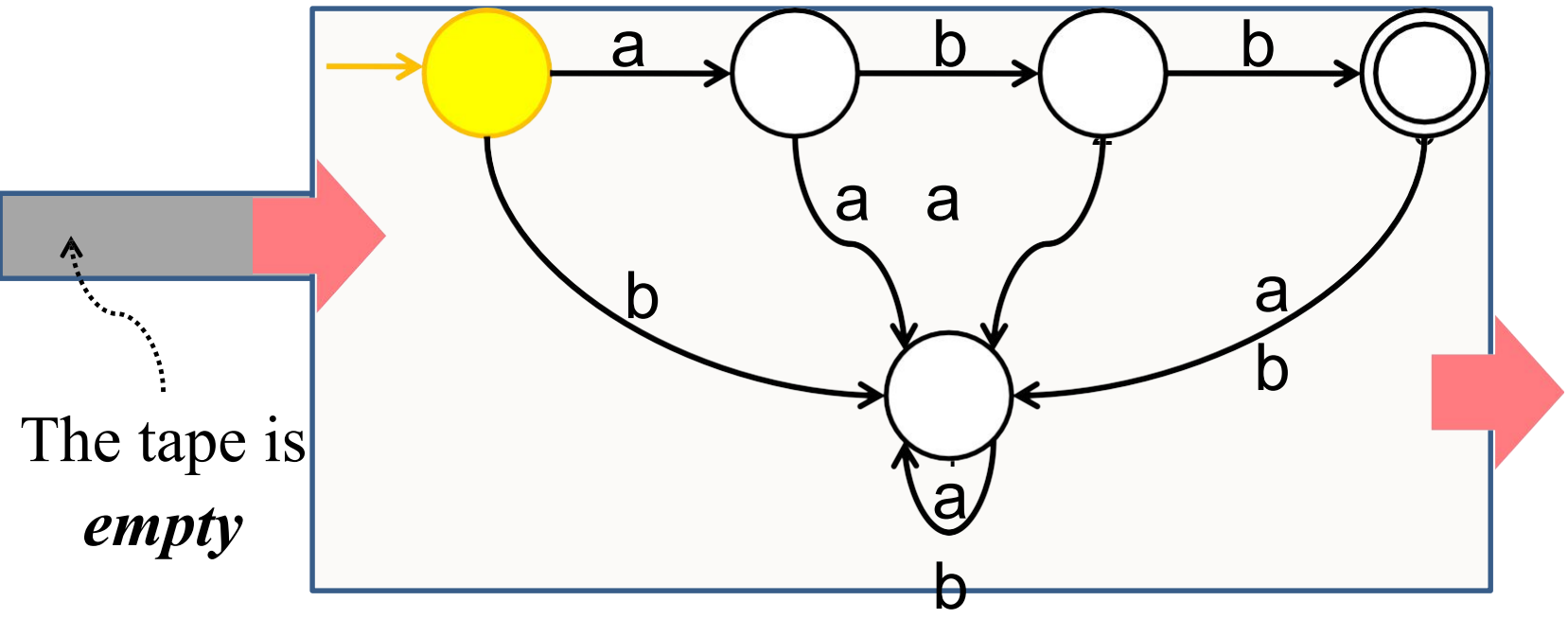
DFA (cont…)

Input string: *abbb*



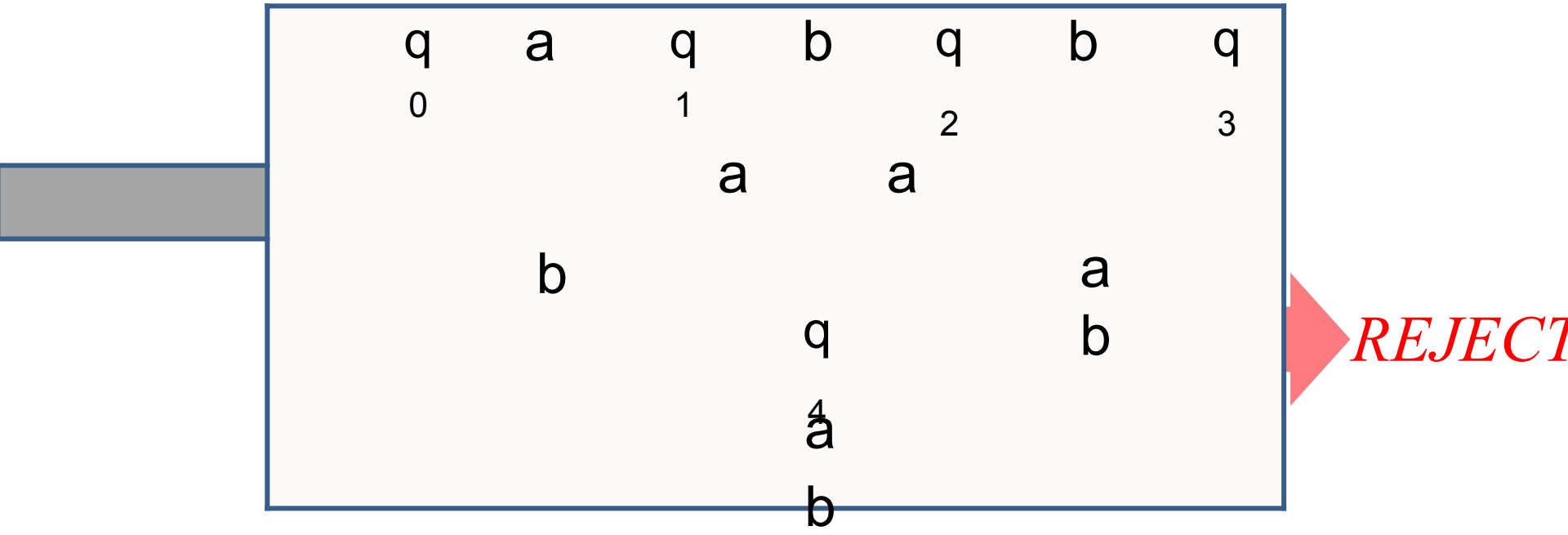
Empty string

Input string: ε

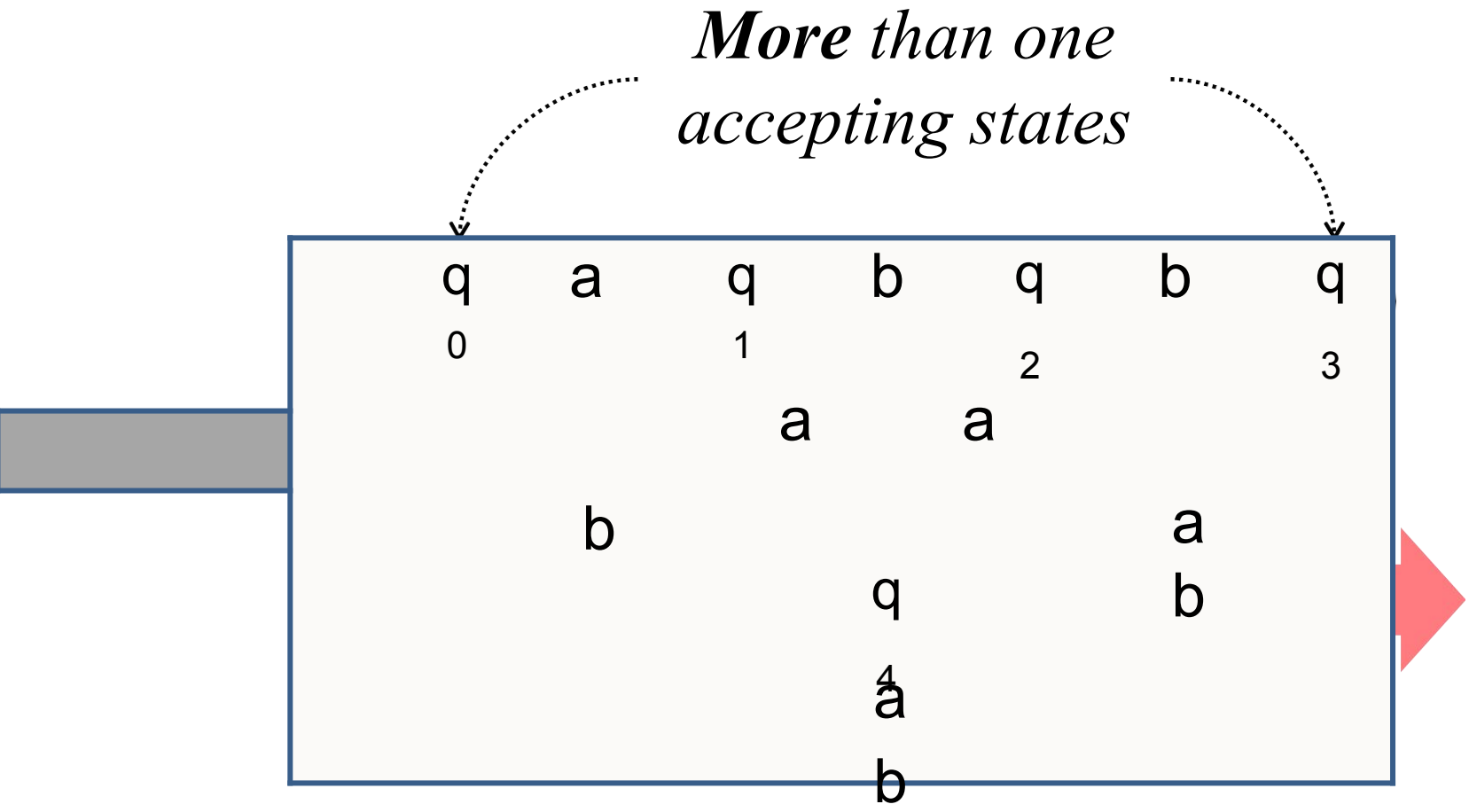


Empty string

Input string: ε

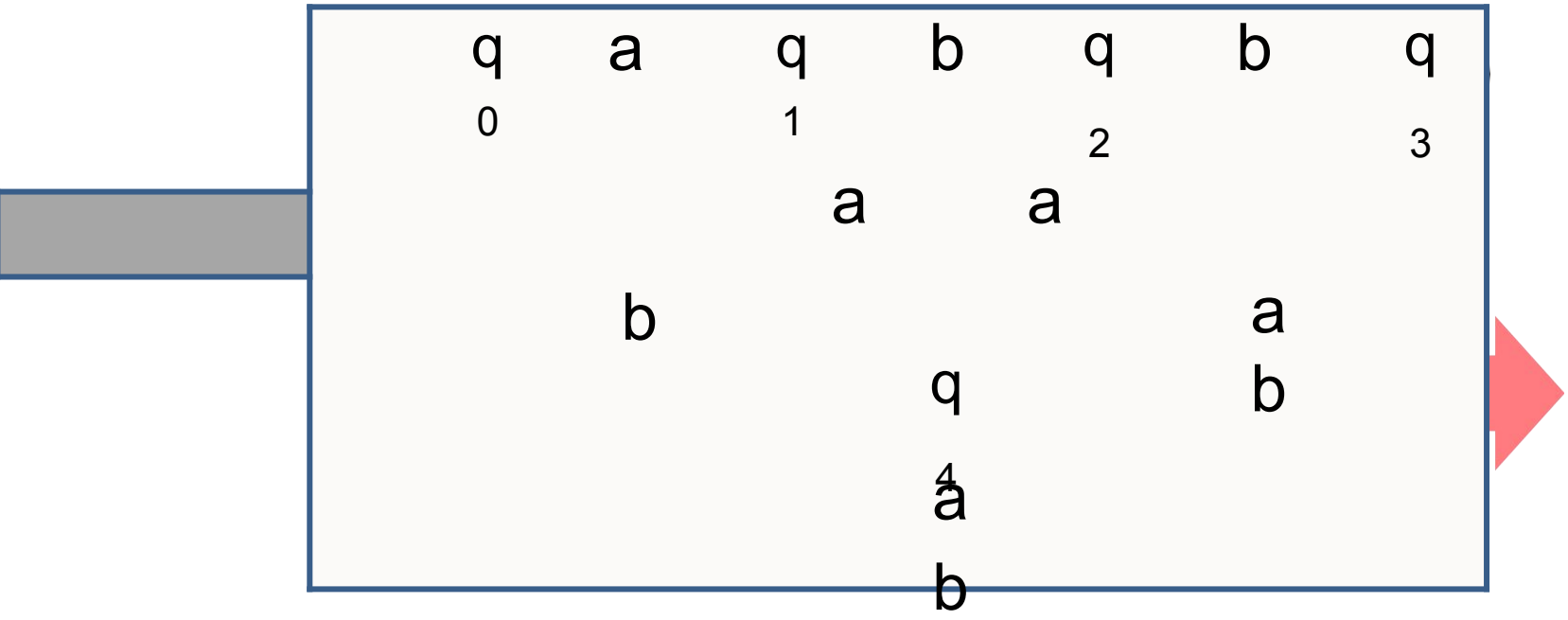


Other Example



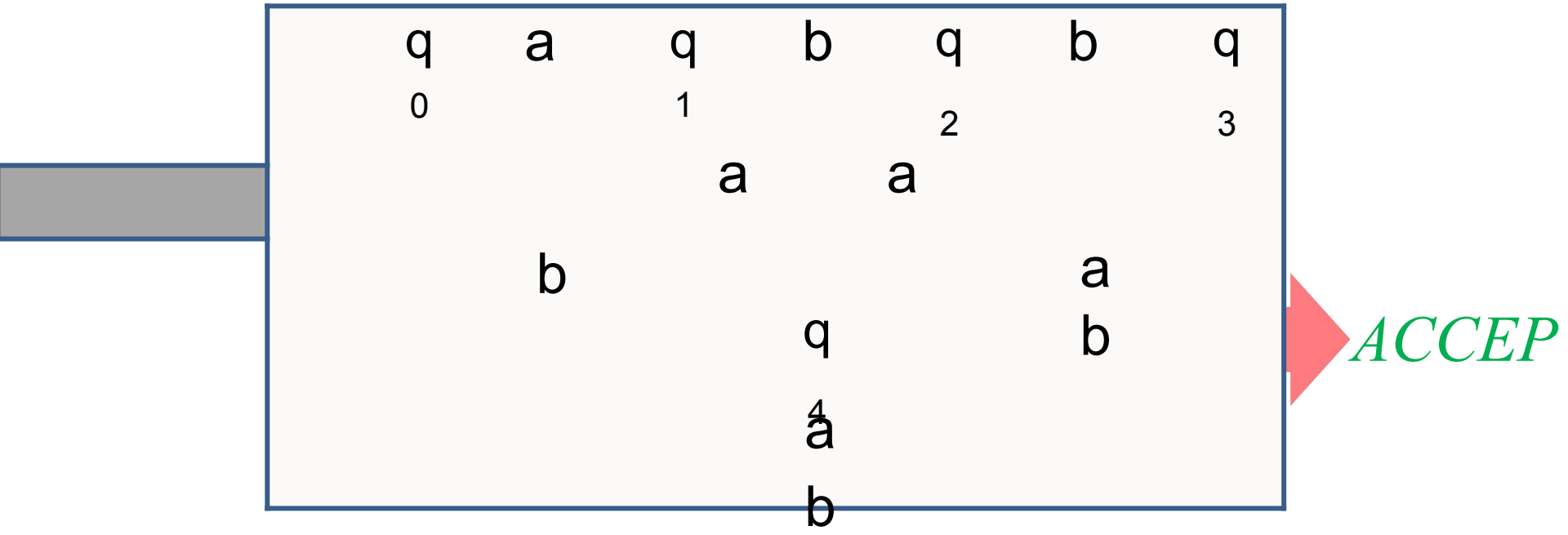
Other Example

Input string: ε



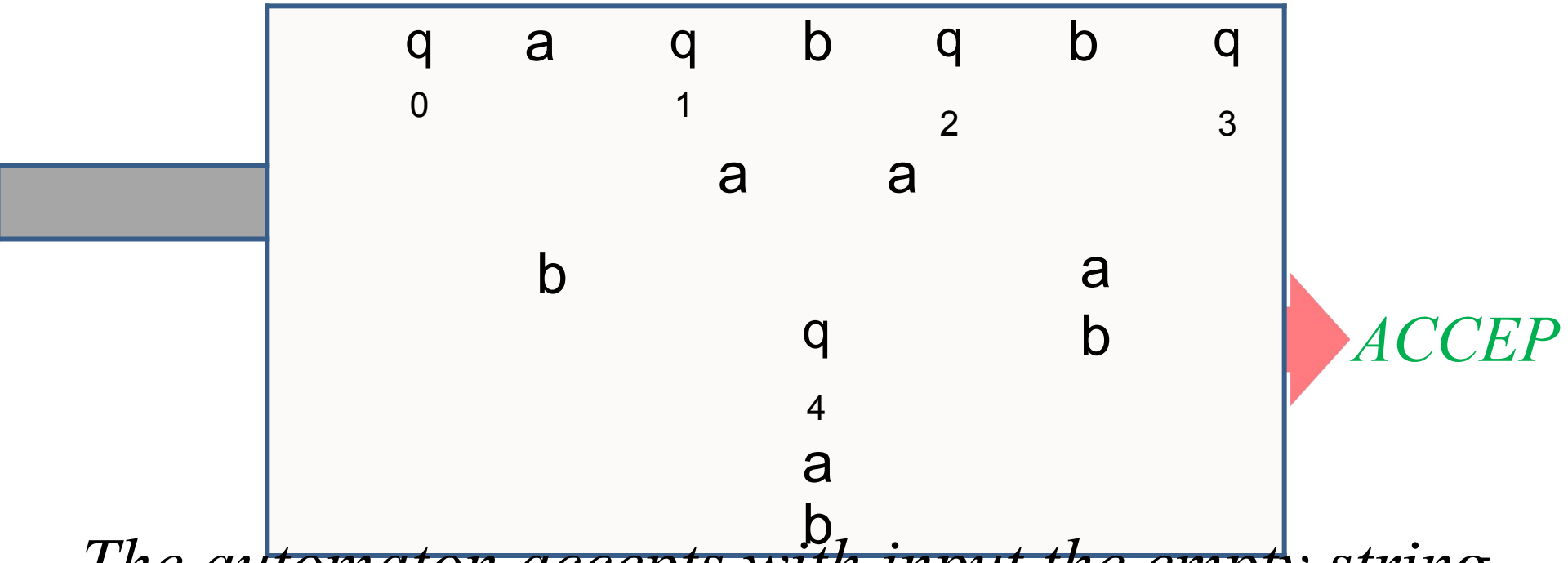
Other Example

Input string: ε



Other Example

Input string: ε

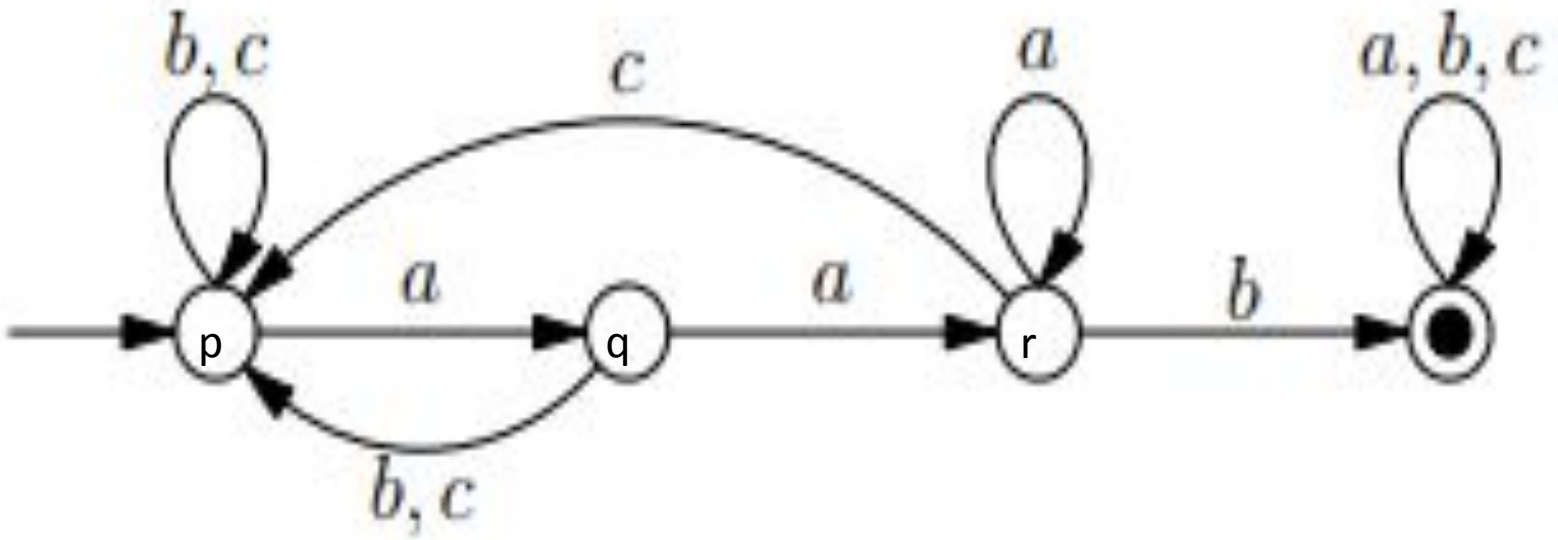


The automaton accepts with input the empty string ε *iff the START state is an ACCEPTING state.*

DFA Example

Give a DFA for Σ = {a, b, c} that accepts any string with aab as a substring.

Set of valid strings:-{aab,aaabb,aaaba…..}

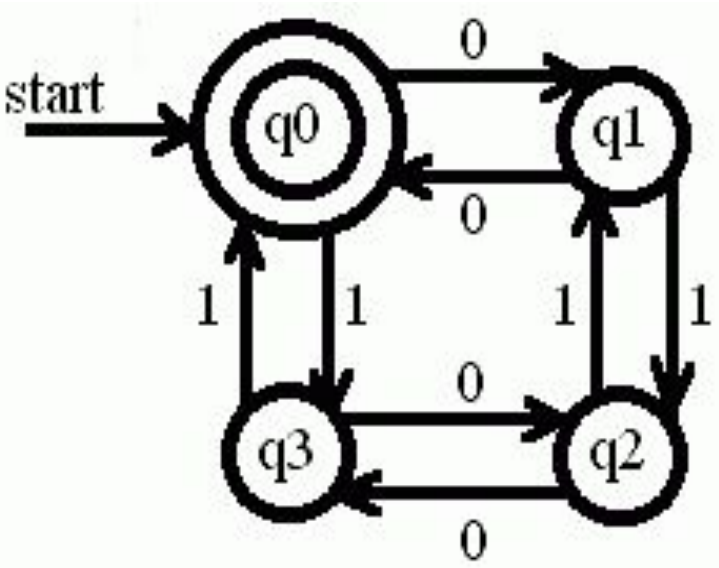


DFA Example

Construct a DFA to accept all strings containing even number of zeros and even number of ones.

Set of valid strings:-{ε,11,00,1010,00011011…….}

q_0 : even 0 even 1
q_1 : odd 0 even 1
q_2 : odd 0 odd 1
q_3 : even 0 odd 1



Non-Deterministic Finite Automata

- In NFA, it is possible to have more than one transition on reading the same input symbol from a given state

Non-Deterministic Finite Automata

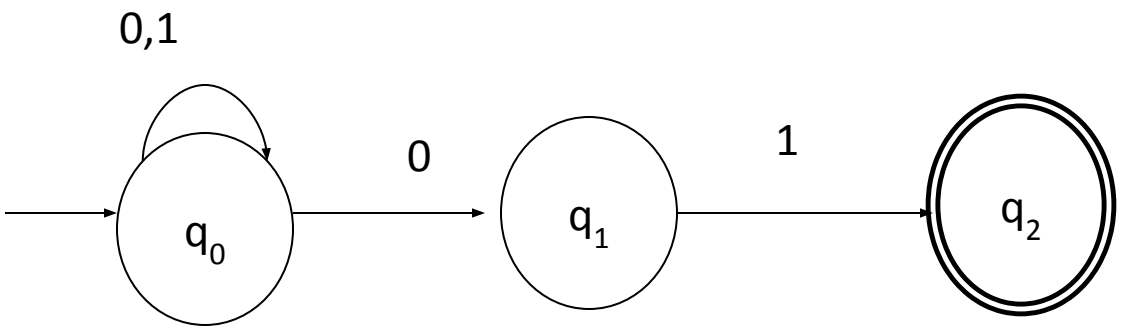
(cont…)

- 5-tuple:
- M= (Q,Σ, δ, q_o,

  F) where,
  - Q: Finite set of states
  - Σ: Finite input alphabet
  - **δ: STF that maps Q X Σ to Q, i.e. Q X $\Sigma \longrightarrow 2^Q$**
  - q_o: Initial state of FA, $q_o \in Q$
  - F: Set of final states, $F \subseteq Q$

NFA Example

NFA that recognizes the language of strings that end in 01

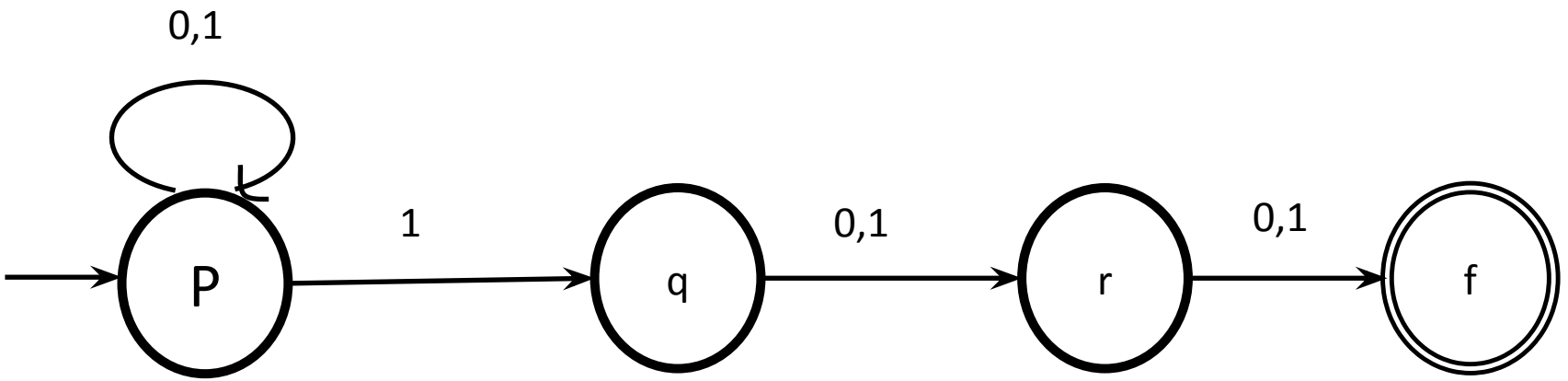


note: $\delta(q_0,0) = \{q_0,q_1\}$

$\delta(q_1,0) = \{\}$

NFA Example

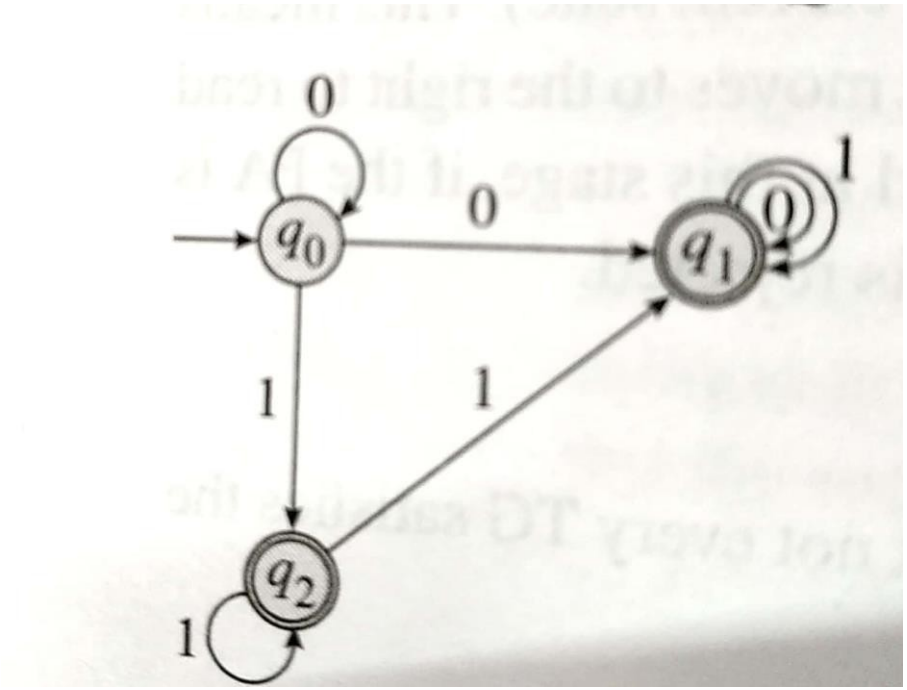
All strings such that the third symbol from the right end is a 1
Valid strings={111,101,110,1100,0100.....}



NFA Example

- **Example:**

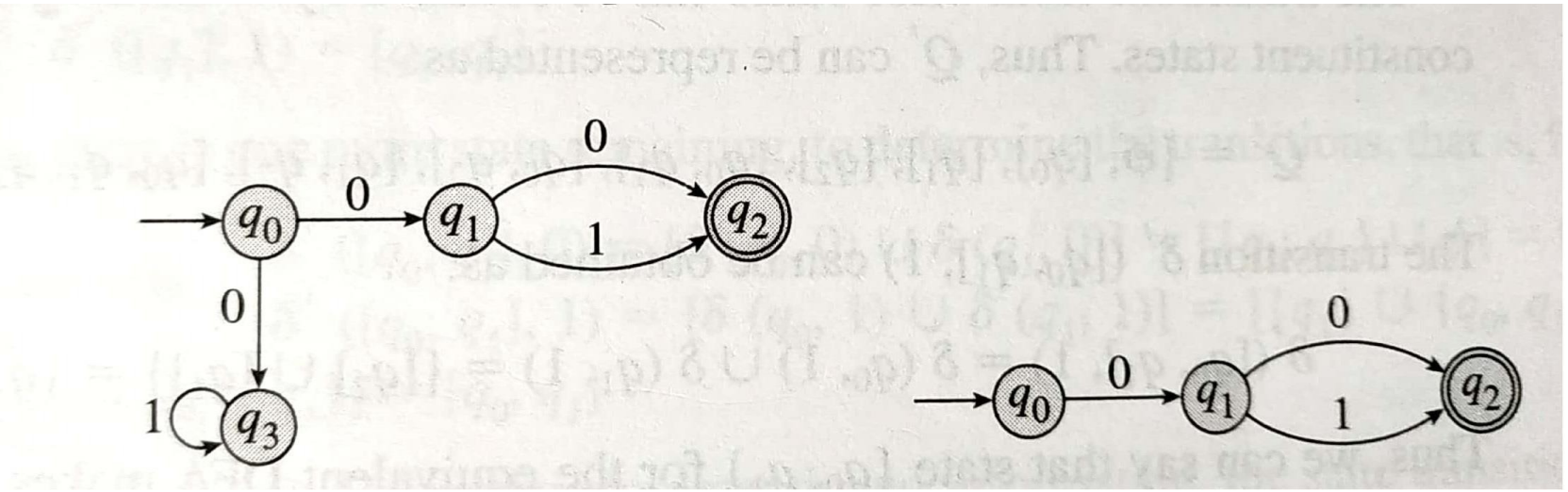
Transition Graph for example NFA



Equivalence of NFA and DFA

- **Example:**

NFA vs DFA



Example NFA

Equivalent DFA

NFA and DFA have equal powers

Conversion of NFA to DFA
(Method I)

- For every NFA, there is an equivalent DFA
- **Let us consider Q'= 2^Q, set of states for resulting DFA**

  Q' X $\Sigma \rightarrow$ Q'
- **Find all transitions from every state in Q' on reading each input symbols**
- Every combination of states can be considered as a new state in the resulting DFA and can be given a new label

Conversion of NFA to DFA

Method I (cont…)

- In the resulting DFA, there should be a unique state resulting from an input to the previous state
- The initial state remains unchanged for the resultant DFA
- **All possible subsets of Q (i.e. 2^Q) are the possible states for the resultant DFA**
- Final state in resulting DFA is all the states containing final state of given NFA

Conversion of NFA to DFA

Method I (cont…)

Example:

Convert the NFA : $M = (\{q_0, q_1\}, \{0, 1\}, \delta, q_0, \{q_1\})$ to its equivalent DFA

State Transition Table δ for example NFA

Q \ Σ	0	1
q_0	$\{q_0, q_1\}$	$\{q_1\}$
q_1	ϕ	$\{q_0, q_1\}$

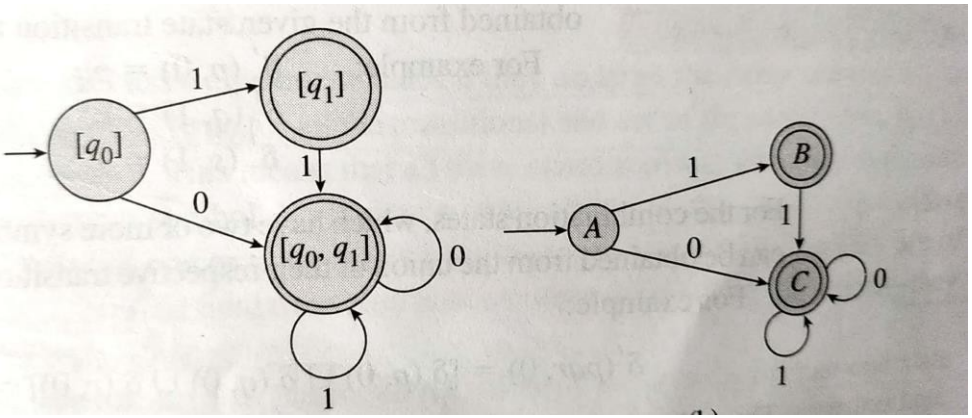
Conversion of NFA to DFA

Method I (cont…)

State Transition table for resultant NFA

Q' \ Σ	0	1
$[q_0]$	$[q_0, q_1]$	$[q_1]$
$[q_1]$	–	$[q_0, q_1]$
$[q_0, q_1]$	$[q_0, q_1]$	$[q_0, q_1]$

State Transition Graph for resultant NFA



Conversion of NFA to DFA

Method II

- Easier and direct
- Takes lesser number of steps and time to build an equivalent DFA
- Instead of considering the set $Q'=2^Q$ and then removing non-required states, consider only those states that are required

Conversion of NFA to DFA

Method II (cont…)

- Start building an equivalent DFA with the transition diagram of given NFA
- Find the transition, one state at a time
- If the next state of a given transition is a new combination, add that to the set of states for the resultant DFA

Conversion of NFA to DFA

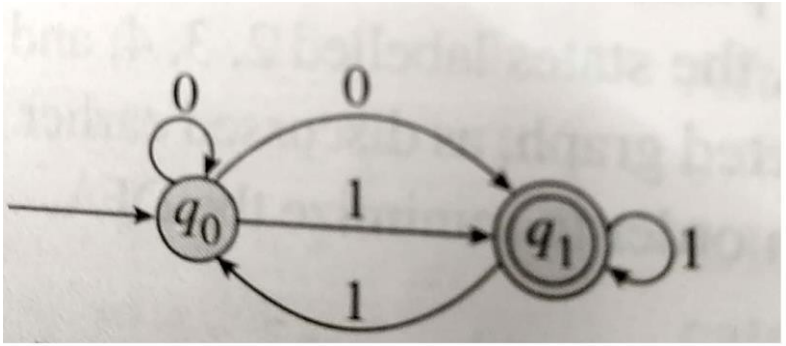
Method II (cont…)

- **Example:**
  - Convert the NFA: M = ($\{q_0, q_1\}$, {0, 1}, δ, q_0, $\{q_1\}$) to its equivalent DFA

State Transition Table δ for example NFA

Q \ Σ	0	1
q_0	$\{q_0, q_1\}$	$\{q_1\}$
q_1	ϕ	$\{q_0, q_1\}$

Transition Graph for example NFA

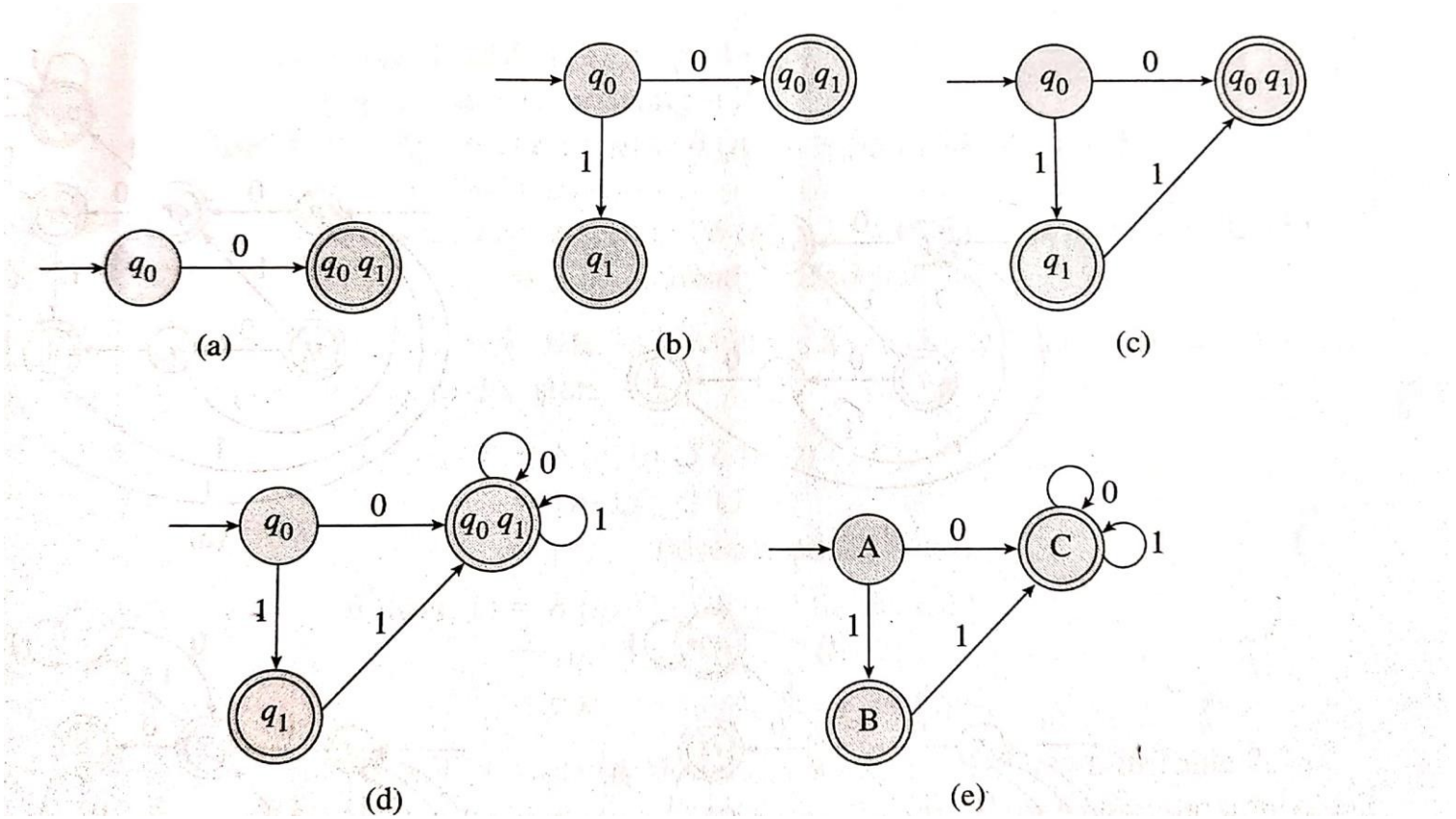


Conversion of NFA to DFA

Method II (cont…)

DFA construction from the given NFA:

(a) Step 1 (b) Step 2 (c) Step 3 (d) Step 4 (e)Final DFA



DFA Minimization

- Identify equivalent states and keep any one of them.
- Replace remaining states by that one.
- Identify unreachable states and remove them
- Identify dead (or trap) states and remove them

DFA Minimization (cont..)

- Equivalent states:
  - go to same next state on reading the same input symbol
  - same type (final or non-final)
- Unreachable states:
  - Can not be reached from initial state on reading any input symbol
- Dead (or Trap) states:
  - Have no outgoing transitions, except to themselves

DFA Minimization Rules

- One non-final state can be replaced by its equivalent non-final state only
- One final state can be replaced by its equivalent final state only
- Initial state can not be replaced by any other state
- One final state can not be replaced by a non-final state or one non-final state can not be replaced by a final state

DFA Minimization Rules (cont…)

- Replacing state A by state B means deleting all entries related to state A i.e. deleting all the transitions for state A from the state transition table.
- If more equivalent states are found after applying all the rules, repeat the same five steps to further reduce the DFA.

DFA Minimization (cont…)

State Transition Table of the DFA to be minimized

Q′ \ Σ		0	1
(1)	*p*	*pq* (5)	*p* (1)
(2)	*q*	*r* (3)	*r* (3)
(3)	*r*	*s* (4)	–
*(4)	*s*	*s* (4)	*s* (4)
(5)	*pq*	*pqr* (11)	*pr* (6)
(6)	*pr*	*pqs* (12)	*p* (1)
*(7)	*ps*	*pqs* (12)	*ps* (7)
(8)	*qr*	*rs* (10)	*r* (3)
*(9)	*qs*	*rs* (10)	*rs* (10)
*(10)	*rs*	*s* (4)	*s* (4)
(11)	*pqr*	*pqrs* (15)	*pr* (6)
*(12)	*pqs*	*pqrs* (15)	*prs* (13)
*(13)	*prs*	*pqs* (12)	*ps* (7)
*(14)	*qrs*	*rs* (10)	*rs* (10)
*(15)	*pqrs*	*pqrs* (15)	*prs* (13)

↑
New labels

'*': Final states

DFA Minimization (cont…)

Minimized DFA

Q' \ Σ	0	1
1	5	1
2	3	3
3	4	–
*4	4	4
5	11	6
6	12	1
*7	12	7
8	4 •	3
*9	4 •	4 •
11	12 •	6
*12	12 •	7 •

'*': Final states
'•': Modified entries due to replacement

DFA Minimization (cont…)

Further minimized DFA

Σ / Q′	0	1
1	5	1
2	3	3
3	4	–
*4	4	4
5	11	6
6	7 •	1
*7	7 •	7
8	4	3
11	7 •	6

'*': Final states
'•': Modified entries due to replacement

NFA withε – Transitions

- 5-tuple:
- M= (Q,Σ, δ, q_o, F) where,
  - Q: Finite set of states
  - Σ: Finite input alphabet
  - **δ: STF that maps Q X (Σ $\cup$ ε) to 2^Q**

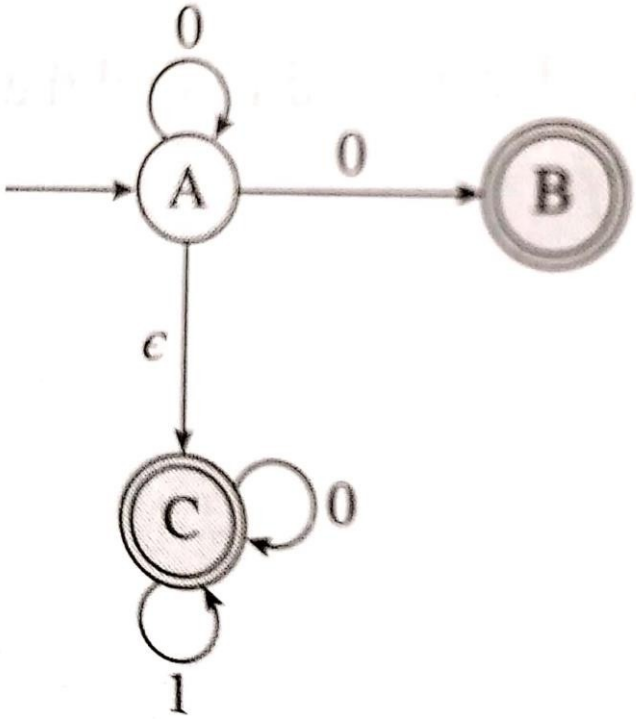
i.e. Q X (Σ $\cup$ ε) { } $\longrightarrow$ 2^Q

  - q_o: Initial state of FA, $q_o \in Q$
  - F: Set of final states, F $\subseteq$ Q

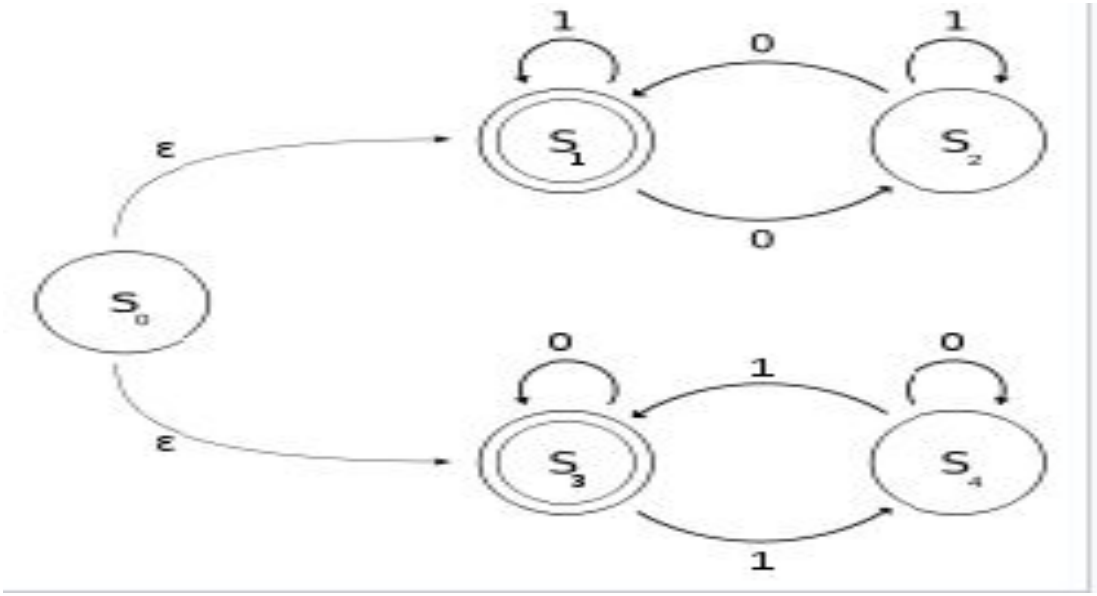
NFA withε – Transitions (cont..)

- **Example:**

Transition Graph for example NFA with ε − transitions

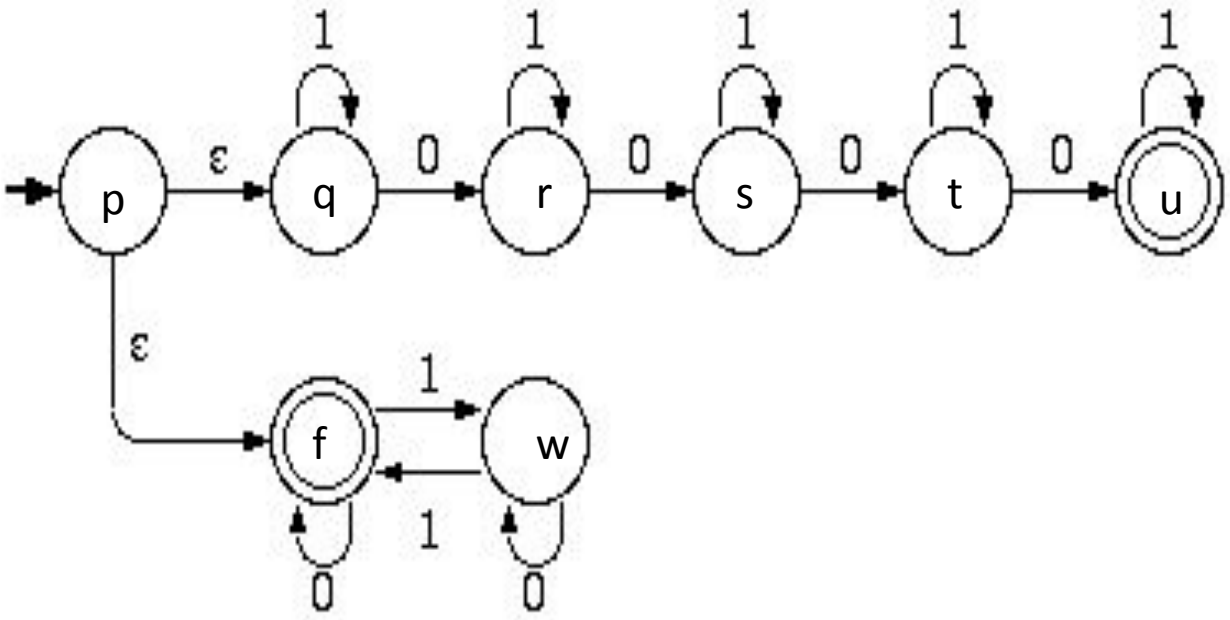


NFA with Ɛ moves Example



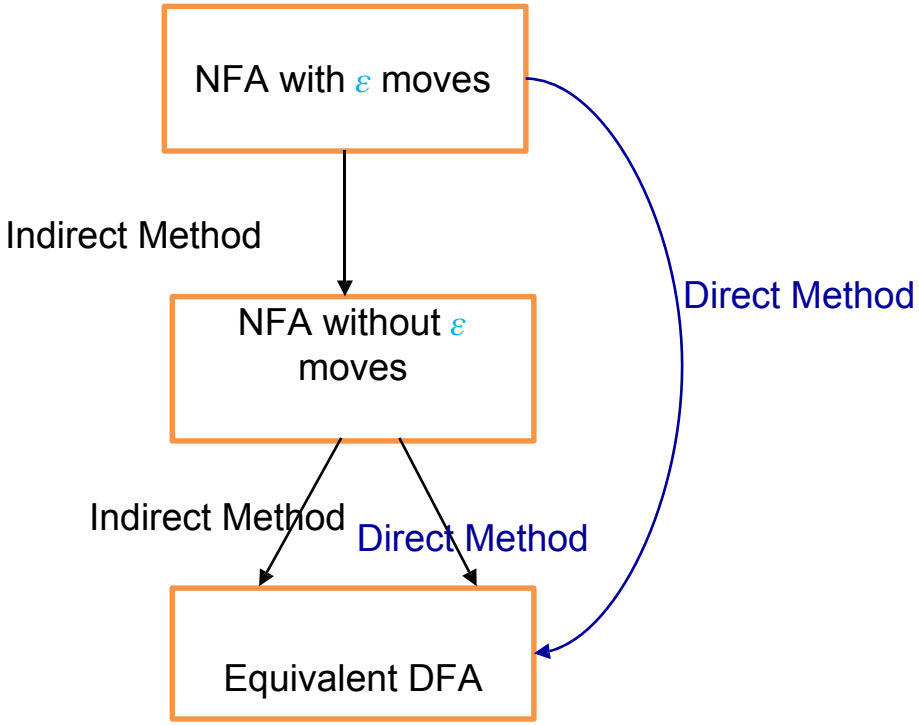
NFA with Ɛ moves Example

All strings containing exactly 4 0s or an even number of 1s.Set of
valid strings=
{ε,11,0000,0101,01010101,1110000,0000111}



Conversion of NFA with ε – Transitions to DFA

- Indirect Conversion Method
- Direct Conversion Method



Indirect Conversion Method

1. Construct the NFA without $\varepsilon -$ *transitions* from the given NFA with $\varepsilon -$ *transitions*
   - $(\quad)$
   - q, a = ε-closure ($\delta(\delta\hat{}(q, \varepsilon), a)$)
2. Construct an equivalent DFA with this newly constructed NFA without $\varepsilon -$ *transitions* (using existing methods)

Indirect Conversion Method (cont…)

- ε-closure of a state:
  - Set of all states p, such that there is a path from state q to state p labeled 'ε', is known as ε-closure (q)
  - Set of all the states having distance zero from state q
  - Every state is at distance zero from itself
  - Denoted by δ^

Indirect Conversion Method

(cont…)

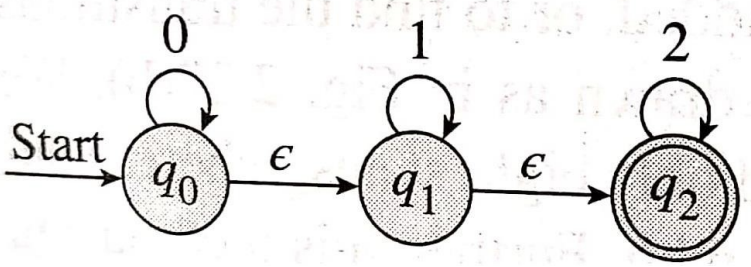
- **Example:**

  Convert the NFA with ε −transitions given in the figure to its equivalent DFA

State Transition table for example NFA with ε −transitions

$\Sigma \cup \{\epsilon\}$ / Q	0	1	2	ϵ
q_0	$\{q_0\}$	ϕ	ϕ	$\{q_1\}$
q_1	ϕ	$\{q_1\}$	ϕ	$\{q_2\}$
q_2	ϕ	ϕ	$\{q_2\}$	ϕ

State Transition Graph for example NFA with ε −transitions

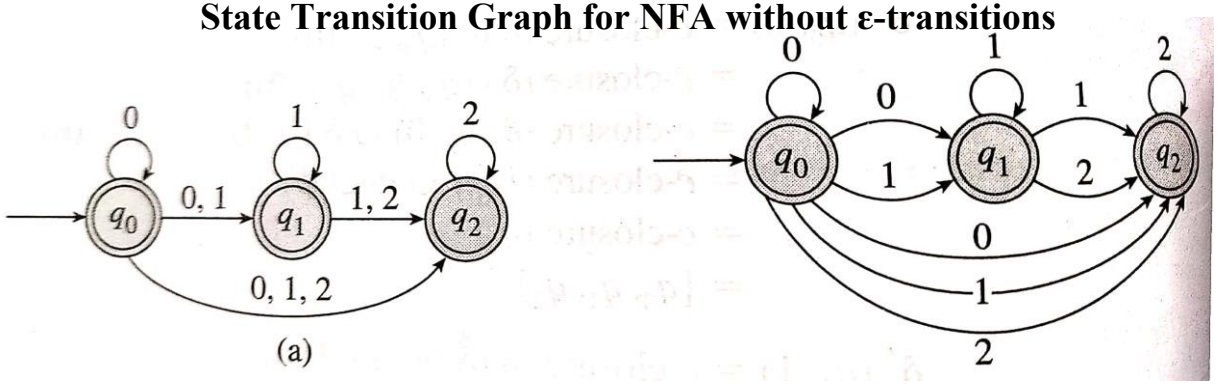


Indirect Conversion Method

(cont…)

1. Conversion of NFA with ε −transitions to NFA without ε −transitions:

State Transition Graph for NFA without ε-transitions



State Transition Table for NFA without ε-transitions

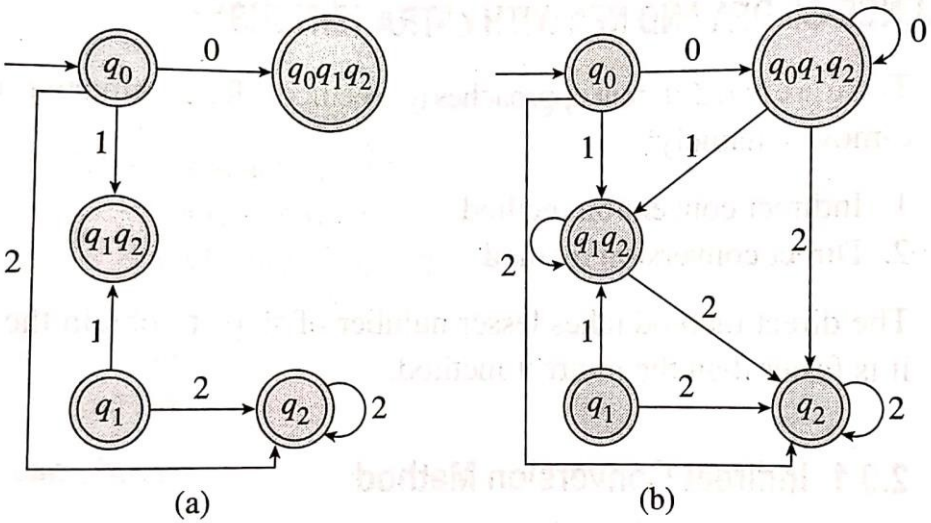
Q \ Σ	0	1	2
q_0	$\{q_0, q_1, q_2\}$	$\{q_1, q_2\}$	$\{q_2\}$
q_1	ϕ	$\{q_1, q_2\}$	$\{q_2\}$
q_2	ϕ	ϕ	$\{q_2\}$

Indirect Conversion Method

(cont…)

2. Conversion of NFA without ε −transitions to an equivalent DFA:

(a) Step 1 (b) Step 2 (c) Final DFA



Indirect Conversion Method

(cont…)

2. Conversion of NFA without ε −transitions to an equivalent DFA:

State Transition table for the equivalent DFA

Q' \ Σ	0	1	2
$*q_0$	$q_0q_1q_2$	q_1q_2	q_2
$*q_1$	–	q_1q_2	q_2
$*q_2$	–	–	q_2
$*q_1q_2$	–	q_1q_2	q_2
$*q_0q_1q_2$	$q_0q_1q_2$	q_1q_2	q_2

'*': Final states

Reduced State Transition table for the equivalent DFA

Q' \ Σ	0	1	2
$*q_0$	q_0 •	q_1 •	q_2
$*q_1$	–	q_1 •	q_2
$*q_2$	–	–	q_2

'*': Final states

'•': Modified entries

Direct Conversion Method

- Begin with the initial state
- Go on adding the states as & when required to the diagram
- Follow the same process until there exists no state without having the transitions specified
- Each state label consists of:
    - Name of state label
    - Combination of all the state symbols, which are reachable from the given state

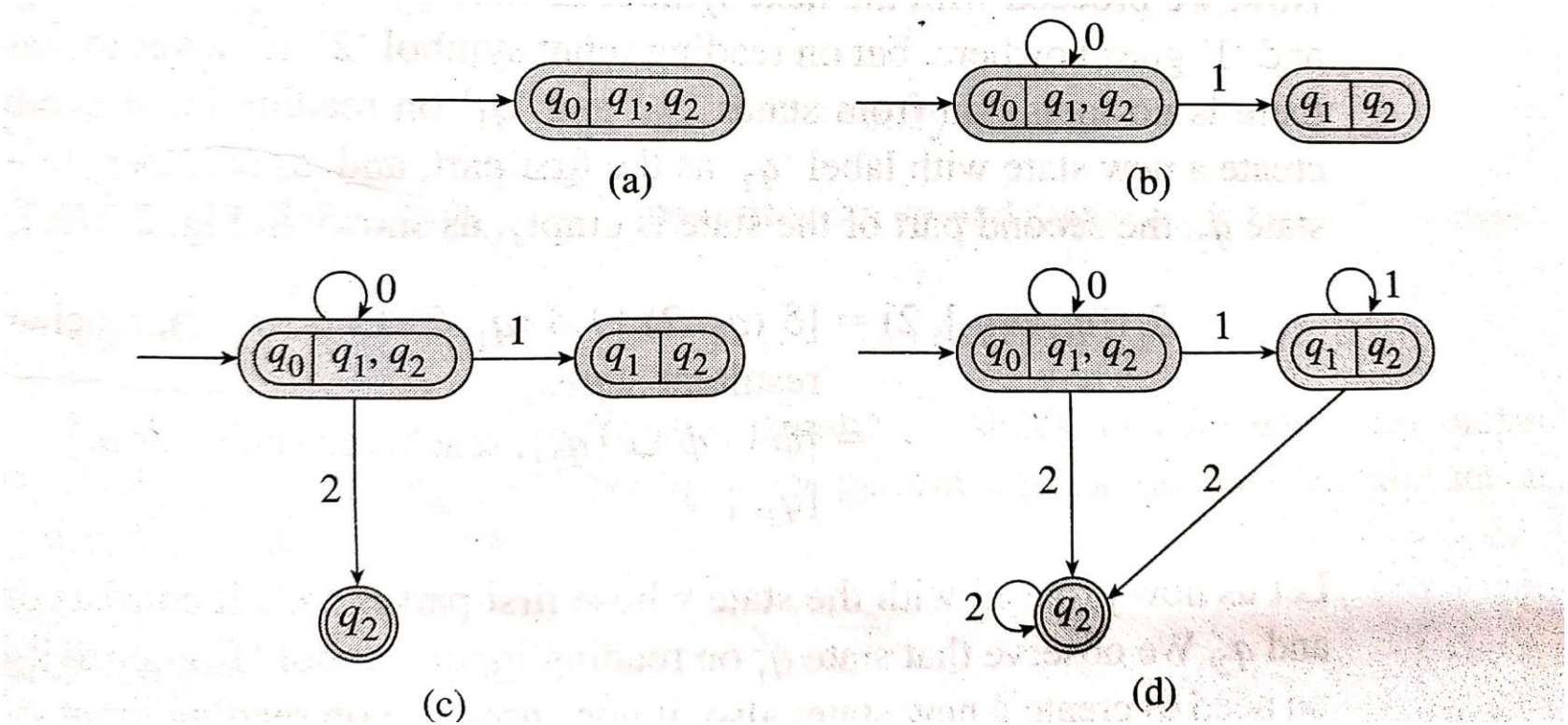
Direct Conversion Method (cont…)

- Start building an equivalent DFA with the transition diagram of given NFA
- Find the transition, one state at a time
- If the next state of a given transition is a new combination, add that to the set of states for the resultant DFA

Direct Conversion Method
(cont…)

- **Example (Refer slide no. 53 for example NFA with ε −transitions**

Resultant DFA (a) Step 1 (b) Step 2 (c) Step 3 (d) Final DFA



FA with output

Machine generates an output on every input. The value of output is a function of current state and the current input.

Such machines are characterised by two behaviours

State transition function(δ)

Output function(λ)

Types of Automata with transition

1) Mealy machine
2) Moore machine

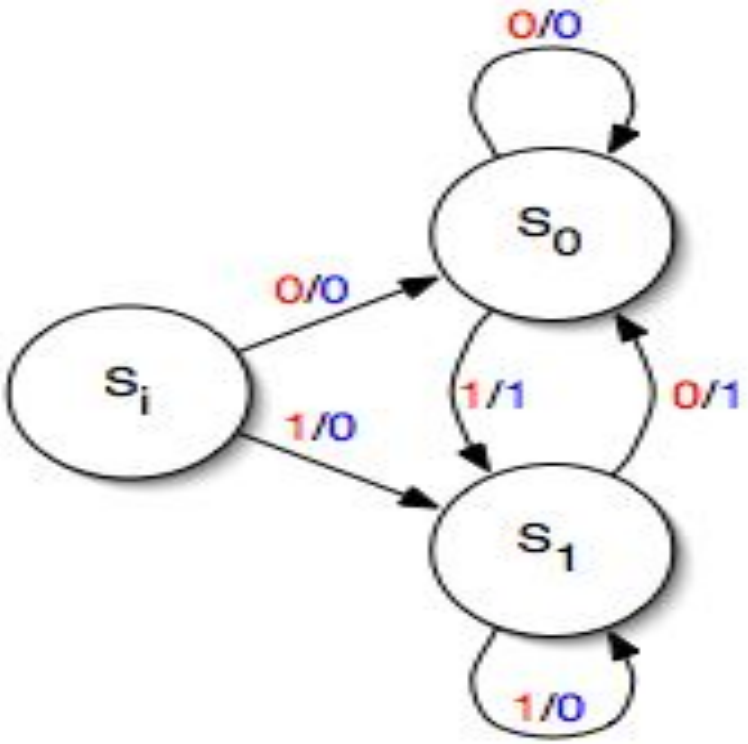
Mealy Machine (cont…)

- Six-tuple
- M= $(Q, \Sigma, \Delta, \delta, \lambda, q_o)$

  where,
  - Q: Finite set of states
  - Σ: Finite input alphabet
  - Δ: An output alphabet
  - δ: STF that maps Q X Σ to Q,
  - i.e. Q X ΣQ
  - **λ: Machine function (MAF); λ: Q X Σ Δ**
  - q_o: Initial state of the machine, $q_o \in Q$

Mealy Machine Example

Mealy machine for exclusive-or of the two most-recent input values



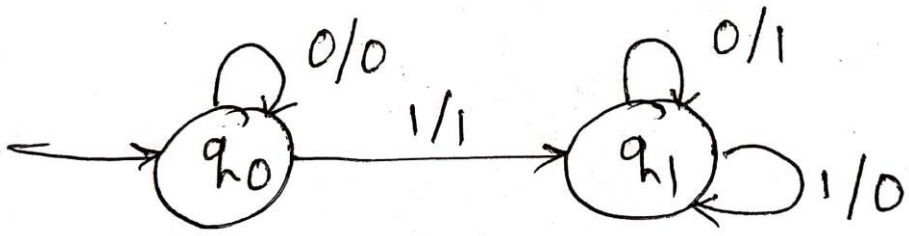
Moore Machine

- The output symbol at any given time depends only upon the current state of the machine (and not on the input symbol read)
- An output symbol is associated with each state
- When the machine is in particular state, it generates the output, irrespective of the input that caused the transition

Mealy Machine

- A machine with finite number of states, and for which the output symbol at any given time is a function of (i.e. it depends on) the current input symbol as well as the current state of the machine
  - **Example:**
  - Construct a Mealy machine to find 2's complement of a given binary number

Example Mealy Machine



Moore Machine

- The output symbol at any given time depends only upon the current state of the machine (and not on the input symbol read)
- An output symbol is associated with each state
- When the machine is in particular state, it generates the output, irrespective of the input that caused the transition

Moore Machine (cont…)

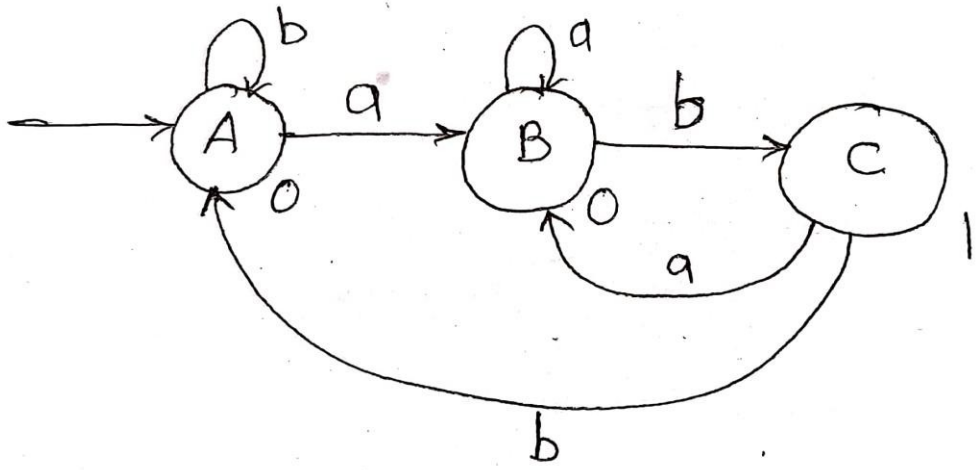
- Six-tuple
- M= $(Q, \Sigma, \Delta, \delta, \lambda, q_o)$

  where,
  - Q: Finite set of states
  - Σ: Finite input alphabet
  - Δ: An output alphabet
  - δ: STF that maps Q X Σ to Q,
  - i.e. Q X Σ Q
  - **λ: Machine function (MAF); λ: Q Δ**
  - q_o: Initial state of the machine, $q_o \in Q$

Moore Machine (cont…)

- **Example:**
  - Construct a Moore machine that takes set of all strings over {a,b} as an input and prints ‘1’ as an output for every occurrence of ‘ab’ as a substring

Example Moore Machine



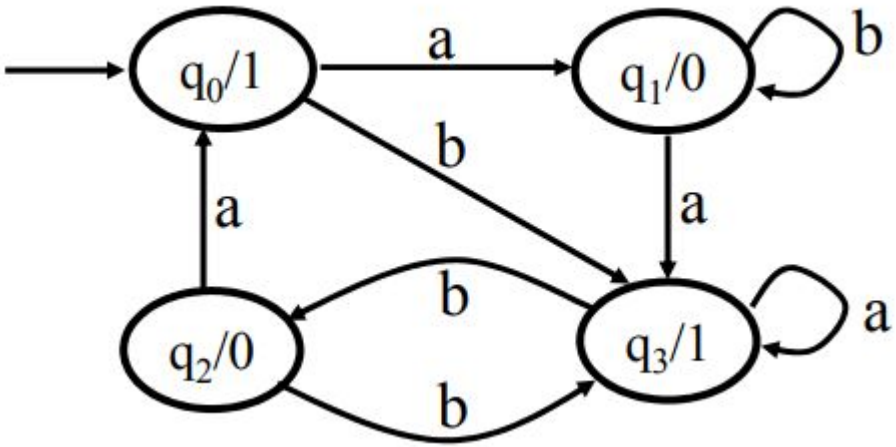
Moore Machine Example

$states = \{q_0, q_1, q_2, q_3\}$ $\quad \Sigma = \{a, b\}$ $\quad \Gamma = \{0, 1\}$

Old state	Output by the old state	New state	
		After input a	After input b
$\rightarrow q_0$	1	q_1	q_3
q_1	0	q_3	q_1
q_2	0	q_0	q_3
q_3	1	q_3	q_2

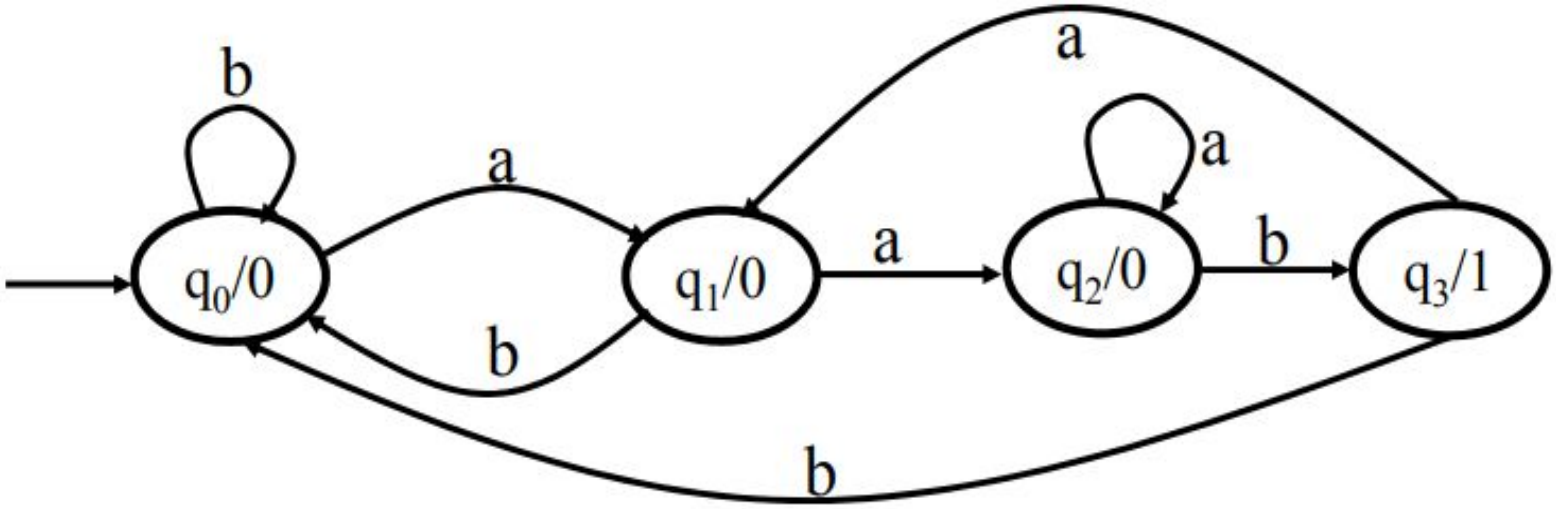
abab

10010



Moore Machine Example

Machine that counts occurrences of aab



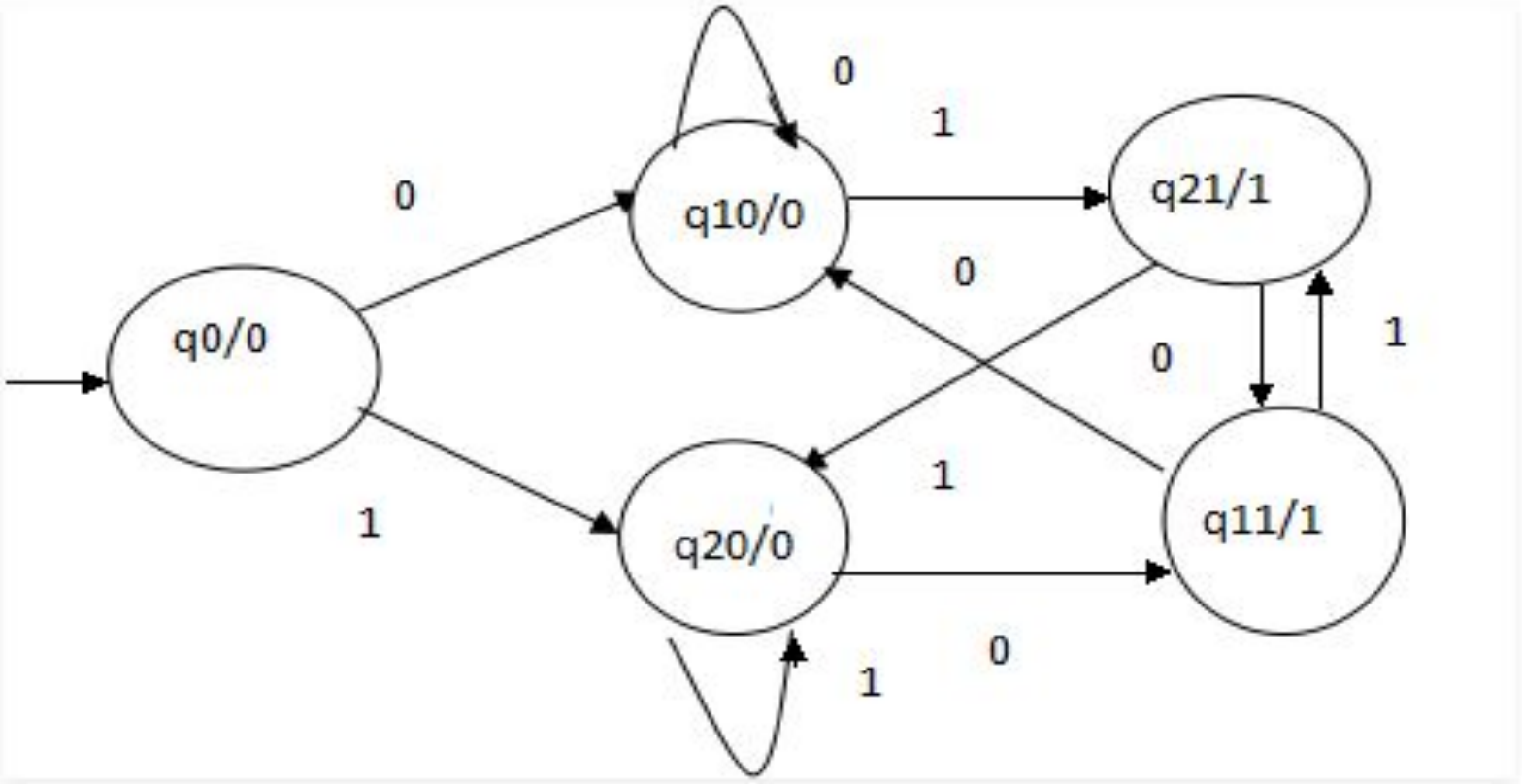
Input		a	a	a	b	a	b	b	a	a	b	b
State	q_0	q_1	q_2	q_2	q_3	q_1	q_0	q_0	q_1	q_2	q_3	q_0
Output	0	0	0	0	1	0	0	0	0	0	1	0

Conversion of Moore Machine to Mealy Machine

- Moore Machine:
  - M1= $(Q, \Sigma, \Delta, \delta, \lambda, q_o)$
- Equivalent Mealy Machine:
  - M2= $(Q, \Sigma, \Delta, \delta, \lambda', q_o)$ where,
  - $\lambda'(q, a) = \lambda(\delta(q, a))$

Moore to Mealy Machine

- Let us take moore machine and it's transition Table.



Moore to Mealy Machine

- It's transition Table.

	Input=0	Input=1	
Present State	Next State	Next State	Output
q0	q10	q20	0
q10	q10	q21	0
q11	q10	q21	1
q20	q11	q20	0
q21	q11	q20	1

Algorithm

Step1: Construct an empty mealy machine using all states of moore machine as shown in Table

	Input=0		Input=1	
Present State	Next State	Output	Next State	Output
q0				
q10				
q11				
q20				
q21				

Algorithm

Step 2: Next state for each state can also be directly found from moo re machine transition Table as:

	Input=0		Input=1	
Present State	Next State	Output	Next State	Output
q0	q10		q20	
q10	q10		q21	
q11	q10		q21	
q20	q11		q20	
q21	q11		q20	

Algorithm

Step 3: As we can see output corresponding to each input in moore machine transition table. Use this to fill the Output entries.

	Input=0		Input=1	
Present State	Next State	Output	Next State	Output
q0	q10	0	q20	0
q10	q10	0	q21	1
q11	q10	0	q21	1
q20	q11	1	q20	0
q21	q11	1	q20	0

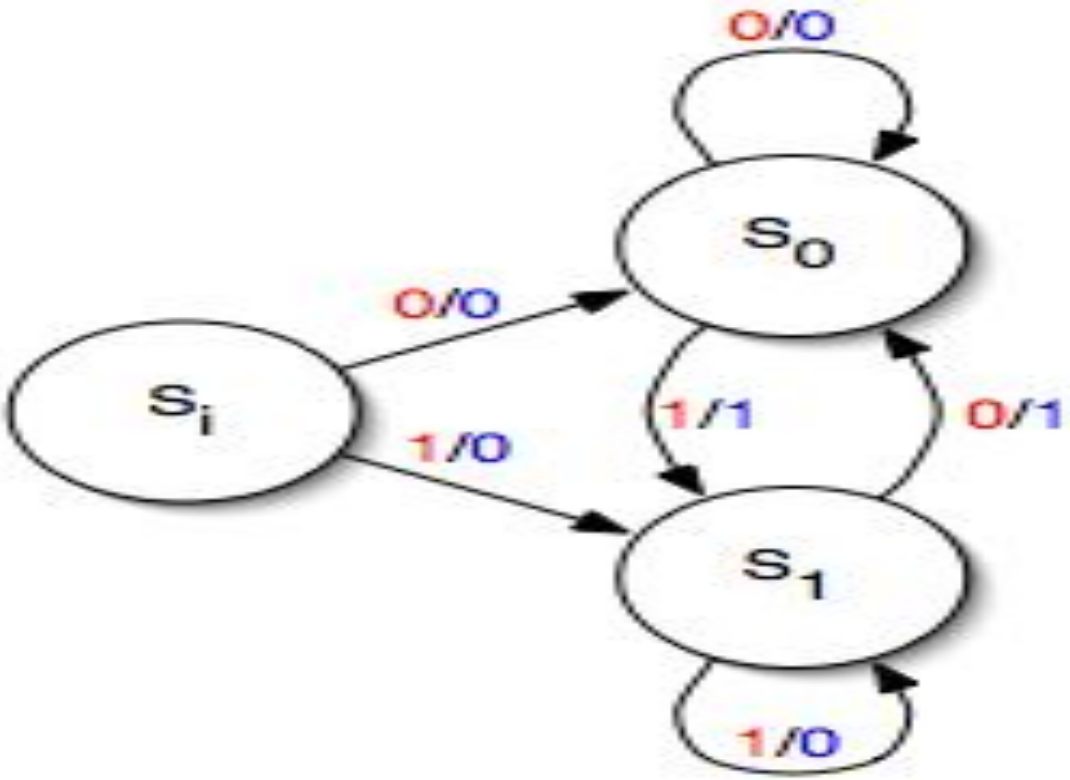
Algorithm

Step4: As we can see from above table, q10 and q11 are similar to each other (same value of next state and Output for different Input). Similarly, q20 and q21 are also similar. So, q11 and q21 can be eliminated.

	Input=0		Input=1	
Present State	Next State	Output	Next State	Output
q0	q10	0	q20	0
q10	q10	0	q21	1
q20	q11	1	q20	0

Final Mealy Machine

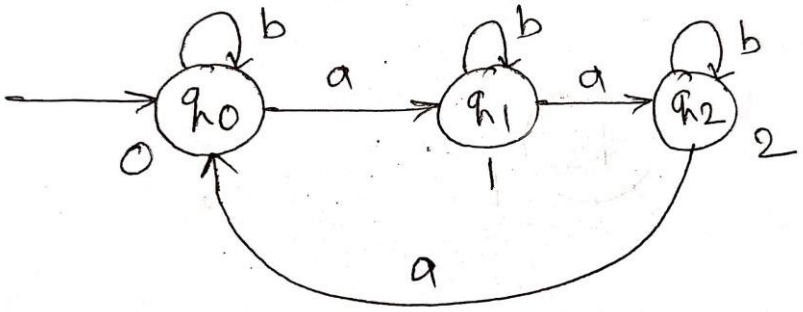
Number of states in Mealy machine can't be greater than number of states in Moore machine



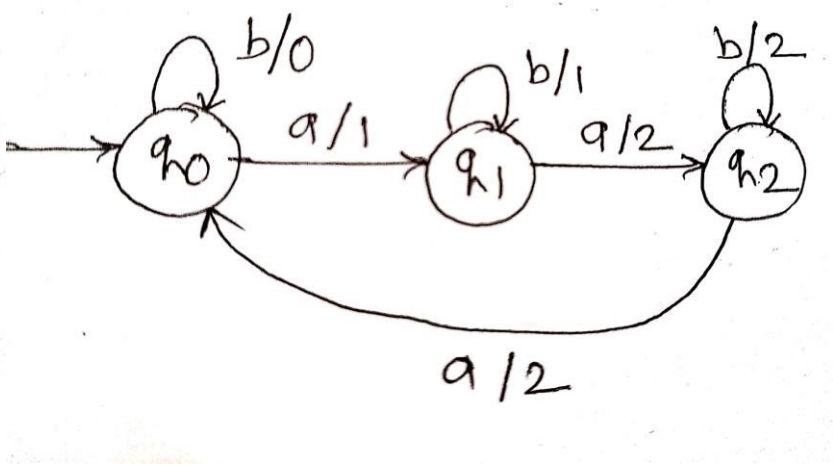
Conversion of Moore Machine to Mealy Machine (cont…)

- **Example:**
  - Convert the given Moore machine to an equivalent Mealy Machine

Example Moore Machine



Equivalent Mealy Machine



Conversion of Mealy Machine to Moore Machine

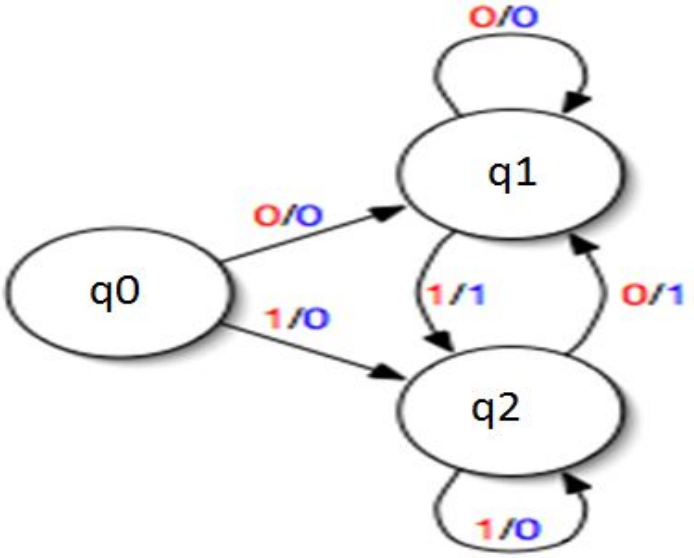
- Mealy Machine:
  - M1= $(Q, \Sigma, \Delta, \delta, \lambda, q_o)$
- Equivalent Moore Machine:
  - M2= $([QX\ \Delta]), \Sigma, \Delta, \delta', \lambda', [q_o, b_o]$

    where,
  - b_o is an arbitrarily selected member of Δ
  - $\delta'([q, b], a) = [\delta(q, a), \lambda(q, a)]$
  - $\lambda'([q, b]) = b$

    b: output symbol a: input symbol

Mealy to Moore Machine

- Let us take mealy machine and it's transition Table.



	Input=0		Input=1	
Present State	Next State	Output	Next State	Output
q0	q1	0	q2	0
q1	q1	0	q2	1
q2	q1	1	q2	0

Algorithm

Step 1. Find states having more than 1 outputs associated with them.
Here we have q1 and q2.

Step 2. Create two states for these states.
For q1, two states will be q10 (state with output 0) and q11 (state with output 1). Similarly for q2, two states will be q20 and q21.

Algorithm

Step 3. Create an empty moore machine with new generated state. For moore machine, Output will be associated to each state irrespective of inputs.

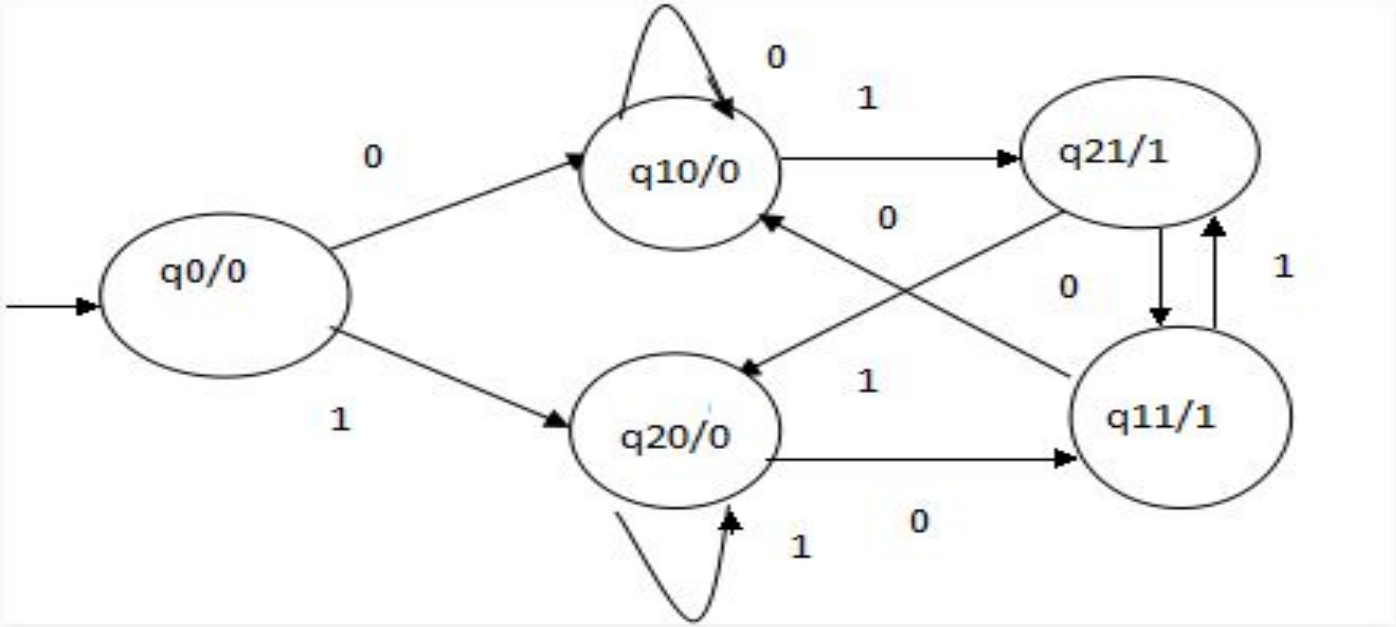
	Input=0	Input=1	
Present State	Next State	Next State	Output
q0			
q10			
q11			
q20			
q21			

Algorithm

Step 4. Fill the entries of next state using mealy machine transition table shown in Table. For q0 on input 0, next state is q10 (q1 with output 0).
Similarly, for q0 on input 1, next state is q20 (q2 with output 0).

	Input=0	Input=1	
Present State	Next State	Next State	Output
q0	q10	q20	0
q10	q10	q21	0
q11	q10	q21	1
q20	q11	q20	0
q21	q11	q20	1

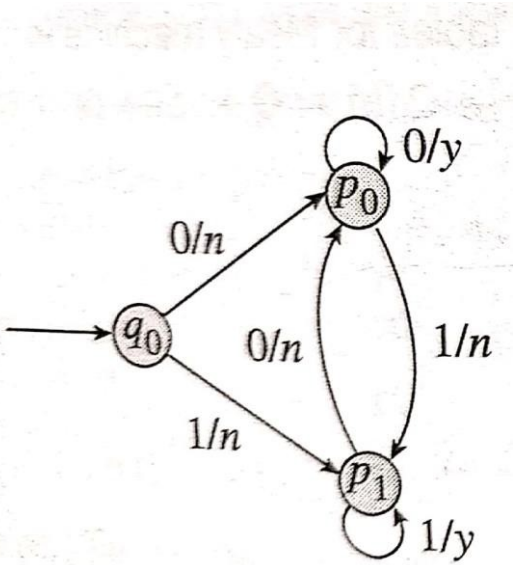
Final Moore Machine

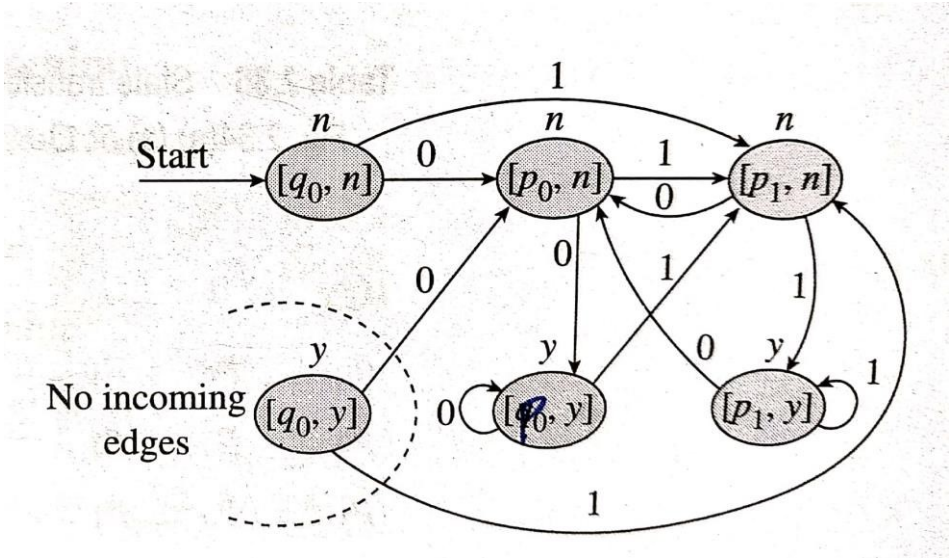


Conversion of Mealy Machine to Moore Machine (cont…)

- **Example:**
  - Convert the given Mealy Machine to an equivalent Moore Machine

Example Mealy Machine **Equivalent Moore Machine**





Comparison of Moore and Mealy Machine

Moore Machine	Mealy Machine
Output depends only upon current state	Output depends upon both current input symbol and current state
Generally, it has more states than Mealy machine	Generally, it has fewer states than Mealy machine
If the input string is of length n, then the output string is of length n+1. This is because first state also produces output	If the input string is of length n, then the output string is also of length n
$\lambda: Q \xrightarrow{\quad} \Delta$	$\lambda: Q \times \Sigma \xrightarrow{\quad} \Delta$