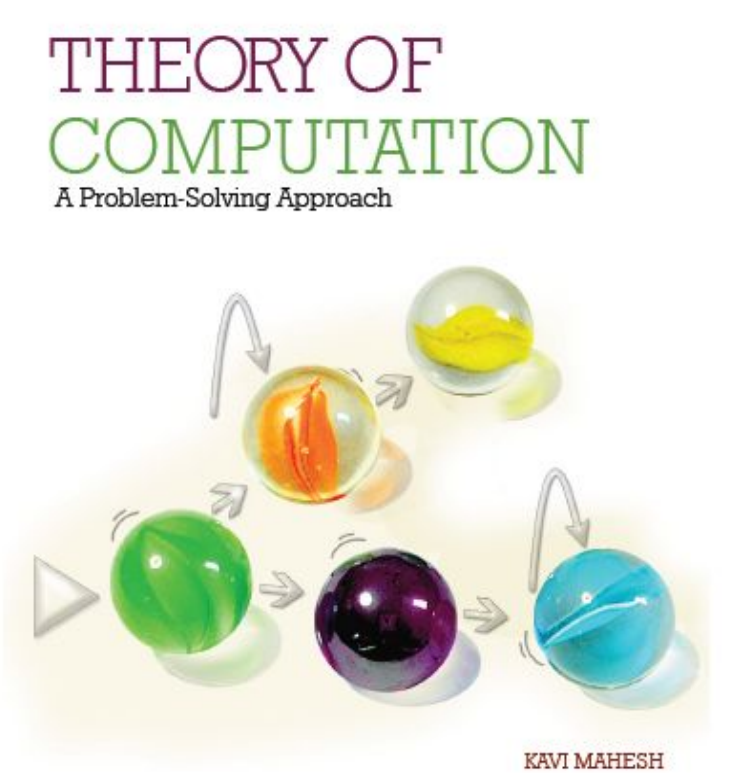


Theory of Computation: A Problem-Solving Approach

Chapter – 2: Automata

Dr. Kavi Mahesh

Learning Objectives

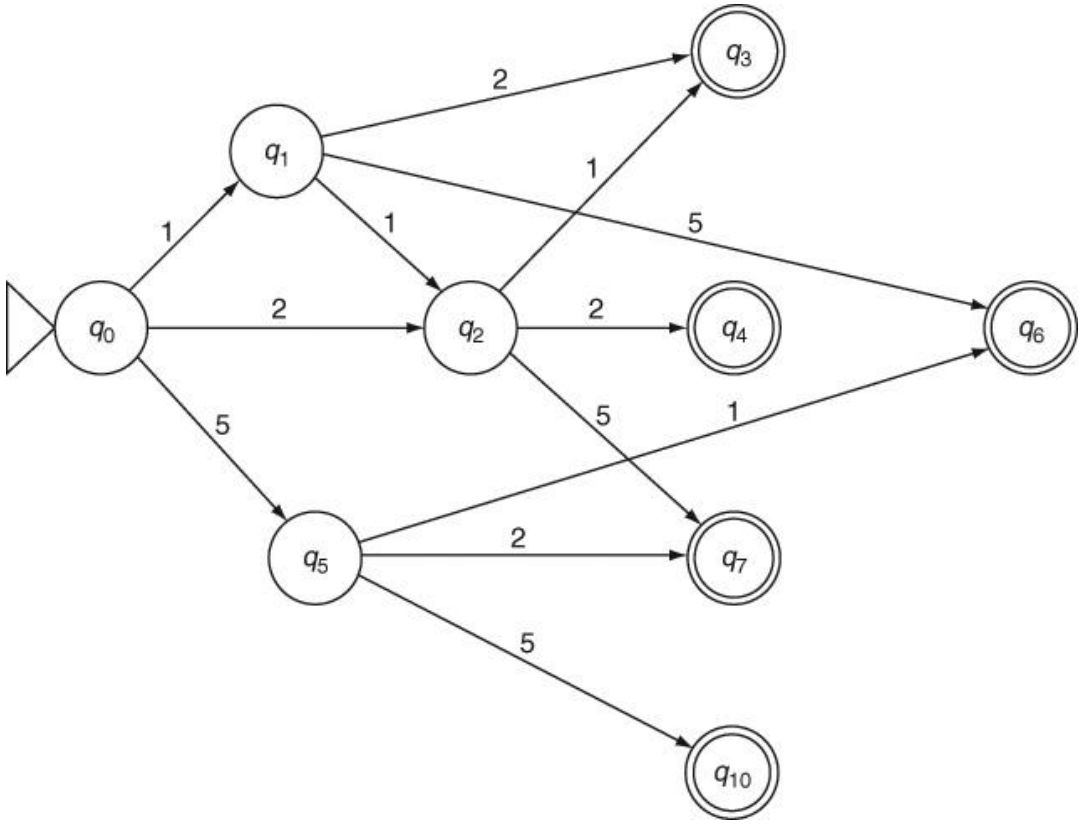
- Recognize that everyday machines such as vending machines are computing devices.
- Learn to design a finite automaton for a given problem.
- Learn to handle end conditions in automata.
- Learn to handle reject states in automata.
- Learn to use states as memory.
- Use a step-by-step method for constructing complex automata.
- Appreciate some limitations of deterministic finite automata.

Automata: Simple Computing Machines

- Input: a string of symbols
- Accept or reject at the end of the input
- Optionally, produce tangible output
- Comprised of a set of internal states
- No memory unit!

An Automatic Ticket Vending Machine

- E.g., a railway platform ticket vending machine
- One ticket Rs. 3; two tickets Rs. 6
- Accepts Rs. 1, 2, 5 coins



How Does it Work?

Rudimentary "Operating System":

- Start in the initial state q_0.
- Wait for an input symbol.
- If a symbol is received as input,
  - Examine current state for an outgoing arrow marked with the given input symbol.
  - If a matching arrow is found, traverse the arrow, change to the state at which the arrow is pointing and consume the input symbol.
  - Wait for the next input symbol; continue processing until input is exhausted.
- When the input is exhausted, if the current state is a final state, the input is declared as accepted; otherwise, the input is declared as rejected by the machine.

Automaton: Definition

An automaton M has five elements, which are defined as follows:

- M.alphabet, denoted by Σ, is called the alphabet and is the finite set of symbols that can appear in the input.
- M.states, also denoted by Q, is the finite set of all states in the automaton.
- M.startState, usually denoted by q0, is the start state of the automaton.
- M.finalStates, denoted by QF, is the subset of M.states that are the final states of the automaton. Also known as accepting states.
- M.transitionFunction is a function δ from Q × Σ to Q, that is, it is a mapping from the current state and the current input symbol to a new state.

AUTOMATON: VENDING MACHINE EXAMPLE

- M.alphabet = {1, 2, 5}
- M.states = {q0, q1, q2, q3, q4, q5, q6, q7, q10 }
- M.startState = q0
- M.finalStates = {q3, q4, q6, q7, q10 }
- M.transitionFunction =

$$\{(q0, 1) \rightarrow q1, (q0, 2) \rightarrow q2, (q0, 5) \rightarrow q5,$$

$$(q1, 1) \rightarrow q2, (q1, 2) \rightarrow q3, (q1, 5) \rightarrow q6,$$

$$(q2, 1) \rightarrow q3, (q2, 2) \rightarrow q4, (q2, 5) \rightarrow q7,$$

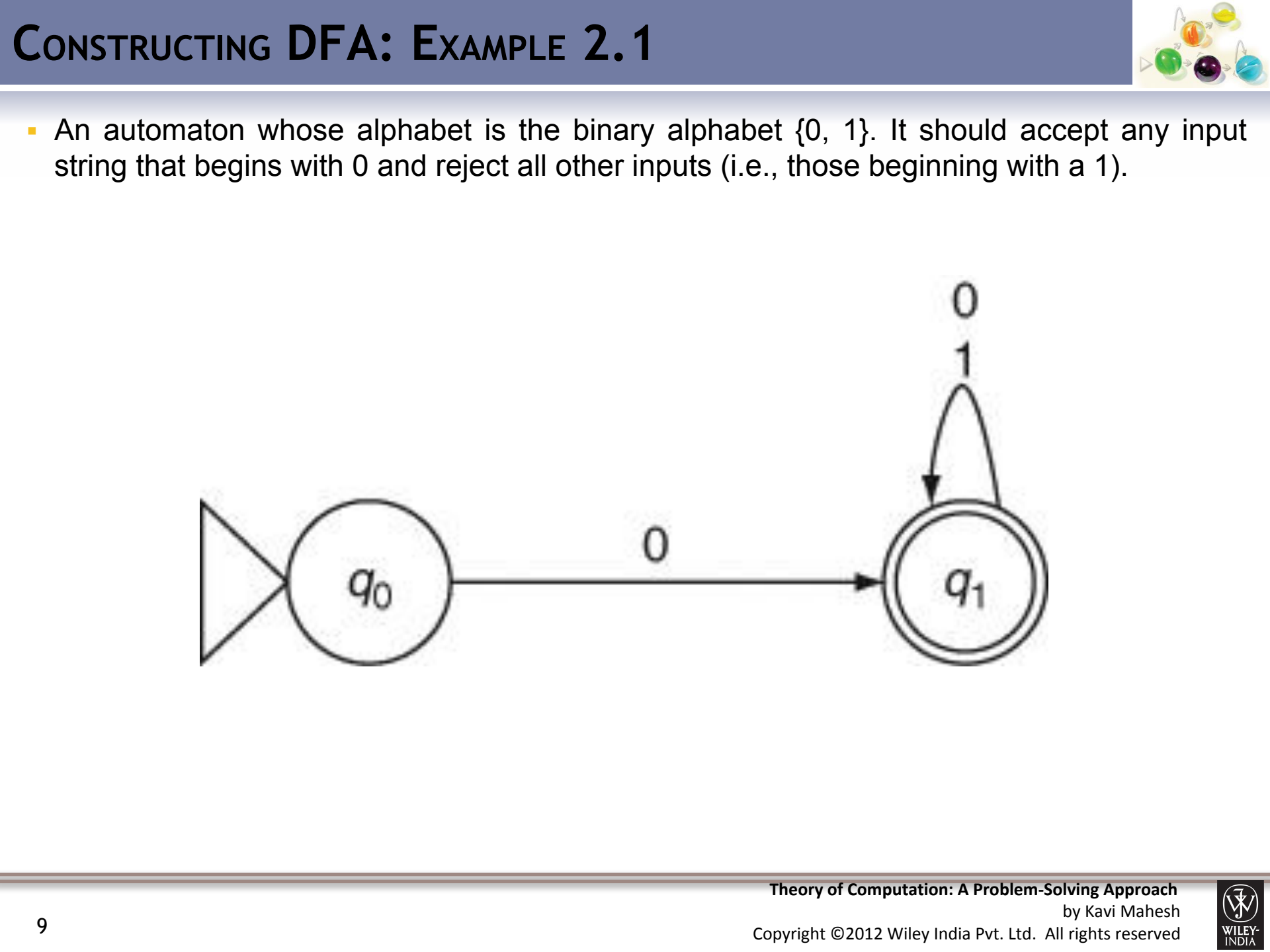
$$(q5, 1) \rightarrow q6, (q5, 2) \rightarrow q7, (q5, 5) \rightarrow q10\}$$

Deterministic Finite Automata (DFA)

A deterministic finite automaton meets two criteria:

- Every state has only one outgoing arrow for any given input symbol (i.e., the transition function is indeed a function).
- No transition can occur without consuming an input symbol, that is, the automaton cannot jump to a new state on its own volition.
- The ticket vending machine is actually a finite state transducer

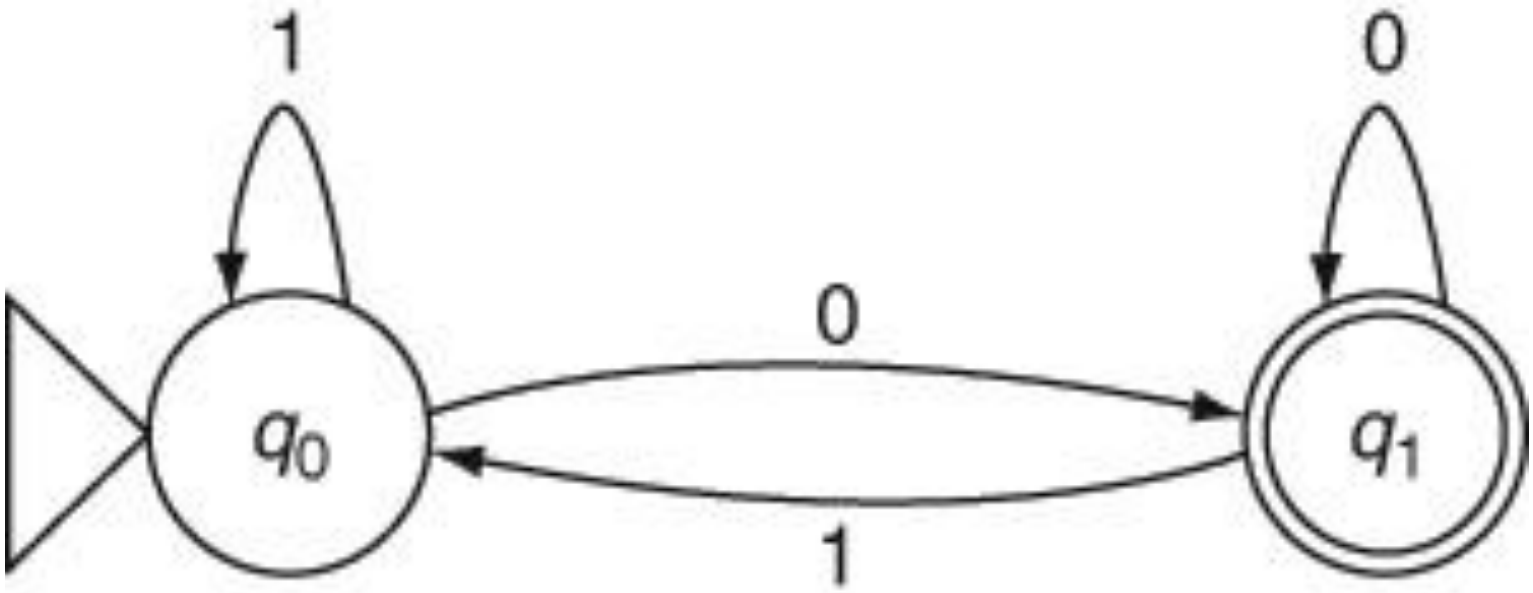
CONSTRUCTING DFA: EXAMPLE 2.1



- An automaton whose alphabet is the binary alphabet {0, 1}. It should accept any input string that begins with 0 and reject all other inputs (i.e., those beginning with a 1).

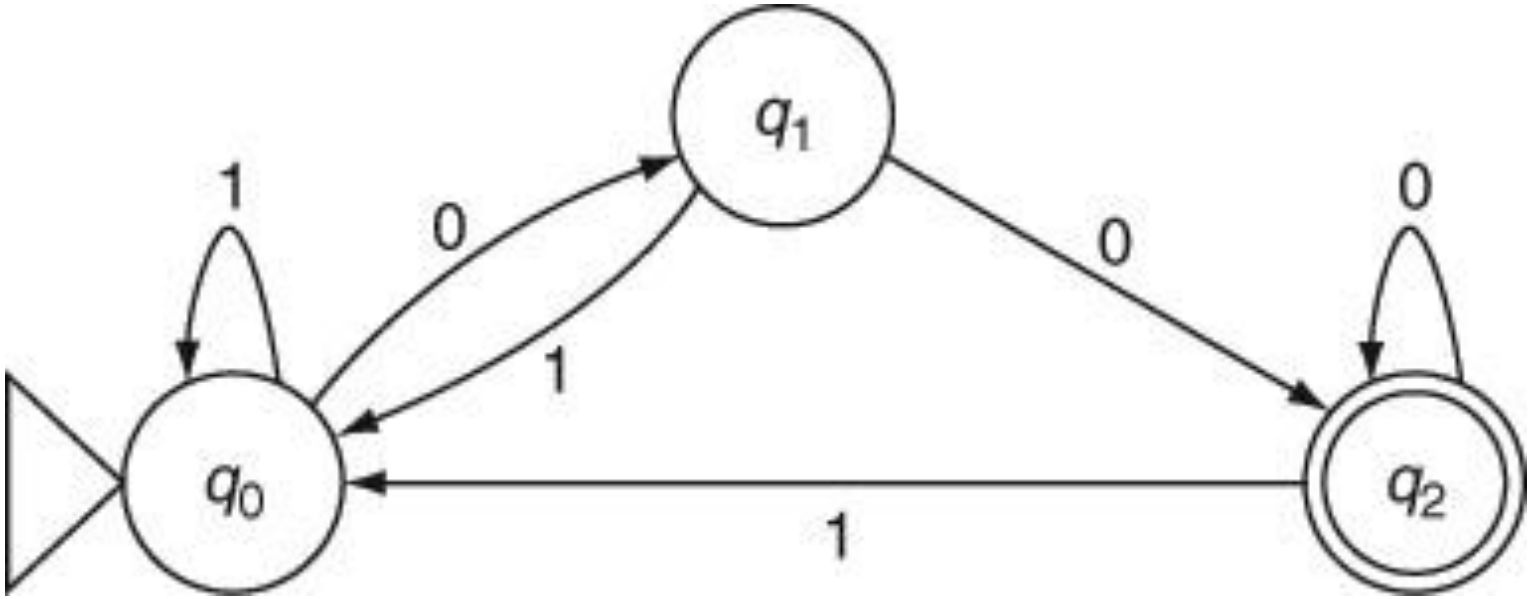
Handling End Conditions: Example 2.2

- Our next example is an automaton that recognizes numbers divisible by 2, that is, even numbers.



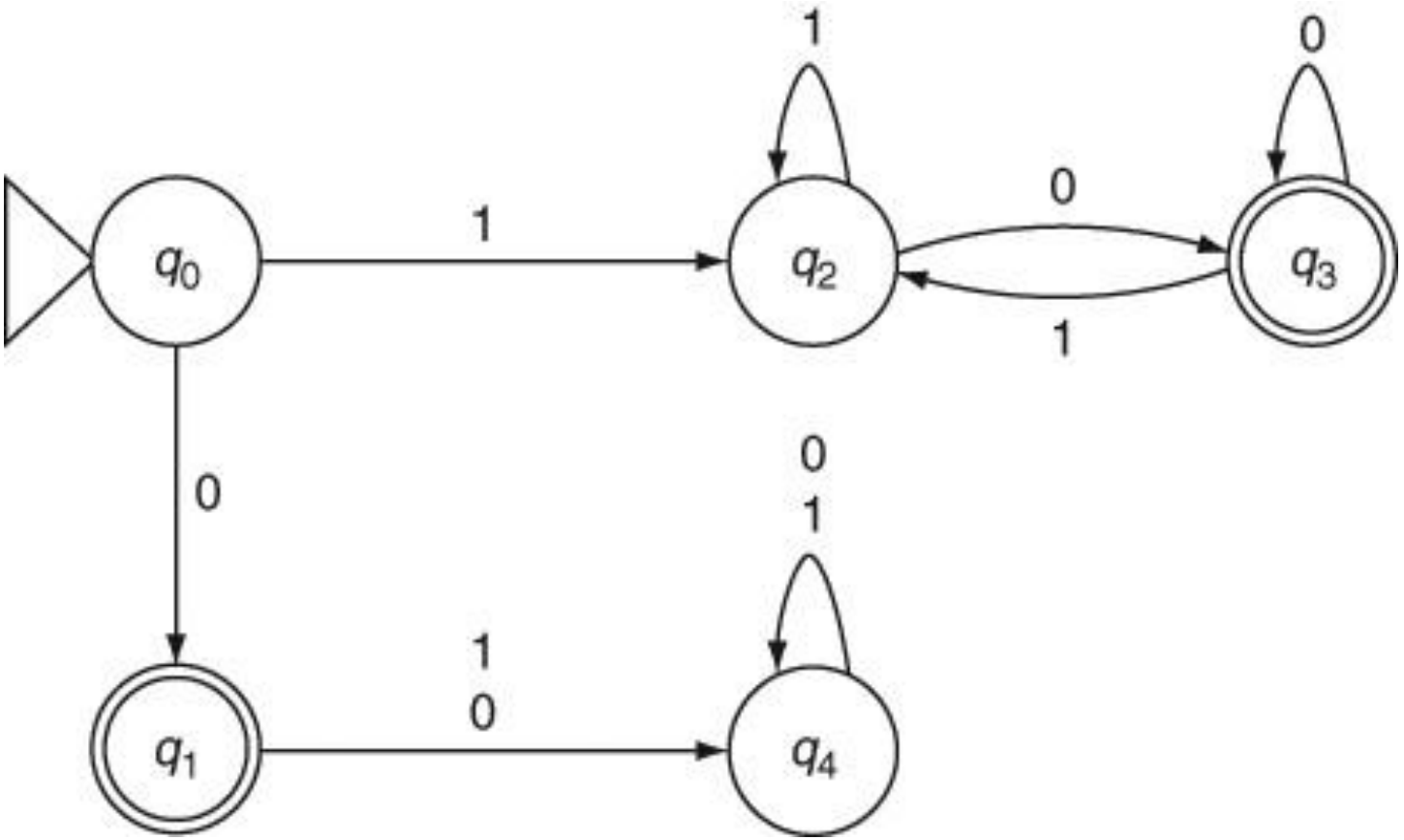
EXAMPLE 2.3

- Binary numbers divisible by 4



Reject States: Example 2.4

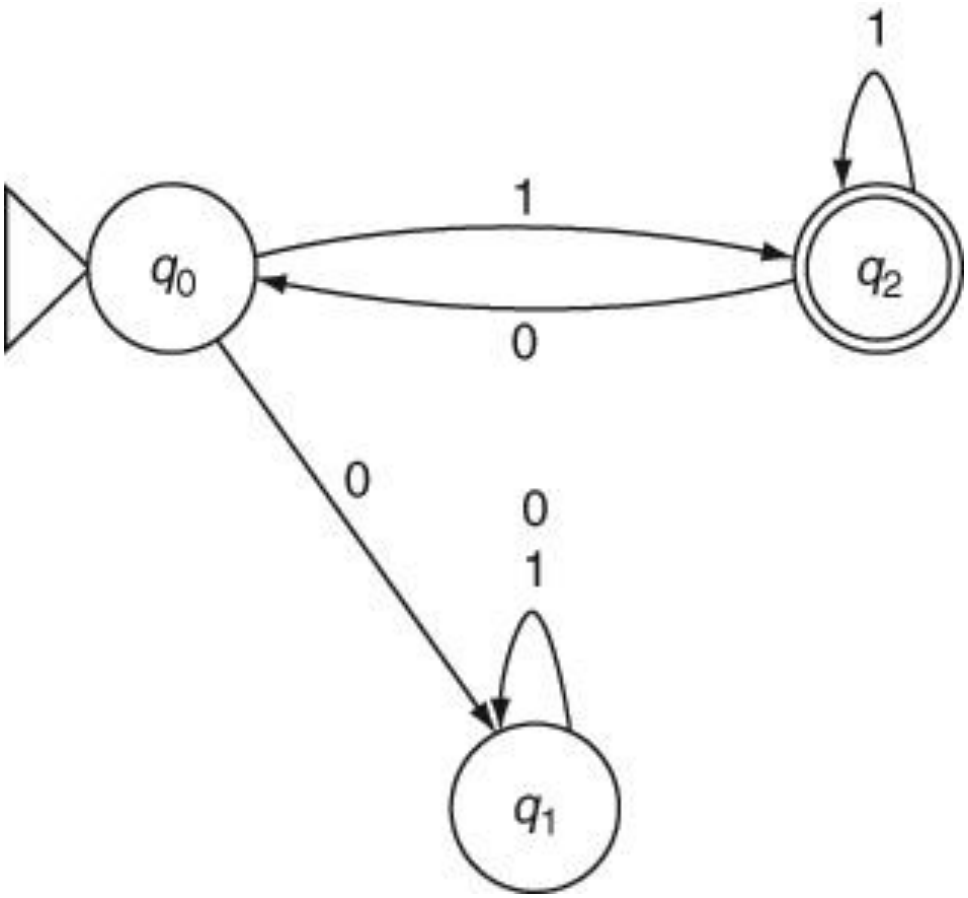
- Even numbers without leading zeros



Theory of Computation: A Problem-Solving Approach
by Kavi Mahesh

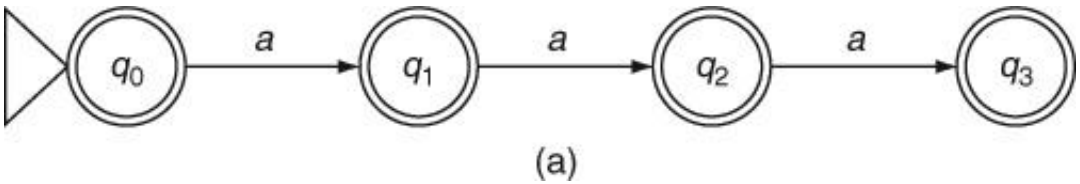
What is Wrong? Example 2.5

- Odd numbers without leading zeros
- What is wrong with this DFA?

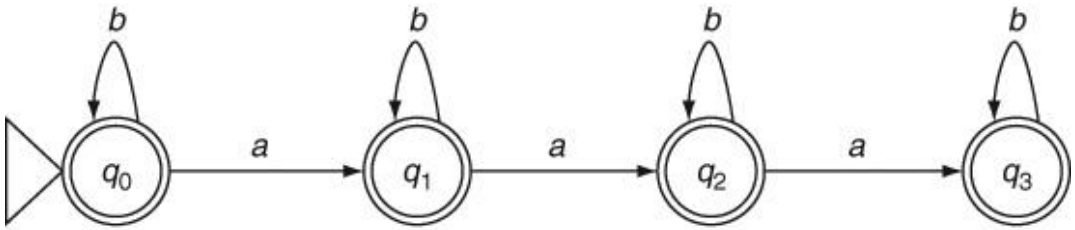


Step-by-Step Method: Example 2.6

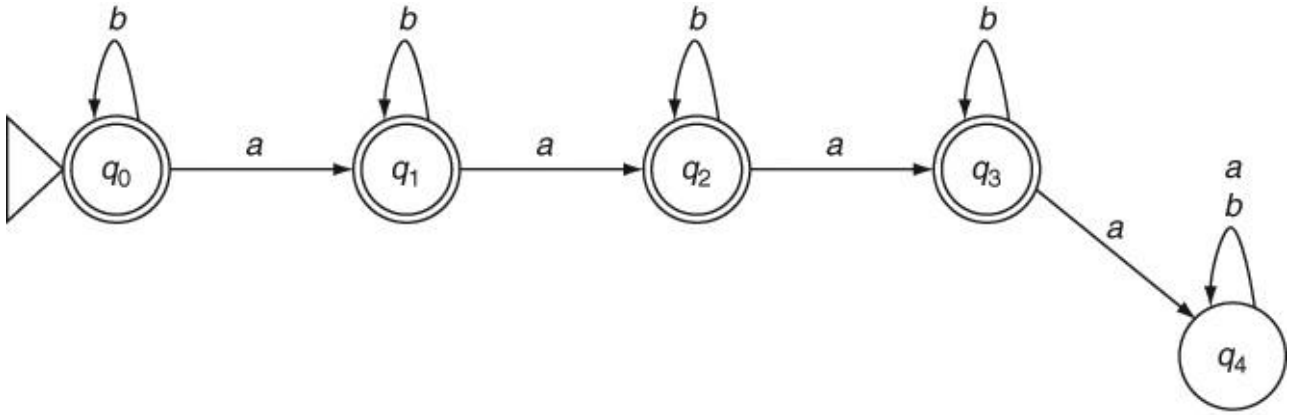
- DFA for strings over {a, b} containing at most 3 a s



(a)



(b)



(c)

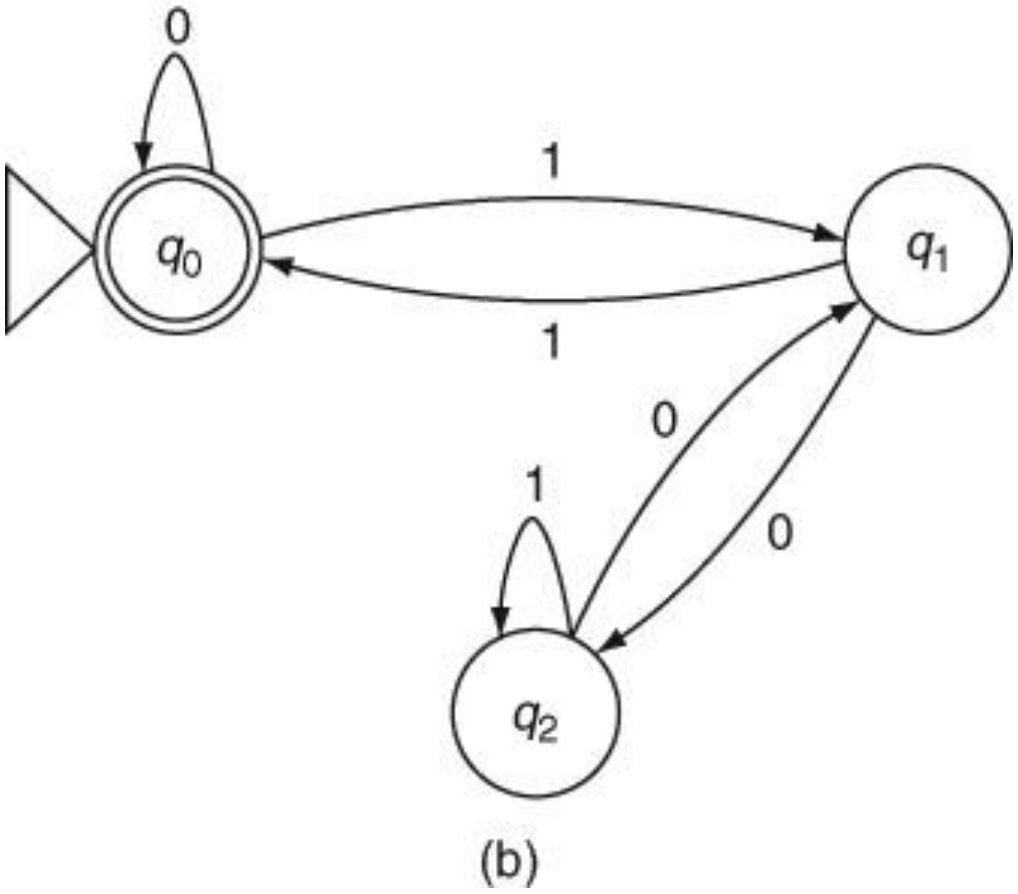
Transition Table for DFA

TABLE 2.1 Transition Table for the Automaton in Fig. 2.7(c)

State	Input = a	Input = b
$\rightarrow{}^{*}q_0$	q_1	q_0
${}^{*}q_1$	q_2	q_1
${}^{*}q_2$	q_3	q_2
${}^{*}q_3$	q_4	q_3
q_4	q_4	q_4

Constructing Complex DFA: Example 2.7

- Binary numbers divisible by 3



(b)

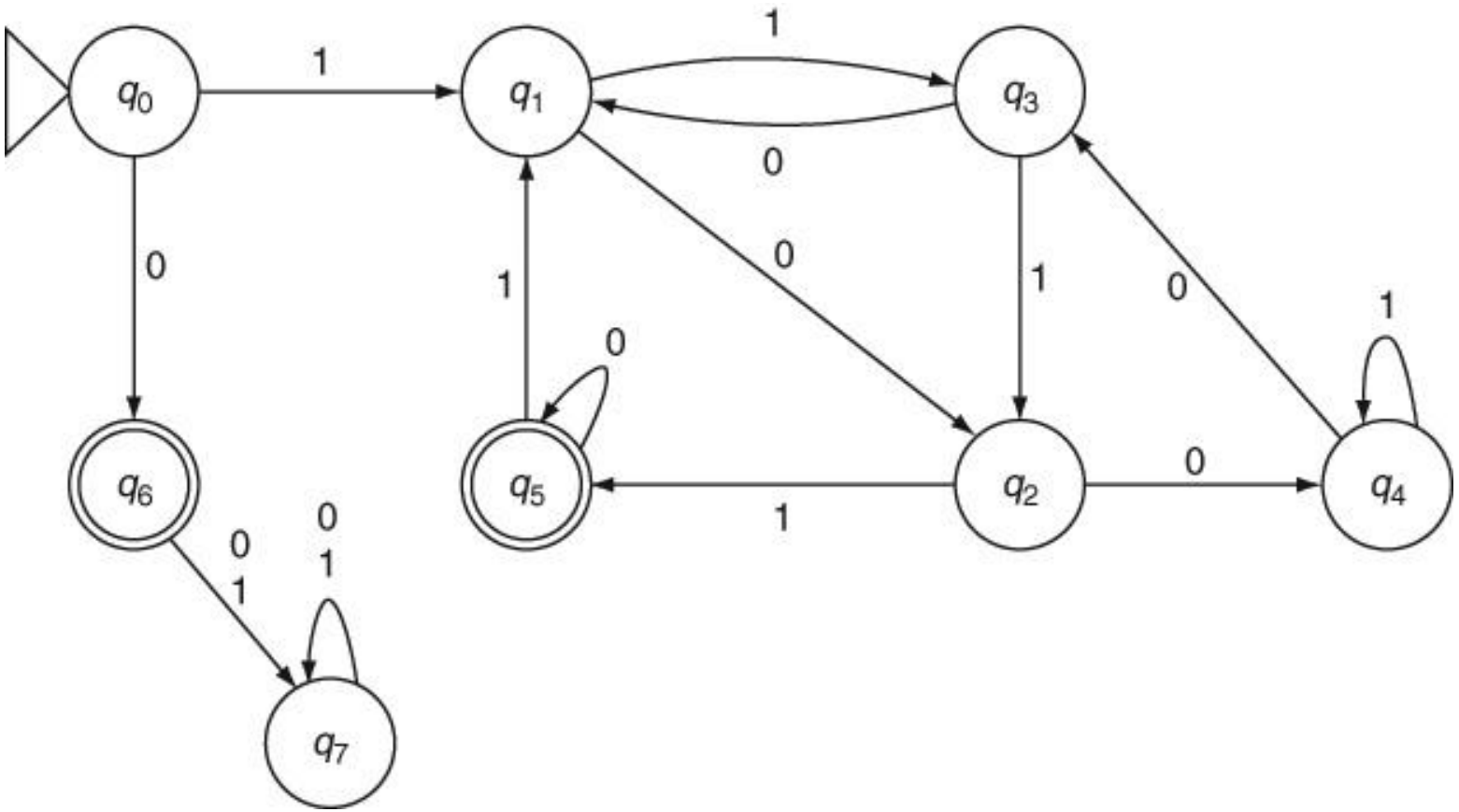
TRANSITION TABLE: EXAMPLE 2.7

TABLE 2.2 Transitions Among States for Division by 3

From State	Next Symbol	New State and Remainder
$q_0(n \bmod 3 = 0)$	0	$q_0((2n) \bmod 3 = 0)$
$q_0(n \bmod 3 = 0)$	1	$q_1((2n + 1) \bmod 3 = 1)$
$q_1(n \bmod 3 = 1)$	0	$q_2((2n) \bmod 3 = 2)$
$q_1(n \bmod 3 = 1)$	1	$q_0((2n + 1) \bmod 3 = 0)$
$q_2(n \bmod 3 = 2)$	0	$q_1((2n) \bmod 3 = 1)$
$q_2(n \bmod 3 = 2)$	1	$q_2((2n + 1) \bmod 3 = 2)$

EXAMPLE 2.8

- Binary numbers without leading zeros that are divisible by 5



TRANSITION TABLE: EXAMPLE 2.8

TABLE 2.3 Transitions Among States for Division by 5

From State	Next Symbol	New State and Remainder
$q_0(\lambda)$	0	$q_6(0 \bmod 5 = 0)$
$q_0(\lambda)$	1	$q_1(1 \bmod 5 = 1)$
$q_1(n \bmod 5 = 1)$	0	$q_2((2n) \bmod 5 = 2)$
$q_1(n \bmod 5 = 1)$	1	$q_3((2n + 1) \bmod 5 = 3)$
$q_2(n \bmod 5 = 2)$	0	$q_4((2n) \bmod 5 = 4)$
$q_2(n \bmod 5 = 2)$	1	$q_5((2n + 1) \bmod 5 = 0)$
$q_3(n \bmod 5 = 3)$	0	$q_1((2n) \bmod 5 = 1)$
$q_3(n \bmod 5 = 3)$	1	$q_2((2n + 1) \bmod 5 = 2)$
$q_4(n \bmod 5 = 4)$	0	$q_3((2n) \bmod 5 = 3)$
$q_4(n \bmod 5 = 4)$	1	$q_4((2n + 1) \bmod 5 = 4)$
$q_5(n \bmod 5 = 0)$	0	$q_5((2n) \bmod 5 = 0)$
$q_5(n \bmod 5 = 0)$	1	$q_1((2n + 1) \bmod 5 = 1)$

Mantras for Constructing DFA

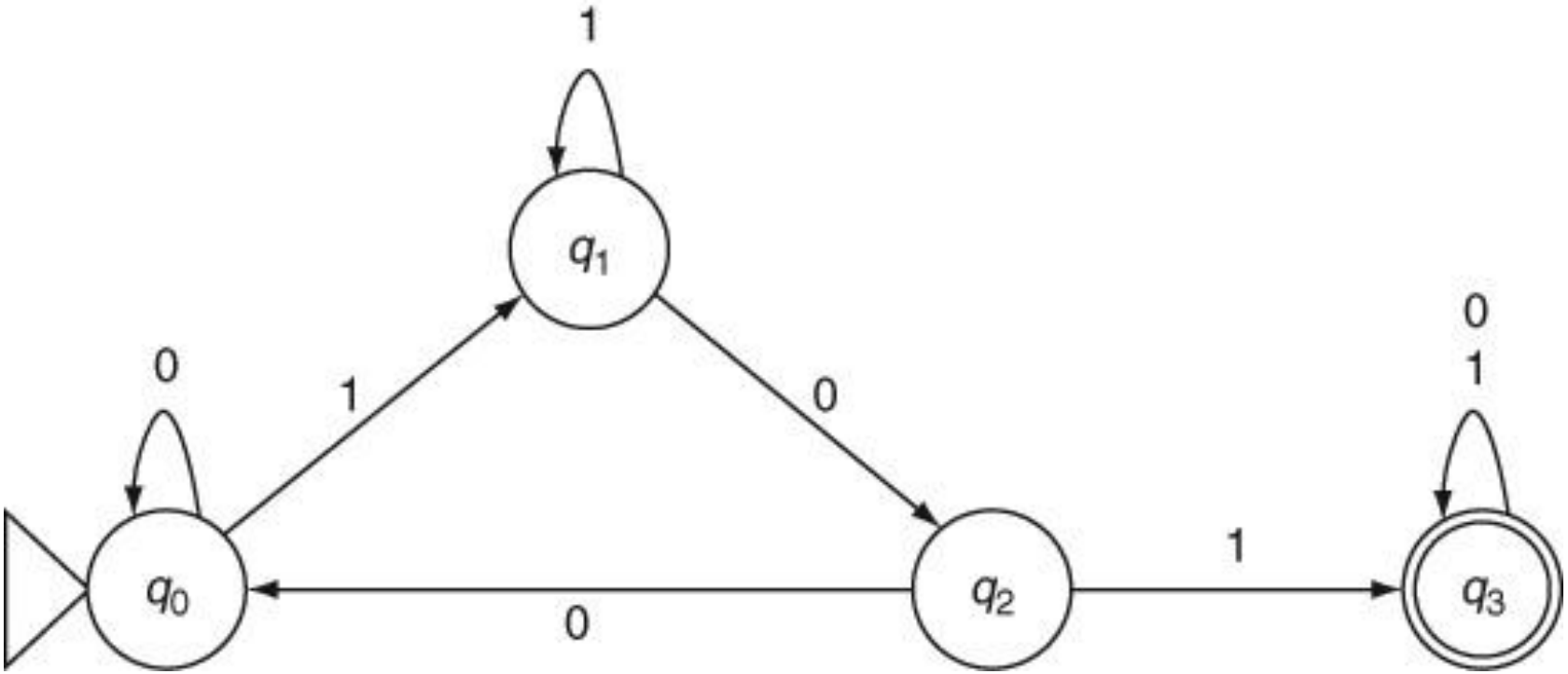
- What is the simplest string that the automaton is expected to accept?
- What are a few other strings that the automaton must accept?
- What are all the other inputs that the automaton must accept?
- What does the machine need to remember?
- When the automaton needs to remember an input symbol consumed in a given state, is the transition that we have introduced from that state for that symbol appropriate?

MANTRAS (CONTD..)

- What is the meaning of each state?
- What are the input strings to be rejected by the automaton?
- Does the automaton have exactly one transition from every state for every symbol in the alphabet?
- Are special cases such as leading zeroes (or other symbols), trailing symbols (e.g., extra zeros after reaching the final state) and the null input λ handled properly?
- Is the start state also an accepting state?

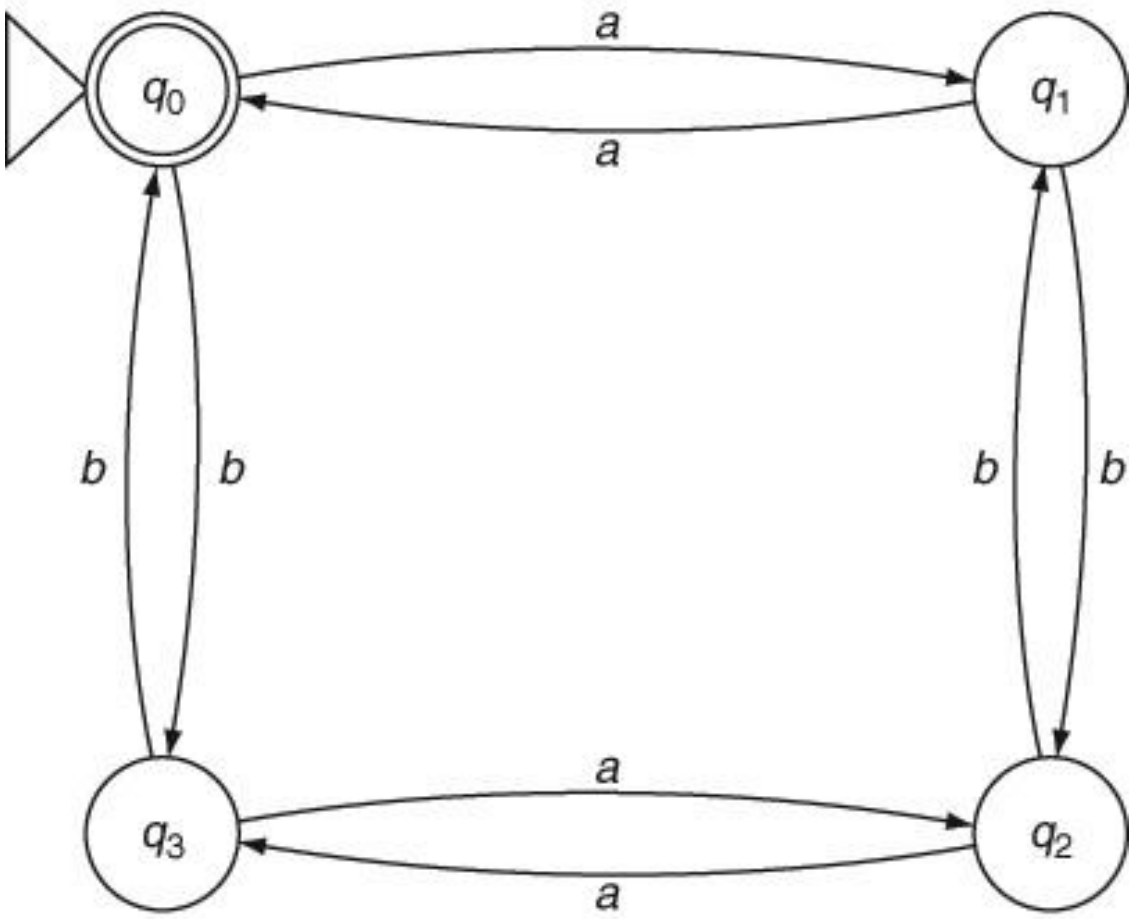
EXAMPLE 2.9

- To search for the keyword 101



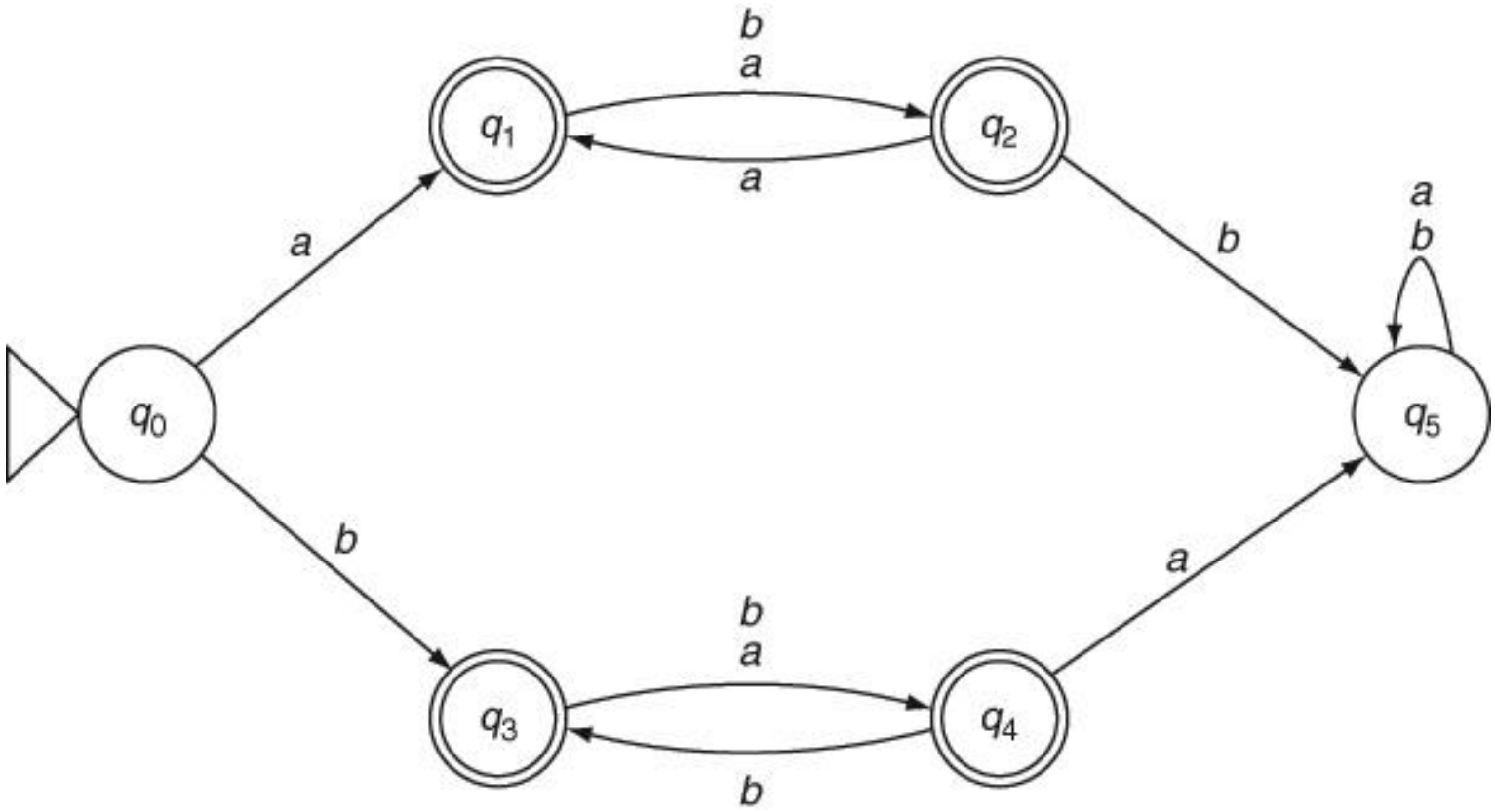
EXAMPLE 2.10

- Even number of *a* s and even number of *b* s



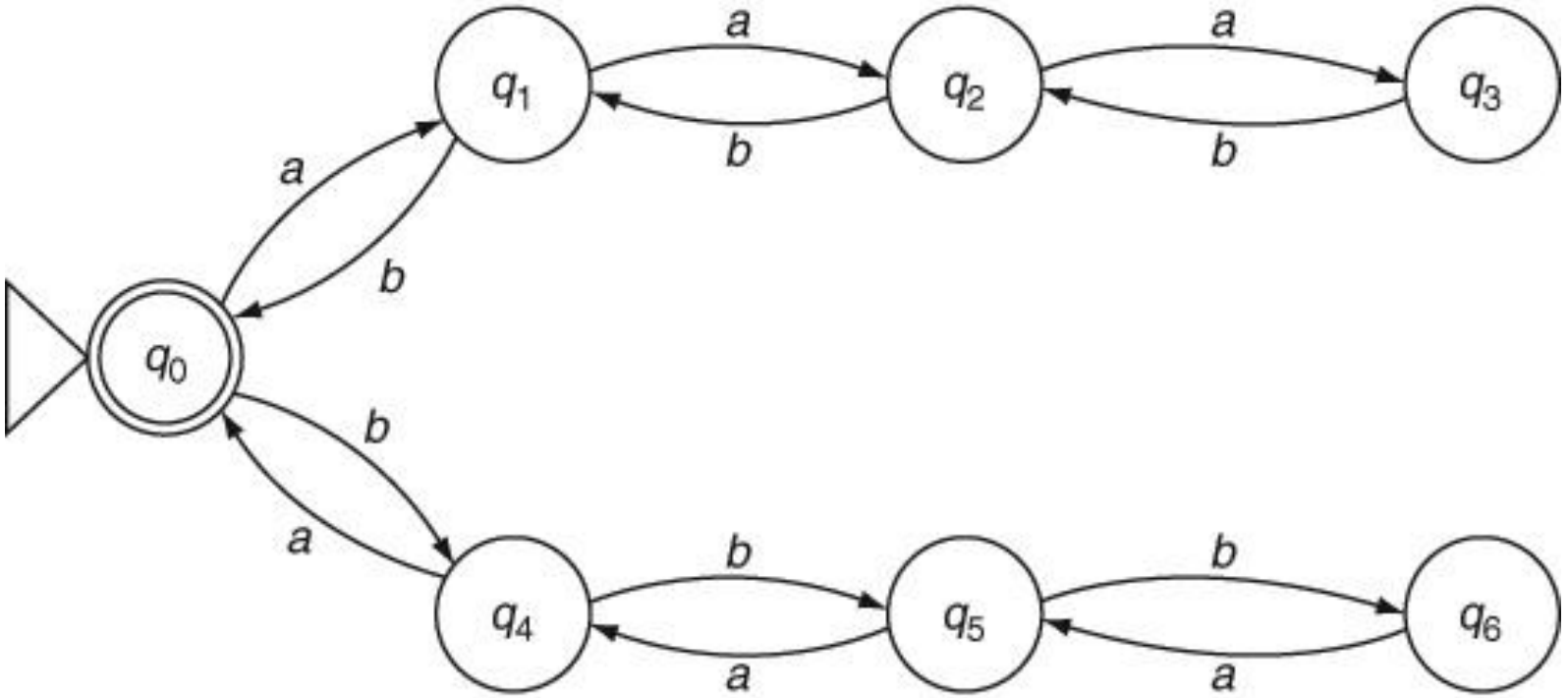
EXAMPLE 2.11

- Same symbols in odd positions



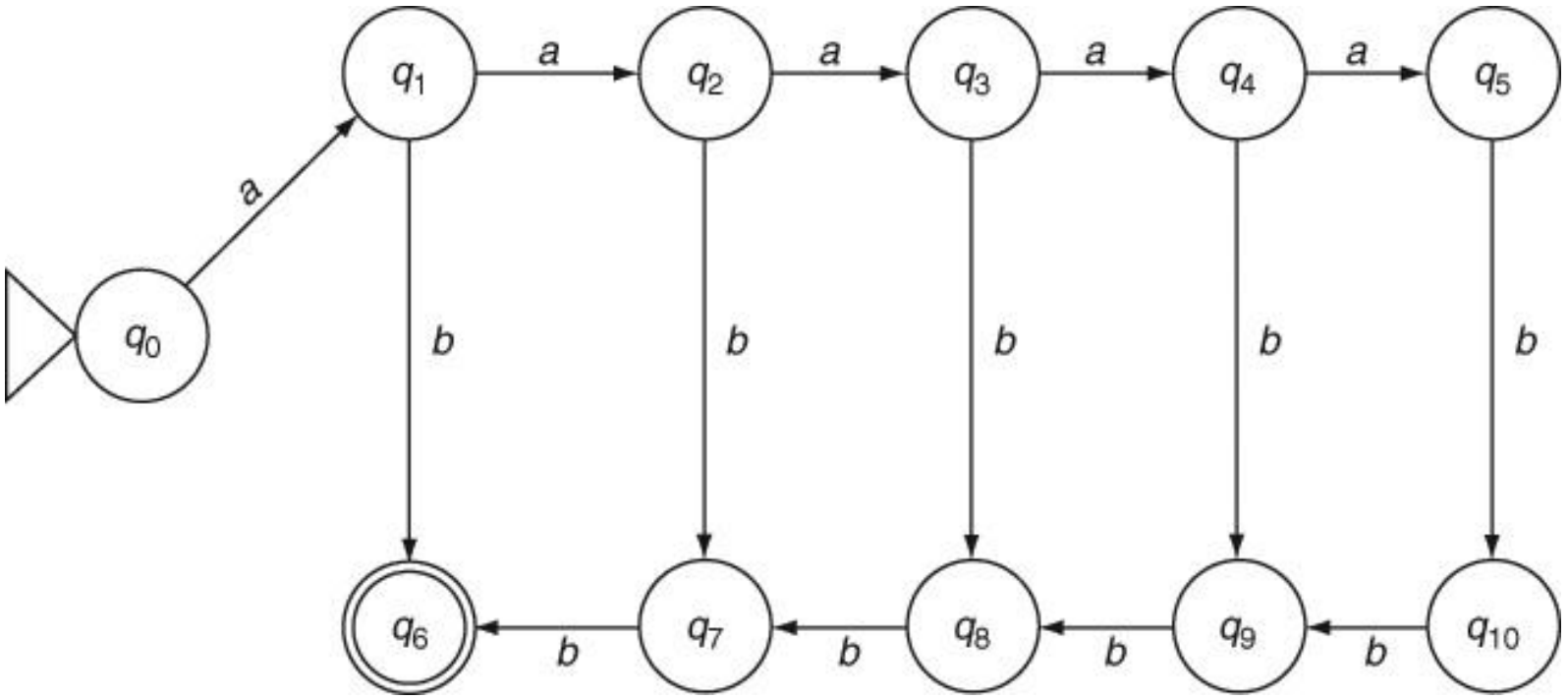
Limitations of Automata: Example 2.12

- Equal numbers of *a* s and *b* s??



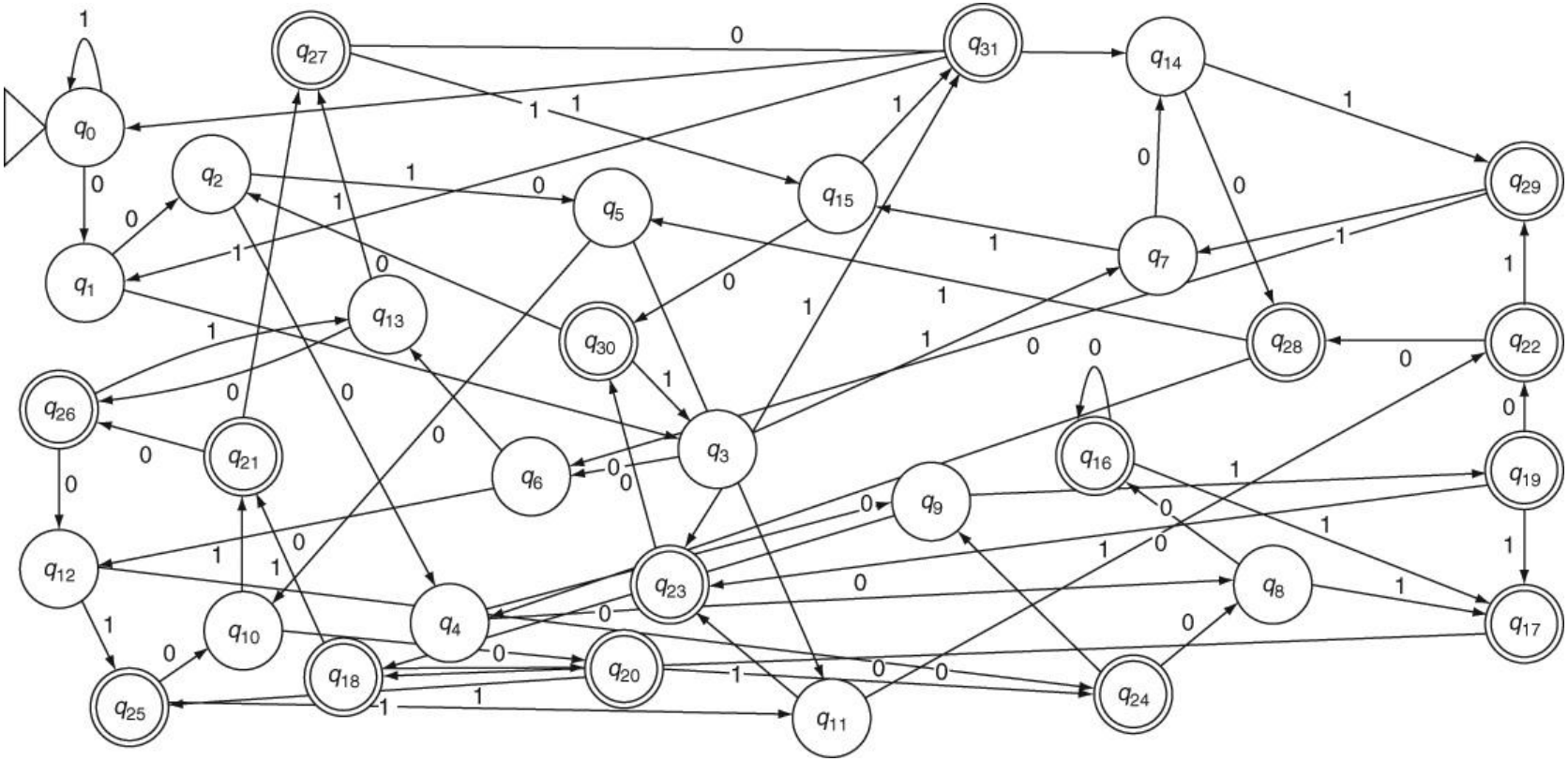
Limitations of Automata: Example 2.13

- $a^n b^n$, $n < 6$



Combinatorial States: Example 2.14

- Fifth symbol from the end is a 0



Key Ideas

- A Computing Machine accepts some input and produces some output by processing the input, usually making some decisions during its processing.
- Automata are the simplest kinds of computing machines. They are abstract models of computing usually represented as state transition diagrams.
- Automata are defined in terms of their states and state transitions. They accept an input string made up of symbols taken from an alphabet and either accept or reject the input by reaching an appropriate state at the end of processing the input.
- States are the only memory that automata have. They need to change states to remember anything.
- Automata are used to model hardware and software solutions to computing problems. Since real solutions can only have a finite amount of hardware or a finite number of lines of code in a software program, automata must also be finite. The number of states in an automaton must be finite.

KEY IDEAS (CONTD..)

- Deterministic finite automata never encounter multiple choices. For a given input symbol, when the automaton is in a given state, there is never more than one possible transition.
- Although there is no algorithm to design an automaton, the set of mantras discussed in the chapter help us in designing an accurate finite automaton for a given problem.
- Automata, being primitive computing machines, are limited in the variety of problems they can solve. A key limitation is that they are unable to count a set of symbols in the input string. As such, they cannot solve some classes of problems without requiring an infinite number of states.
- Even when they can solve a problem, deterministic finite automata often require a large number of states.

End of Chapter 2