1- Introduction

Course Objectives:

- Explore the theoretical foundations of computer science from the perspective of formal languages and classify machines by their power to recognize languages.
- Introduce students to the mathematical foundations of computation including automata theory; the theory of formal languages and grammars; the notions of algorithm, decidability, complexity, and computability.

Course Outcomes: After completion of the course students will be able to

- Demonstrate the conversion of NFA into DFA, ε-NFA into DFA and Minimization of Finite Automata by using Myhill-Nerode Theorem
- Formulate DFA, RE and FA with output.
- Design CFG and check the language is not CFL.
- Design PDA and convert n-PDA into d-PDA.
- Design Turing machines for addition, subtraction, multiplication etc.
- Formulate finite state machines, push down automata and Turing machines for automated functioning of devices.

Pre-requisites:

Experience in any programming language.

Basics of Discrete Mathematical Structures.

1.1 Automata Theory: Automata Theory is the study of abstract computing devices or machines.

Abstraction: The process of taking away or removing characteristics from something in order to reduce it to a set of essential characteristics to focus on details of greater importance; it is similar in nature to the process of *generalization*.

The process of abstraction can also be referred to as *modelling*.

In philosophical terminology, *abstraction* is the thought process wherein ideas are distanced from objects.

1.2 Evolution of Automata Theory:

- **1930:** A. Turing studied an abstract machine that had all the capabilities of today's computer.
  - Turing's goal was to describe precisely the boundary between what a computing machine could do and what it could not do.
- **1940's and 1950's:** Simpler kinds of machines called "finite automata" were studied by researchers.
  - Proposed to model brain function.
  - Later turned out to be extremely useful for variety of other purposes.
- **Late 1950's:** Linguist N. Chomsky begun the study of formal "grammars".
  - Serve today as the basis for some important software components like parts of compilers.
- **1969:** S. Cook extended Turing's study of what could and what couldn't be computed. Cook was able to separate
  - Problems that can be solved efficiently by computer.
  - Problems that can in principle be solved but in practice take too much time. (Intractable or NP-hard problems)

All of these theoretical developments bear directly on what computer scientists do today.

- **Finite Automata & Formal Grammars:** Used in the design and construction of important kinds of software.
- **Turing Machines:** Help us understand what we can expect from our software.

1.3 Applications of Automata Theory:

Each model in automata theory plays important roles in several applied areas. Finite automata are **used in text processing, compilers, and hardware design**. Context-free grammar (CFGs) are used in programming languages and artificial intelligence.

1.3.1 Finite Automata (FA) –

- To design the lexical analysis phase of a compiler.
- For recognizing the pattern using regular expressions.
- To design the combinational and sequential circuits using Mealy and Moore Machines.
- Used in text editors.
- For the implementation of spell checkers.

1.3.2 Push Down Automata (PDA) –

- To design the Syntax Analysis phase of a compiler (Parser).
- For the implementation of stack applications.
- For evaluating the arithmetic expressions.
- For solving the Tower of Hanoi Problem.

1.3.3 Linear Bounded Automata (LBA) –

- In genetic programming.
- For constructing syntactic parse trees during the compilation.

1.3.4 Turing Machine (TM) –

- For solving recursively enumerable problems.
- For understanding complexity theory.
- For implementation of neural networks.
- For implementation of Robotics Applications.
- In artificial intelligence (Natural Language Processing).

1.4 Central concepts of automata theory

In this section the definitions of the terms used throughout the automata theory are introduced. They are:

1) **Alphabet:** An *alphabet* is a finite nonempty set of symbols. It is denoted by Σ

   Examples:

   Binary alphabet: $\Sigma = \{0, 1\}$

   English alphabet: $\Sigma = \{a...z, A...Z\}$

   Alphabet for the programming language 'C': Σ = {a...z, A...Z, 0...9, +,-,*, /, %,<,>, &, | …etc.}

2) **Strings:** A *string* is a sequence of zero or more symbols chosen from a particular alphabet.

   E.g., **001100** is a binary string.

3) **Empty string:** String with zero occurrences of symbols. Empty string is denoted by ε.

4) **Length of a string:** Number of positions in the string. If 'w' is any string its length is denoted by $|w|$. e.g., w= 01011101 $|w| = 8$.

   (It is not the number of symbols in the string, for example in the above string there are two symbols '0' and '1', but the length is 8)

5) **Powers of an alphabet:** if Σ is an alphabet then Σ^k is the set of all strings of length 'k'.

   If $\Sigma = \{0, 1\}$ then $\Sigma^0 = \{\varepsilon\}$,

   $\Sigma^1 = \{0, 1\}$

   $\Sigma^2 = \{00, 01, 10, 11\}$

   $\Sigma^3 = \{000, 001, 010, 011, 100, 101, 110, 111\}$

   Set of all the strings over an alphabet is denoted Σ^*

   $$\Sigma^* = \Sigma^0 \cup \Sigma^1 \cup \Sigma^2 \cup \Sigma^3 \cup \dots..$$

   If empty string ε is excluded then the set of all nonempty strings is denoted by Σ^+

   $$\Sigma^+ = \Sigma^1 \cup \Sigma^2 \cup \Sigma^3 \cup \dots..$$

$$\Sigma^* = \Sigma^+ \cup \{\varepsilon\}$$

6) **Concatenation of strings:** If ‘x’ and ‘y’ are two strings then ‘xy’ denotes the concatenation of ‘x’ and ‘y’.

   Example: Let x=11011 and y=00110 then xy=1101100110 and yx=0011011011.

7) **Language:** Set of strings all of which are chosen from some Σ^* is called a language.

   If L is a language on some alphabet Σ then $L \subseteq \Sigma^*$.

Finite Automaton (FA) or Finite state machine (FSM) or finite-state automaton (FSA, plural: automata)

State: The purpose of a state is to remember the relevant portion of the system’s history.

Finite: The machine has only a finite number of states, so that it can be implemented with a fixed set of resources.

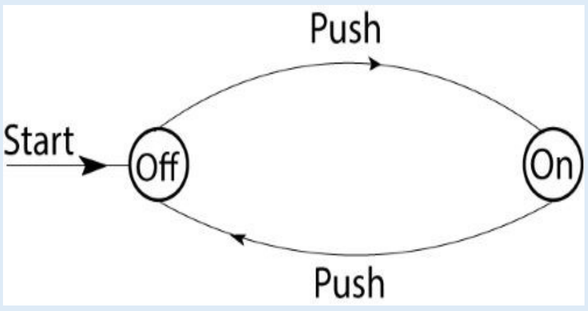
- It is an abstract machine.
- It can be in exactly one of a finite number of *states* at any given time.
- The FSM can change from one state to another in response to some inputs.
- The change from one state to another is called a *transition.*
- An FSM is defined by a list of its states, its *initial state*, and the *inputs that trigger each transition*.

Examples:

- Vending machines, which dispense products when the proper combination of coins is deposited.
- Elevators, whose sequence of stops is determined by the floors requested by riders.
- Traffic lights, which change sequence when cars are waiting.
- Combination locks, which require the input of a sequence of numbers in the proper order.

Finite Automaton is represented by a directed graph.

- ➢ The states are represented by circles.
- ➢ Arcs (edges) between the states are labelled by “inputs” which represent external influences on the system.
- ➢ One of the states is designated the “start state”, the state in which the system is placed initially.
- ➢ It is necessary to indicate one or more states as “final” or “accepting” states. Entering one of these states after a sequence of inputs indicates that the input sequence is good in some sense.



1.5 Deterministic Finite Automata (DFA):

Deterministic: On each input there is one and only one state to which the automaton can transition from its current state.

Definition: A DFA is represented formally by a **5-tuple** $A = (Q, \Sigma, \delta, q_0, F)$ where:

- ❖ Q is the finite nonempty set of states.
- ❖ Σ is the finite set of input symbols.
- ❖ δ is the transition function that takes as arguments the current state and input symbol and returns a state.

  $\delta(q, a) = p$, ‘q’ is the current state, ‘a’ is an input symbol and ‘p’ is the next state.
- ❖ q_0 is the start or initial state. $q_0 \in Q$
- ❖ F is the set of final or accepting states. $F \subseteq Q$

1.5.1 Processing of strings by a DFA:

How a DFA decides whether or not to "accept" a sequence of input symbols. The "language" of the DFA is the set of all the strings that the DFA accepts. Suppose **$w=a_1a_2...a_n$**, is a sequence of input symbols i.e., 'w' is a string of length 'n'.

- The DFA starts in state q_0.
- Consult the transition function **δ**, say $\delta(q_0, a_1) = q_1$, to find the state that the DFA enters after processing the first input symbol 'a_1' in state 'q_0'.
- Process the next input symbol, a_2, by evaluating $\delta(q_1, a_2)$; let this state is 'q_2'.
- We continue in this manner, finding the states q_3, $q_4...q_n$, such that $\delta(q_{i-1}, a_i) = q_i$ for each 'i'.
- $\delta(q_{n-1}, a_n) = q_n$ is the state of the DFA after reading the last symbol.
- If q_n is a member of F, i.e., $q_n \in F$, then the sequence **$a_1a_2...a_n$**, is "accepted", if not then it is "rejected".

Example 1: DFA that accepts all strings on the alphabet Σ = {a, b} that have the sequence **ab** somewhere in the string.

i.e., DFA 'A' to represent the language L= {w|w is of the form x**ab**y for some strings x and y.

What we know about the automaton to accept the language L?

- Alphabet is Σ = {a, b}.
- It has some set of states Q, of which q_0 is the start state.
- This automaton has to remember the important facts about what inputs it has seen so far? The automaton 'A' needs to remember

Facts	State
Has it already seen **ab**? If so it accepts every sequence of further inputs; i.e., it will only be in **accepting state** from now on.	q_2
It has never seen ab, but its last input was **a**, so if it sees a **b** now, it can accept everything it sees from here on.	q_1
It has just started **or** it has seen only sequence of b's.	q_0

Table 1.1

These three conditions can each be represented by a state. Condition 3 is represented by the start state q_0. Condition 2 is represented by the state q_1. Condition 1 is represented by the state q_2.

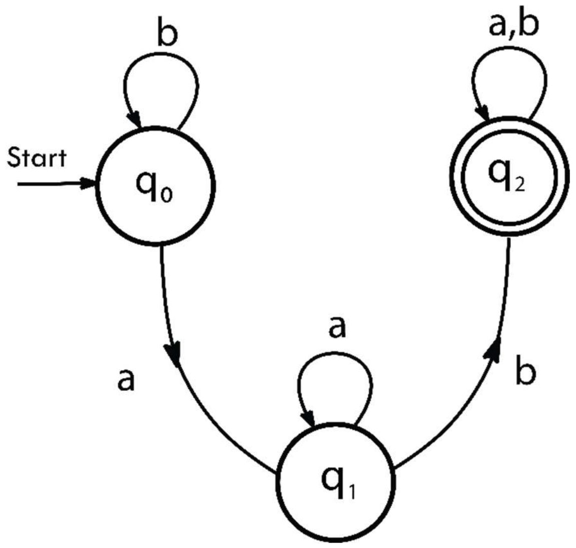


Figure 1.1

The DFA A is described as $A = (Q, \Sigma, \delta, q_0, F)$ where:

$Q = \{q_0, q_1, q_2\}$, $\Sigma = \{a, b\}$, q_0 is the start | initial state, $F = \{q_2\}$.

The transition function δ is defined as follows:

$\delta(q_0, a) = q_1$, $\delta(q_0, b) = q_0$

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

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

1.5.2 Notations for DFA's:

There are two notations for describing the automata.

1. A *transition diagram*, which is a directed graph.
2. A *transition table*, which is a tabular listing of the function δ.

Transition diagram (State diagram):

A *transition diagram* for a DFA $A = (Q, \Sigma, \delta, q_0, F)$ is a directed graph defined as follows.

a) For each state 'q' in Q there is a node.
b) For each state 'q' in Q there is a node and for each input symbol 'a' in Σ, let δ (q, a) =p. Then the transition diagram has an arc from node 'q' to node 'p', labelled 'a'. If there are several input symbols causing transitions from 'q' to 'p', then the transition can have one arc, labelled by the list of these symbols.
c) There is an arrow into the start state q_0, labelled *Start*. This arrow does not originate at any node.
d) Nodes corresponding to the accepting states (those in F) are marked by a double circle.

Transition Table:

A *transition table* is a tabular representation of the function δ.

a) The rows of the table correspond to states.
b) The columns of the table correspond to the inputs.
c) The entry for the row corresponding to the state 'q' and the column corresponding to input 'a', is the state δ (q, a).

Transition table for Example 1:

δ	a	b
→q_0	q_1	q_0
q_1	q_1	q_2
*q_2	q_2	q_2

Table 1.2

1.5.3 Extending the transition function to strings:

- The DFA defines a language: the set of all strings that result in a sequence of state transitions from the start state to an accepting state, i.e., the set of strings accepted by the DFA.
- Extended transition function
  - Describes what happens when we start in any state and follow a sequence of inputs.
  - If δ is our transition function, then the extended transition function is denoted by $\hat{\delta}$.
  - The extended transition function is a function that takes a state '*q*' and a string '*w*' as inputs and returns a state 'p' (the state that the automaton reaches when starting in state 'q' and processing the sequence of inputs in 'w').

Formal definition of the extended transition function:

Definition by induction on the length of the input string

Basis: $\hat{\delta}(q, \varepsilon) = q$ (string length is 0)

If we are in a state 'q' and read no inputs, then we are still in state 'q'.

Induction: Suppose '*w*' is a string of the form '*xa*'; that is '*a*' is the last symbol of '*w*', and '*x*' is the string consisting of all but the last symbol

Then: $\hat{\delta}(q, w) = \delta(\hat{\delta}(q, x), a)$.

- To compute $\hat{\delta}(q, w)$, first compute $\hat{\delta}(q, x)$, the state that the automaton is in after processing all but the last symbol of '*w*'.
- Suppose this state is p, i.e., $\hat{\delta}(q, x) = p$.
- Then $\hat{\delta}(q, w)$ is what we get by making a transition from state '*p*' on input '*a*' - the last symbol of '*w*'.

Construction of $\hat{\delta}$ to process the string baaabab, for DFA in example1.

$\hat{\delta}(q_0, \varepsilon) = q_0$

$\hat{\delta}(q_0, b) = \delta(\hat{\delta}(q_0, \varepsilon), b) = \delta(q_0, b) = q_0$

$\hat{\delta}(q_0, ba) = \delta(\hat{\delta}(q_0, b), a) = \delta(q_0, a) = q_1$

$\hat{\delta}(q_0, baa) = \delta(\hat{\delta}(q_0, ba), a) = \delta(q_1, a) = q_1$

$\hat{\delta}(q_0, baaa) = \delta(\hat{\delta}(q_0, baa), a) = \delta(q_1, a) = q_1$

$\hat{\delta}(q_0, baaab) = \delta(\hat{\delta}(q_0, baaa), b) = \delta(q_1, b) = q_2$

$\hat{\delta}(q_0, baaaba) = \delta(\hat{\delta}(q_0, baaab), a) = \delta(q_2, a) = q_2$

$\hat{\delta}(q_0, baaabab) = \delta(\hat{\delta}(q_0, baaaba), b) = \delta(q_2, b) = q_2$

$\because$ $\mathbf{q_2 \in F}$, the string **baaabab** is accepted by the DFA.

Example 2: DFA to accept the language

L = {w | w has both an even number of 0's and an even number of 1's}

i.e., all binary strings with even number of 0's and even number of 1's.

The states of this DFA are used to remember the number of 0's seen so far is even or odd and also the number of 1's seen so far is even or odd.

Thus there are four possibilities.

Facts		**State**
Number of 0's	**Number of 1's**	
Even	Even	q_0
Even	Odd	q_1
Odd	Even	q_2
Odd	Odd	q_3

Table 1.3

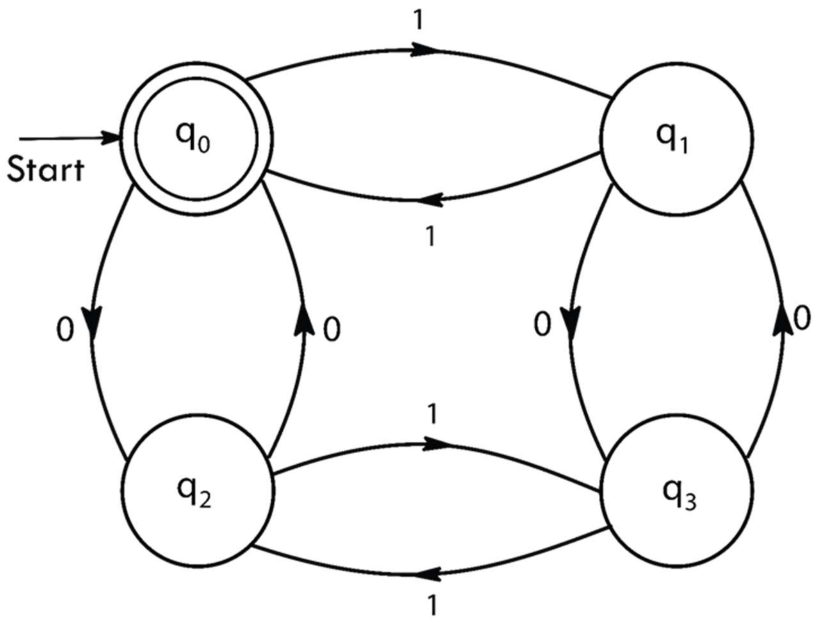


Figure 1.2

Components of the DFA

1. $Q = \{q_0, q_1, q_2, q_3\}$
2. $\Sigma = \{0, 1\}$
3. Transition function

δ	0	1
$\rightarrow q_0{*}$	q_2	q_1
q_1	q_3	q_0
q_2	q_0	q_3
q_3	q_1	q_2

4. Initial or start state is q_0
5. $F = \{q_0\}$

1.5.4 The Language of a DFA:

The language of a DFA $A = (Q, \Sigma, \delta, q_0, F)$, denoted $L(A)$ is defined by

$$L(A) = \{w \mid \hat{\delta}(q_0, w) \text{ is in } F\}$$

The language of A is the set of strings '*w*' that take the start state '*q_0*' to one of the accepting states. If L is L (A) from some DFA, then L is a *regular language* i.e., if L is defined by some DFA A, then it is a regular language.

Construction of $\hat{\delta}$ to process the string 100010, for DFA in example2.

$\hat{\delta}(q_0, \varepsilon) = q_0$

$\hat{\delta}(q_0, 1) = \delta(\hat{\delta}(q_0, \varepsilon), 1) = \delta(q_0, 1) = q_1$

$\hat{\delta}(q_0, 10) = \delta(\hat{\delta}(q_0, 1), 0) = \delta(q_1, 0) = q_3$

$\hat{\delta}(q_0, 100) = \delta(\hat{\delta}(q_0, 10), 0) = \delta(q_3, 1) = q_2$

$\hat{\delta}(q_0, 1000) = \delta(\hat{\delta}(q_0, 100), 0) = \delta(q_2, 1) = q_3$

$\hat{\delta}(q_0, 10001) = \delta(\hat{\delta}(q_0, 1000), 1) = \delta(q_3, 1) = q_2$

$\hat{\delta}(q_0, 100010) = \delta(\hat{\delta}(q_0, 10001), 0) = \delta(q_2, 0) = q_0$

$\because q_0 \in F$, the string 100010 is accepted by the DFA i.e., it is a valid string.

Example 3: DFA to accept strings on Σ = {0, 1} ending with 1.

Facts	State
Last input is 0.	A
Last input is 1.	B

Table 1.4

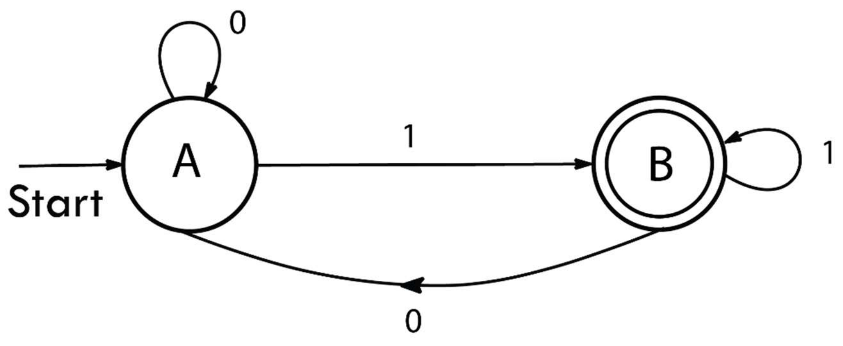


Figure 1.3

Components of the DFA

1. Q = {A, B}
2. Σ = {0, 1}
3. Transition function

δ	0	1
→A	A	B
B*	A	B

4. Initial or start state q_0 = **ee**
5. F = {B}

Example 4: DFA to accept strings on Σ = {0, 1} ending with 01.

Facts	State
The DFA has not seen any input or seen only a sequence of 1's.	q_0
Last input is 0.	q_1
The last two inputs are 0 followed by 1.	q_2

Table 1.5

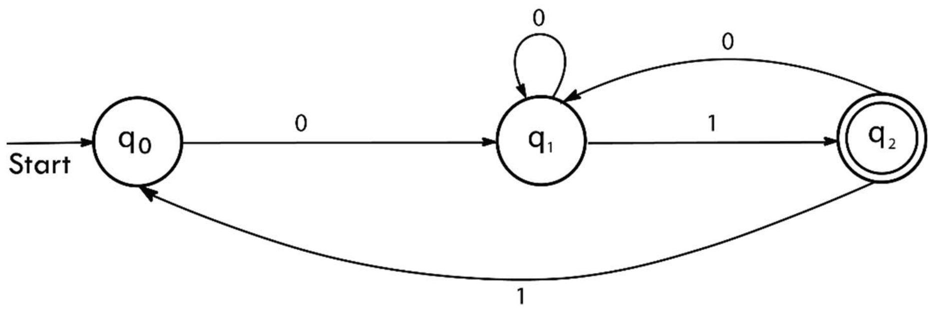


Figure 1.4

Components of the DFA

1. $Q = \{q_0, q_1, q_2\}$
2. $\Sigma = \{0, 1\}$
3. Transition function

δ	0	1
$\rightarrow q_0$	q_1	q_2
q_1	q_1	q_2
q_2*	q_1	q_0

4. Initial or start state q_0
5. F = {q2}

Example 5: DFA to accept strings on Σ = {0, 1} starting with 0.

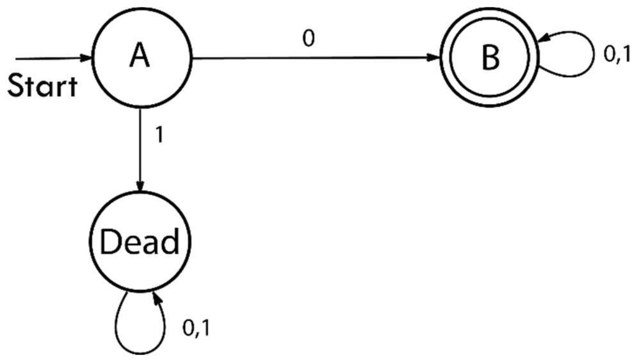


Figure 1.5

Components of the DFA

1. Q = {A, B, Dead}
2. Σ = {0, 1}
3. Transition function

δ	0	1
→A	B	Dead
B*	B	B
Dead	Dead	Dead

4. Initial or start state q_0
5. F = {q2}

Example 6: DFA to accept strings on Σ = {a, b} such that the second symbol is b.

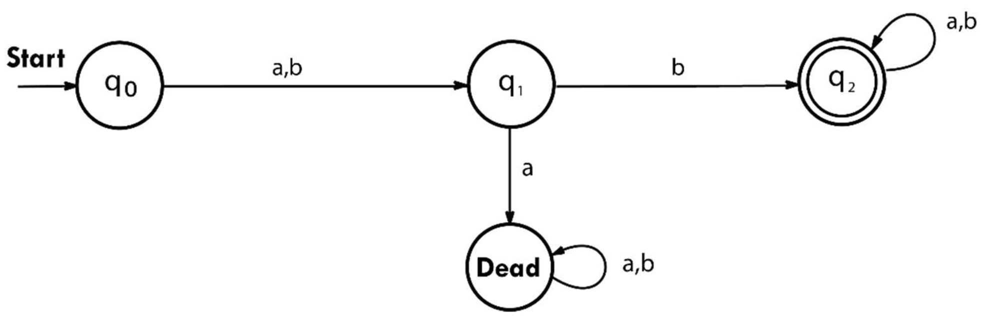


Figure 1.6

Components of the DFA

1. Q = {q_0, q_1, q_2, Dead}
2. Σ = {a, b}
3. Transition function

δ	a	b
→q_0	q_1	q_2
q_1	Dead	q_2
q_2*	q_2	q_2
Dead	Dead	Dead

4. Initial or start state q_0
5. F = {q2}

Example 7: DFA to accept strings on Σ = {a, b} such that the string contains at least one a.

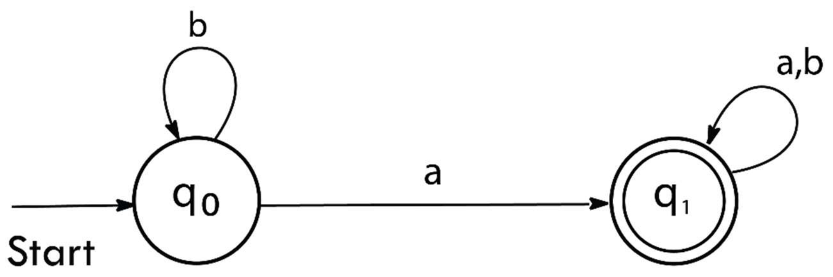


Figure 1.7

Example 8: DFA to accept strings on Σ = {a, b} such that the string contains exactly one a.

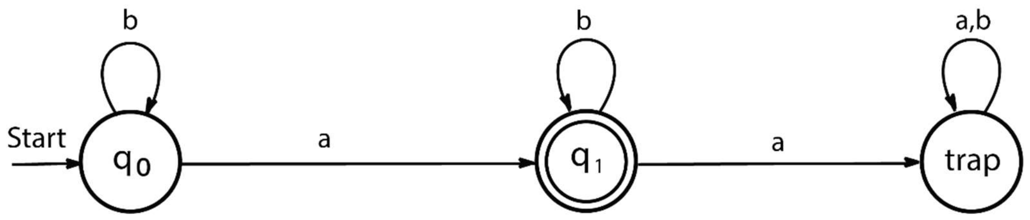


Figure 1.8

Components of the DFA

1. Q = $\{q_0, q_1, \text{trap}\}$
2. Σ = {a, b}
3. Transition function

δ	a	b
→q_0	q_1	q_0
q_1*	Trap	q_1
trap	trap	trap

4. Initial or start state q_0
5. F = $\{q_2\}$

Example 9: DFA to accept strings on Σ = {a, b} such that the string contains exactly two a's.

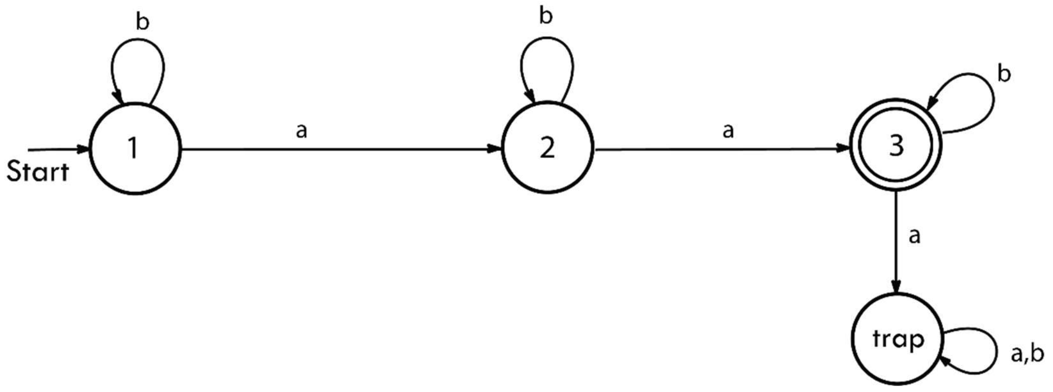


Figure 1.9

Example 10: DFA to accept strings on Σ = {a, b} such that the string contains at most three a's.

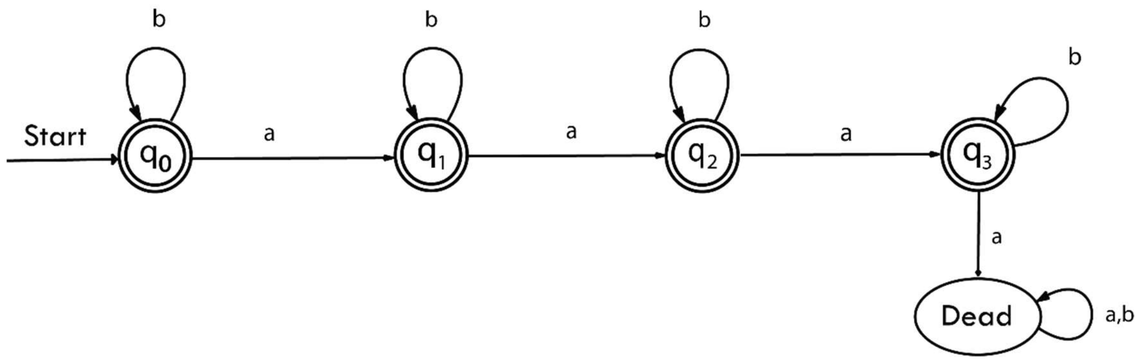


Figure 1.10

Components of the DFA

1. Q = {q_0, q_1, q_2, q_3, Dead}
2. Σ = {a, b}
3. Transition function

δ	a	b
→q_0*	q_1	q_0
q_1*	q_2	q_1
q_2*	q_3	q_2
q_3*	Dead	q3
Dead	Dead	Dead

4. Initial or start state q_0
5. F = { q_0, q_1, q_2, q_3}

Example 11: DFA to accept strings on Σ = {0, 1} such that the string contains the substring 00 at least once.

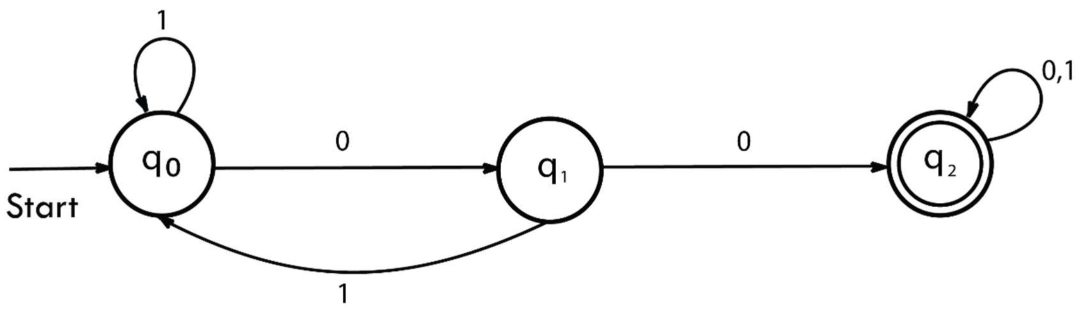


Figure 1.11

Example 12: DFA to accept strings on Σ = {0, 1} such that the string contains the substring 01 at least once.

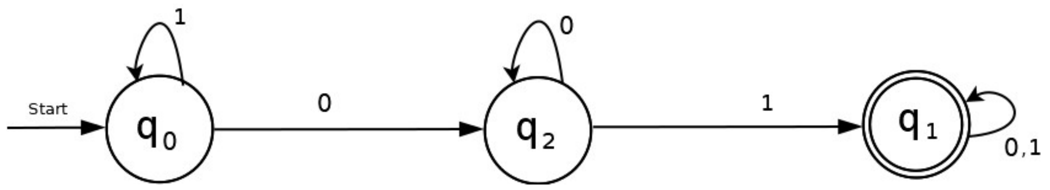


Figure 1.12

Example 13: DFA to accept strings on Σ = {0, 1} such that the string contains the substring 011 at least once.

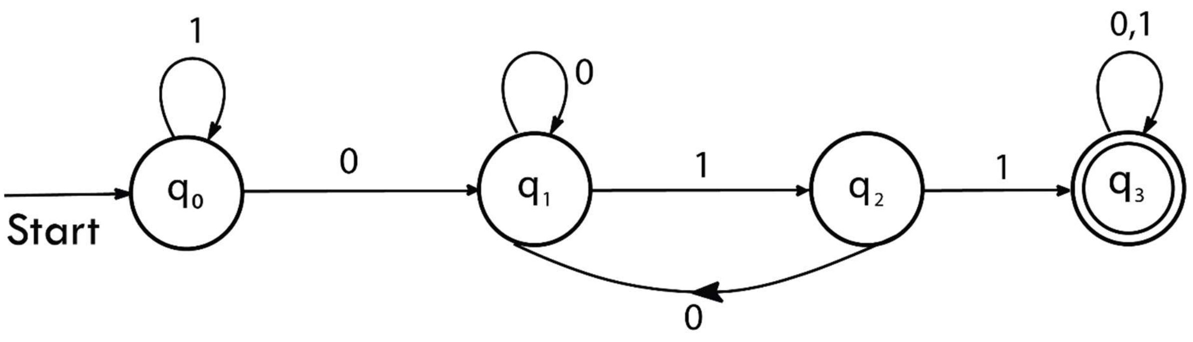


Figure 1.13

Example 14: Design a DFA to accept strings on Σ = {a, b} such that the string contains at least one a and at least one b.

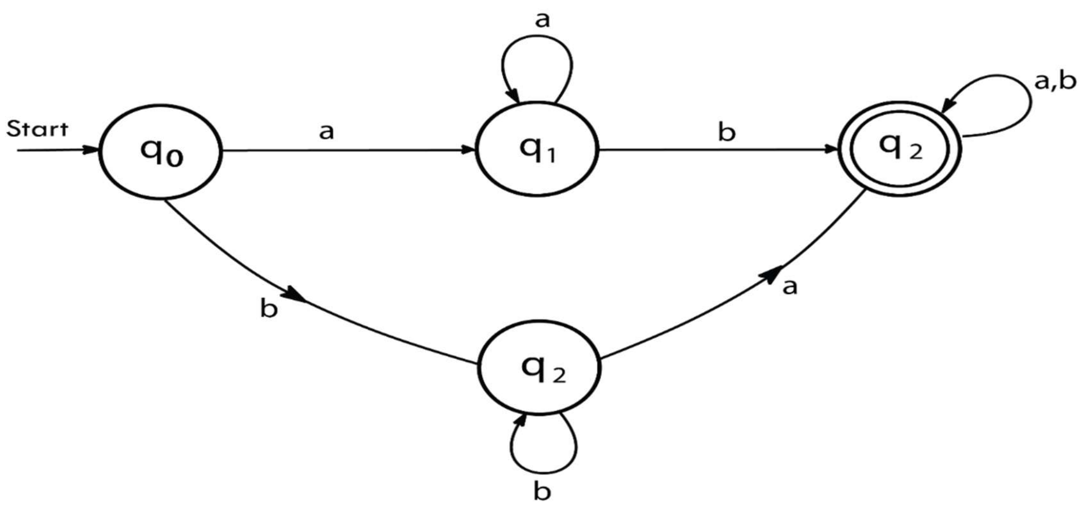


Figure 1.14

Example 15: DFA to accept strings on Σ = {a, b} such that the string contains at least one a or at least one b.

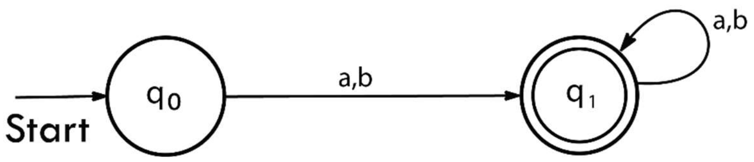


Figure 1.15

Example 16: DFA to represent the language L, where

L = {awa | w ε {a, b}*} i.e., the language L contains all strings with first, last symbol as a and these two a's separated by any string on Σ = {a, b}.

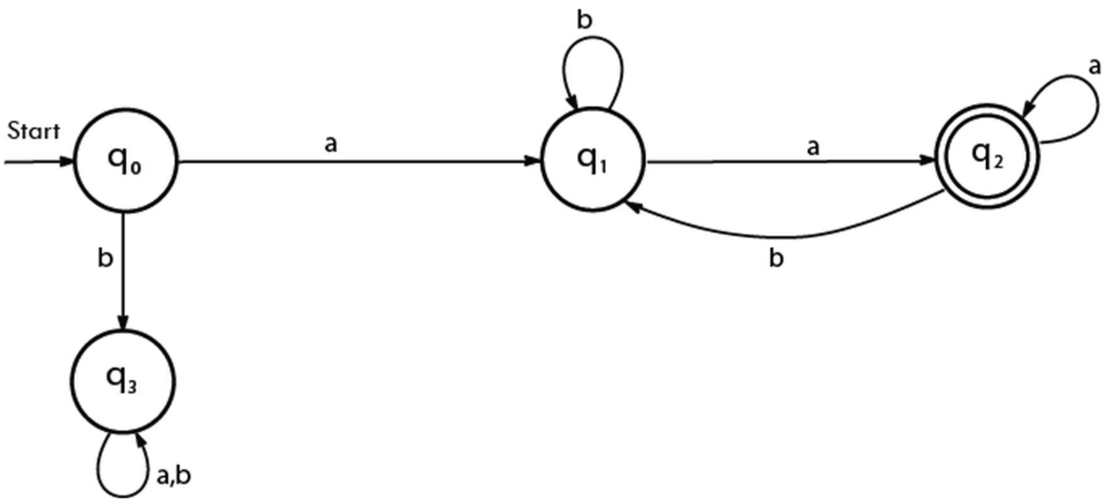


Figure 1.16

Example 17: DFA to represent the language L, where

L = {w | w ε {a, b}* and |w| <=2} i.e., the language L contains all strings on Σ = {a, b} with length less than or equal to two.

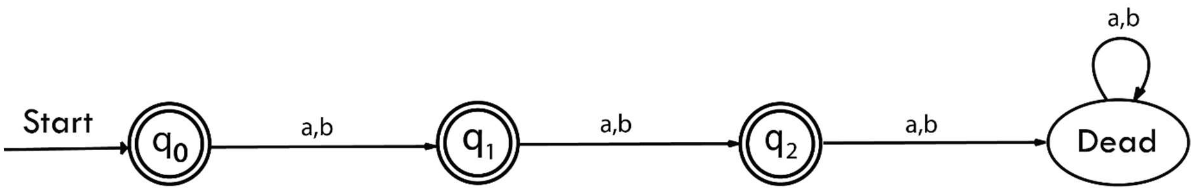


Figure 1.17

Components of the DFA

1. Q = {q_0, q_1, q_2, Dead}
2. Σ = {a, b}
3. Transition function

δ	a	b
→q_0*	q_1	q_1
q_1*	q_2	q_2
q_2*	Dead	Dead
Dead	Dead	Dead

4. Initial or start state q_0
5. F = { q_0, q_1, q_2,}

Example 18: DFA to represent the language L, where

L = {w | w ε {a, b}* and |w| >=2} i.e., the language L contains all strings on Σ = {a, b} with length greater than or equal to two.

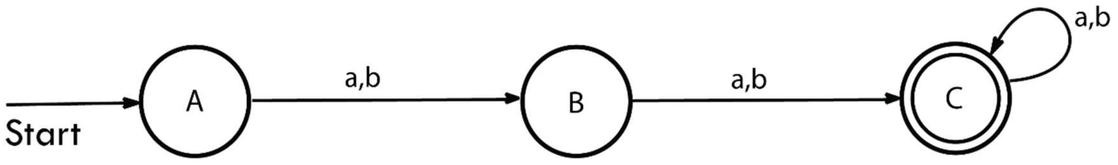


Figure 1.18

Example 19: DFA to represent the language L, where

L = {w | w ε {a, b}* and |w| mod 2 = 0} i.e., the language L contains all strings on Σ = {a, b} with even length.

Facts	State
String length divisible by 2	q_0
String length not divisible by 2	q_1

Table 1.6

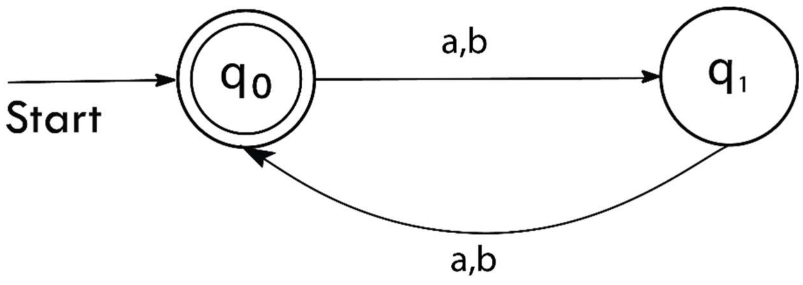


Figure 1.19

Example 20: DFA to represent the language L, where

L = {w | w ε {a, b}* and |w| mod 3 = 0}.

Facts	State
String length mod 3 = 0	q_0
String length mod 3 = 1	q_1
String length mod 3 = 2	q_2

Table 1.7

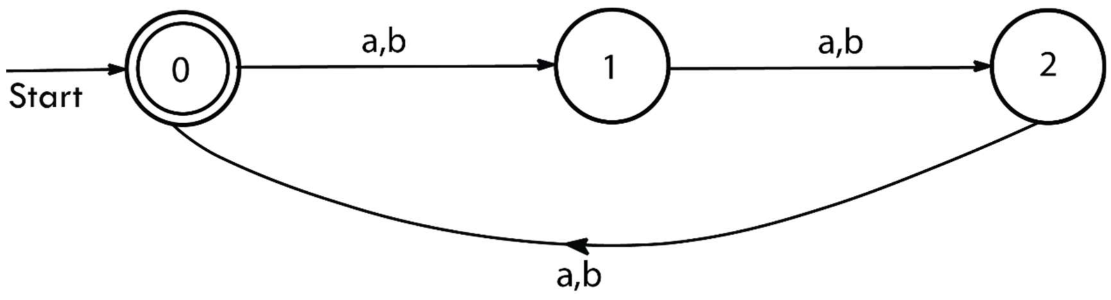


Figure 1.20

Example 21: DFA to represent the language L, where

L = {w | w ε {a, b}* and |w| mod 3 ≠ 0}

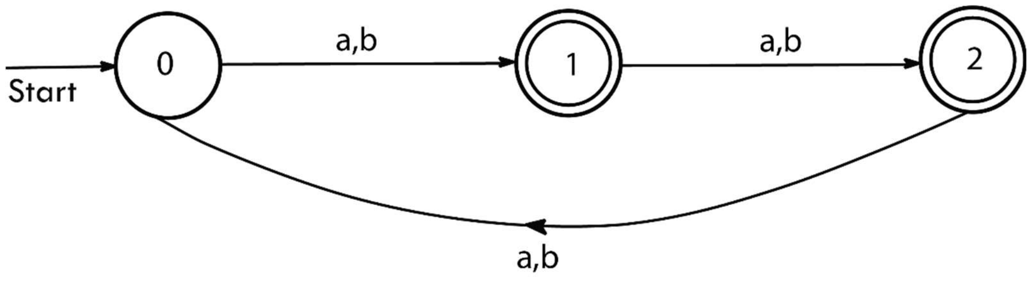


Figure 1.21

Example 22: DFA to represent the language L, where

L = {w | w ε {a, b}* and |w| mod 3 = 1}

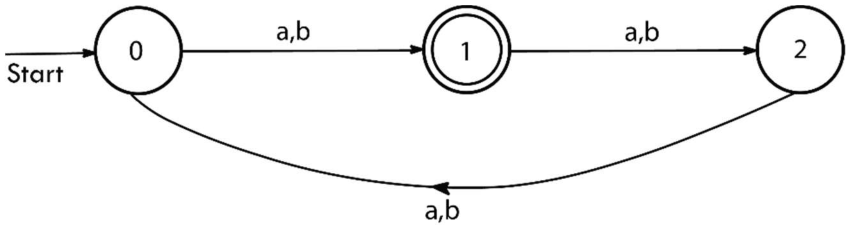


Figure 1.22

Example 23: DFA to represent the language L, where

L = {w | w ε {a, b}* and |w| mod 3 = 2}

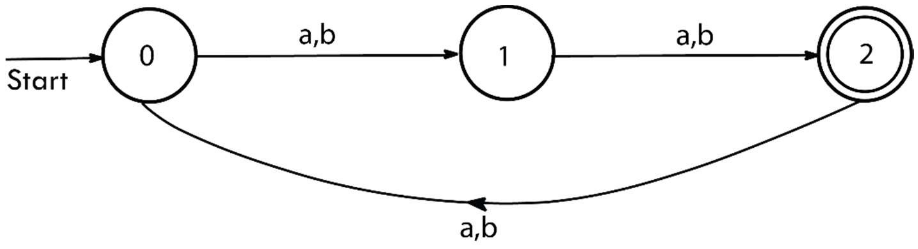


Figure 1.23

Example 24: DFA to represent the language L, where

L = {w | $n_a(w) \bmod 2 = 0$} i.e., number of a's in the string 'w' is even.

(n_a (w) – number of a's in the string w)

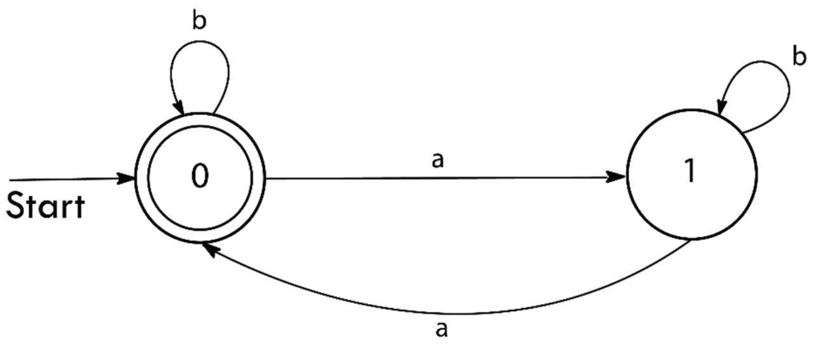


Figure 1.24

Example 25: DFA to represent the language L, where

L = {w | $n_b(w) \bmod 3 = 0$} i.e., number of b's in the string 'w' is divisible by 3.

Facts	State
Number of b's mod 3 = 0	0
Number of b's mod 3 = 1	1
Number of b's mod 3 = 2	2

Table 1.8

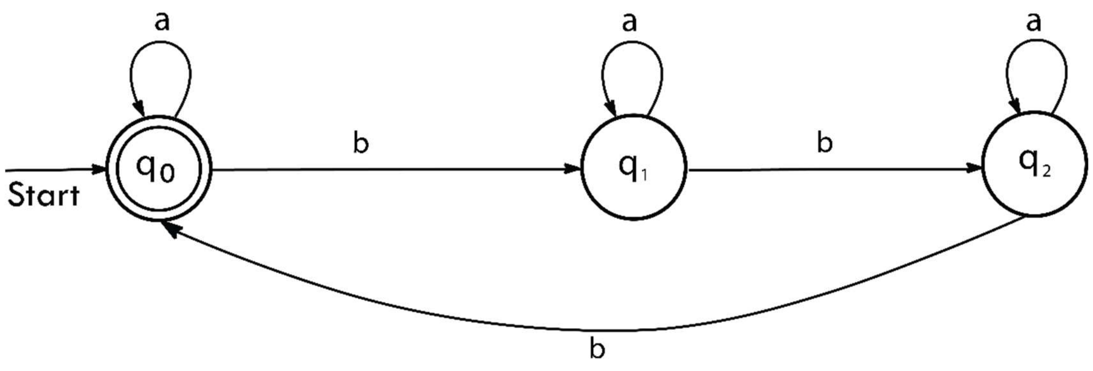


Figure 1.25

Example 26: Give DFA to accept all the binary strings, which when interpreted as binary numbers are divisible by 5, i.e., to represent the language

L = {w | w ∈ {0, 1}* and w is divisible by 5.}

Facts	State
w mod 5 = 0	q_0
w mod 5 = 1	q_1
w mod 5 = 2	q_2
w mod 5 = 3	q_3
w mod 5 = 4	q_4

Table 1.9

DFA (Transition table)

δ	0	1
→q_0*	q_0	q_1
q_1	q_2	q_3
q_2	q_4	q_0
q_3	q_1	q_2
q_4	q_3	q_4

Table 1.10

Example 26: Construct DFA to represent the strings on Σ= {a, b}, such that every block of 5 characters contains at least 2 a's.

Each state is represented by a pair of numbers (i, j) where

i is the number of characters read in a block. Possible values for i are 0,1,2,3,4,5

j is defined as follows.

$j = 0$, if no a's read in the block.

$j = 1$ if one a is read in the block.

$j = 2$ if number of a's read in the block >= 2.

Transitions are defined below.

$\delta((i, j), a) = (i+1, j+1)$ if $i <= 4$ and $j < 2$.

$\delta((i, j), a) = (i+1, j)$ if $i <= 4$ and $j = 2$.

$\delta((i, j), b) = (i+1, j)$ if $i <= 4$ and $j = 2$.

$$\left.\begin{array}{l} \delta((5, 2), a) = (1, 1). \\ \delta((5, 2), b) = (1, 0). \end{array}\right\} \quad \boxed{(5, 2) \text{ is the final state}}$$

(4, 0) and (5, 1) are the dead or trap states.

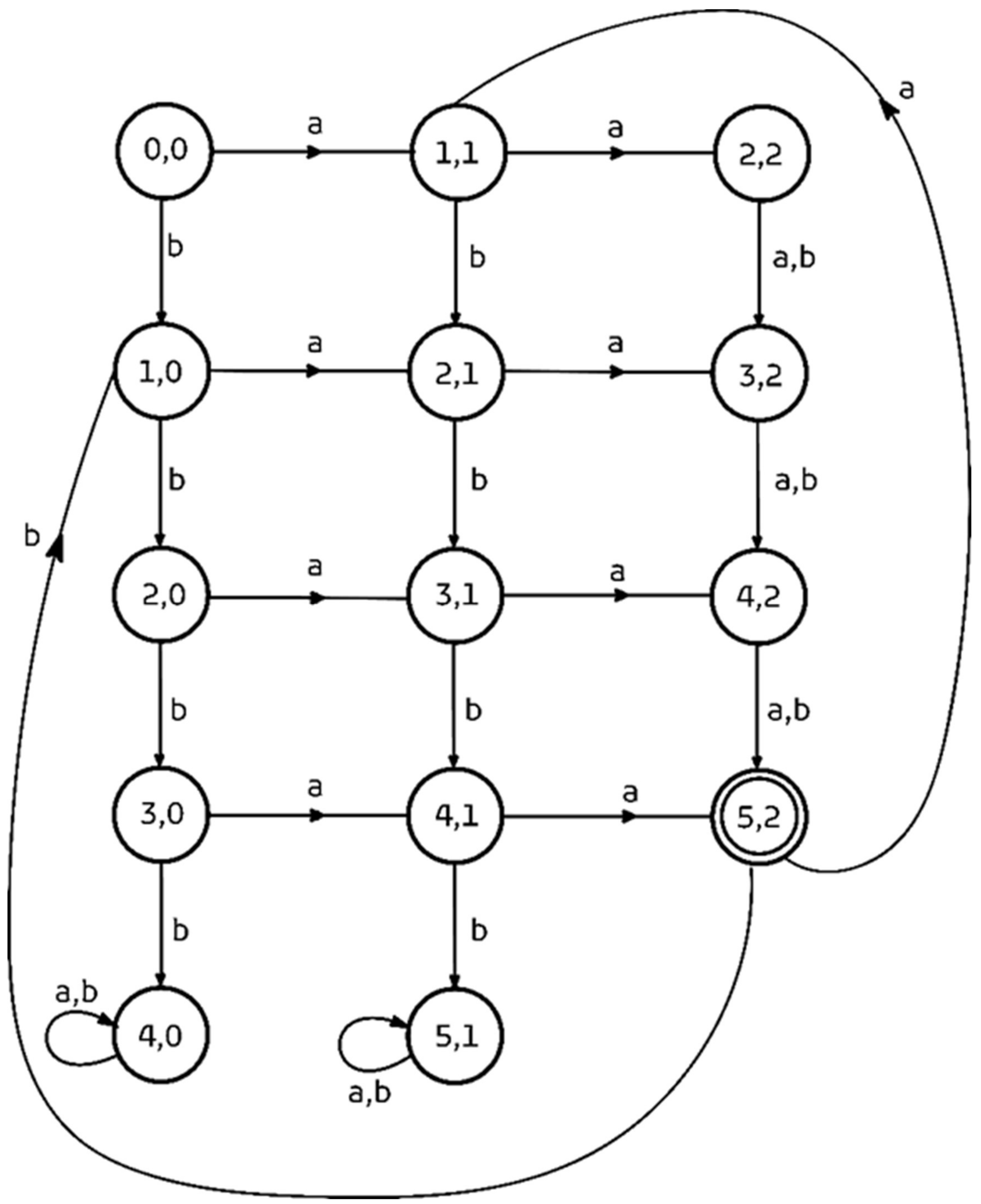


Figure 1.26