Assiut University
Faculty of Computer and Information
Department Computer Science

Deterministic Finite Automata (DFA)

Team Member:

1. Yousry Essam Ayoub
2. Magy Hamdy Waseily
3. Karin Essam El-Nagy

Table of Contents

1. Introduction:

- In the first, we will talk about the theory of automata briefly and DFA machines.

1.1 Theory of automata:

- **Definition:**
  - **Automata Theory** is theoretical branch of computer science. It established its roots during the 20th Century, as mathematicians began developing - both theoretically and literally - machines which imitated certain features of man, completing calculations more quickly and reliably.
  - The word **automaton** itself, closely related to the word "automation", denotes automatic processes carrying out the production of specific processes. Simply stated, automata theory deals with the logic of computation with respect to simple machines, referred to as **automata**. Through automata, computer scientists are able to understand how machines compute functions and solve problems and more importantly, what it means for a function to be defined as **computable** or for a question to be described as **decidable**.
  - **Automatons** are **abstract models** of machines that perform computations on an input by moving through a series of states or configurations. At each state of the computation, a transition function determines the next configuration on the basis of a finite portion of the present configuration. As a result, once the computation reaches an accepting configuration, it accepts that input. **The most general and powerful automata is the Turing machine.**
- **Objectives:**
  - The **major objective** of automata theory is to develop methods by which computer scientists can describe and analyze the dynamic behavior of discrete systems, in which signals are sampled periodically. The behavior of these discrete systems is determined by the way that the system is constructed from storage and combinational elements.
- **Characteristics:**
  - Characteristics of such machines include:

1. **Inputs:** assumed to be sequences of symbols selected from a finite set *I* of input signals. Namely, set *I* is the set $\{x_1, x_2, x_3 \ldots x_k\}$ where *k* is the number of inputs.
2. **Outputs:** sequences of symbols selected from a finite set Z. Namely, set *Z* is the set $\{y_1, y_2, y_3 \ldots y_m\}$ where *m* is the number of outputs.
3. **States:** finite set *Q*, whose definition depends on the type of automaton.

- **Types:**

  - There are **four major families of automaton** :

    1. Finite-state machine.
    2. Pushdown automata.
    3. Linear-bounded automata.
    4. Turing machine.

1.2 DFA Machines:

- DFA refers to deterministic finite automata. Deterministic refers to the uniqueness of the computation. The finite automata are called deterministic finite automata if the machine is read an input string one symbol at a time.
- In DFA, there is only one path for specific input from the current state to the next state.
- DFA does not accept the null move, i.e., the DFA cannot change state without any input character.
- DFA can contain multiple final states. It is used in Lexical Analysis in Compiler

2. Machine Design:

2.1 DFA Definition:

A deterministic finite automaton (DFA) is a 5-tuple (Q, Σ, δ, q0, F), where:

- **Q** is a finite set called **the states**,
- **Σ** is a finite set called **the alphabet**,
- **δ : Q × Σ → Q** is the **transition** function,
- **q0 ∈ Q** is the **start state**, and
- **F ⊆ Q** is the **set of accept states**.

2.2 Case Studies:

2.2.1 DFA Machine (1):

- DFA (1) is accepting any string that has even 0's and odd 1's.

Solution

By using formal definition:

- Q = {q1, q2, q3, q4}.
- Σ = {0, 1}.
- q0 = {q1}.
- F = {q2}.
- δ: From transition function **Q × Σ → Q**

	0	**1**
Q1	Q1	Q2
Q2	Q2	Q1
Q3	Q2	Q1
Q4	Q2	Q1

Table 2.1

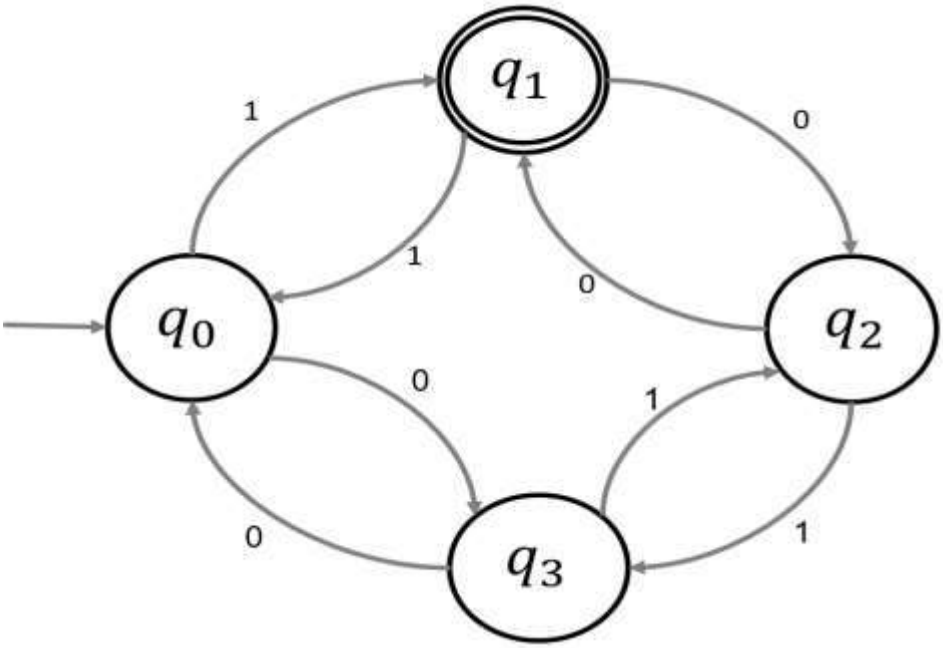

Figure 2.2 DFA Machine (1).

2.2.2 DFA Machine (2):

- DFA (2) is accepting strings that start with 1 and end with 0 or vice versa.

Solution

By using formal definition:

- Q = {q0, q1, q2, q3, q4}.
- Σ = {0, 1}.
- q0 = {q0}.
- F = {q2, q4}.
- δ: From transition function **Q × Σ → Q**

Table 2.3

	0	1
Q0	Q1	Q3
Q1	Q1	Q2
Q2	Q1	Q2
Q3	Q4	Q3
Q4	Q4	Q3

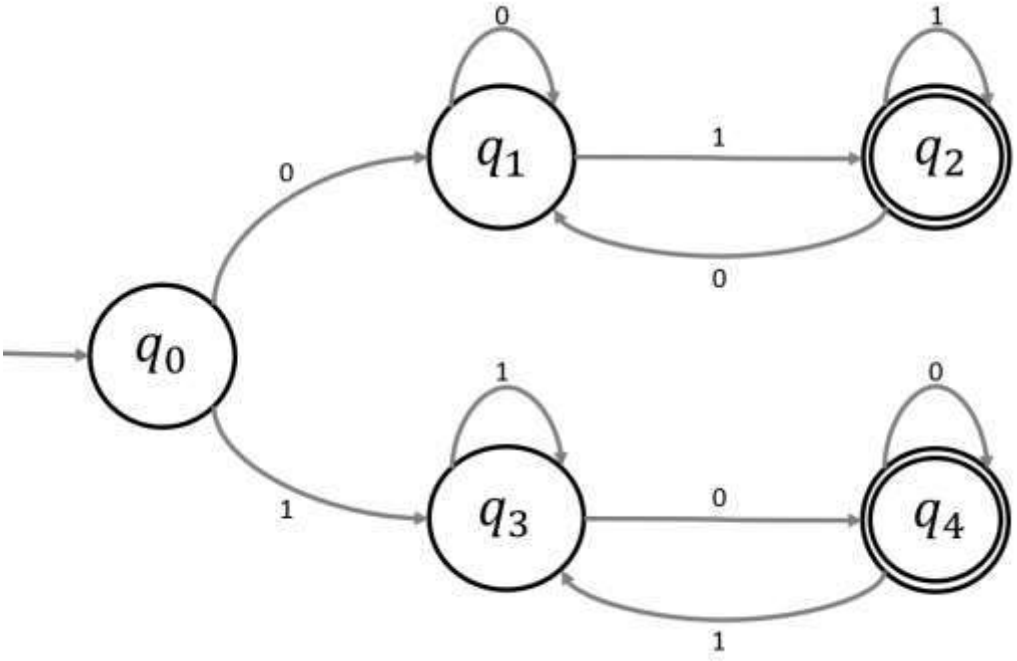

Figure 2.4 DFA Machine (2).

2.2.3 DFA Machine (3):

- DFA (3) is accepting the set of all strings with three consecutive 0's

Solution

By using formal definition:

- Q = {q0, q1, q2, q3}.
- Σ = {0, 1}.
- q0 = {q0}.
- F = {q3}.
- δ: From transition function **Q × Σ → Q**

	0	1
Q0	Q1	Q0
Q1	Q2	-
Q2	Q3	-
Q3	Q1	Q3

Table 2.5

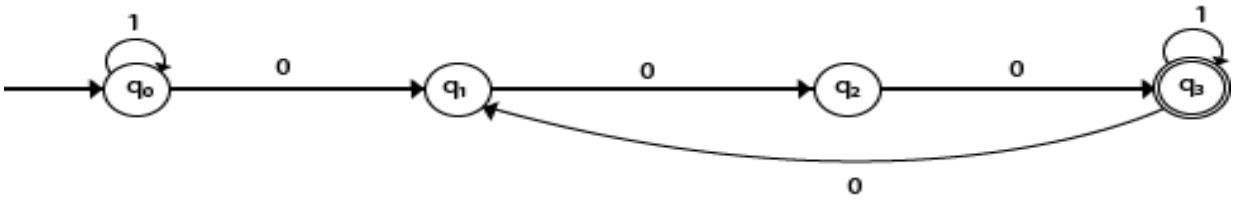

Figure 2.6 DFA Machine (3).

3. Machine Implementation:

- In this section we will talk about the machine and how it was implemented.

3.1 code illustration:

- Now, we will talk about the machine implementation:
  - **Language:** C++.
  - **Structure:** Dynamic Structure (Vector, Map, Set)
  - **Objective:** Check any strings for any DFA machines.
  - **Input:** All the details of DFA machine (number of nodes, alphabets, Nodes, Initial State, number of final state, Final state, number of dead state, Dead state, Strings). [**Figure 3.5, 3.6, 3.7**].
  - **Output:** "**Accepted**" if the DFA machine accepts it OR "**Rejected**" if the DFA machine don't accept it.
- Now, We will see every part of the code and how it works:

```
8.  #include<map>
9.  #include<vector>
10. #include<set>
11. #include<algorithm>
12. #include<cmath>
13. using namespace std;
14. set<int>accept;
15. char alpha1, alpha2;
16. map<int, pair<int, int>>DFA;// <state number ,< alpha1 go to , alpha2 go to  >
17. int nodesNumber, finalNodesNumber, initialState;
```

Figure 3.1

- In Figure 3.1 we included all libraries that needed for the code and formal definition of DFA machine.

```
18.  int DFA_path(string s)
19.  {
20.      int cur_node = initialState;
21.      for (int i = 0; i < s.size(); i++)
22.      {
23.          if (s[i] == alpha1) cur_node = DFA[cur_node].first;
24.          else cur_node = DFA[cur_node].second;
25.      }
26.      return cur_node;
27.  }
```

Figure 3.2

- In Figure 3.2 there is a function where we input a string and this output the last state which the string will reach.

```
28.  bool is_Accept(int node)
29.  {
30.      for (auto i : accept) if (node == i) return 1;
31.      return 0;
32.  }
```

Figure 3.3

- In figure 3.3 there is a function where we input the last state string stop then this function define whether this state is accepted state or not if this state is final state the function will return 1 if not will return 0.

```
33.  bool AcceptedString(string str)
34.  {
35.      for (int i = 0; i < str.size(); i++)
36.          if (str[i] != alpha1 && str[i] != alpha2)
37.              return 0;
38.      return 1;
39.  }
```

Figure 3.4

- In figure 3.4 there is a function that checks each character in string which we have inputted if it follows the alphabet machine or not.

```
41. int main()
42. {
43.     printLine();
44.     cout << "Please enter number of nodes in your DFA: \n";
45.     cin >> nodesNumber;
46.     printLine();
47.     cout << "Please enter two input alphabetical OF DFA :\n";
48.     cin >> alpha1 >> alpha2;
49.     printLine();
50.     cout << "Nodes ( States ) created  and it equal = \n";
51.     for (int i = 1; i <= nodesNumber; i++)
52.     {
53.         DFA[i];
54.         cout << i;
55.         (i == nodesNumber) ? cout << '\n' : cout << ' ';
56.     }
57.     printLine();
58.     cout << "What is the initial state in this DFA :\n";
59.     cin >> initialState;
60.     printLine();
61.     cout << "What is the number of final states in this DFA :\n";
62.     cin >> finalNodesNumber;
63.     printLine();
64.     for (int i = 0; i < finalNodesNumber; i++)
65.     {
66.         cout << "Please enter " << i + 1 << " final state : " << endl;
67.         int finalState; cin >> finalState;
68.         if (finalState <= nodesNumber) accept.insert(finalState);
69.     }
70.     printLine();
```

Figure 3.5

```
71.     for (int i = 1; i <= nodesNumber; i++)
72.     {
73.         int n1, n2;
74.         cout << "Node " << i << " at alphabet " << alpha1 << " = ";
75.         cin >> n1;
76.         cout << "Node " << i << " at alphabet " << alpha2 << " = ";
77.         cin >> n2;
78.         printLine();
79.         DFA[i].first = n1;
80.         DFA[i].second = n2;
81.     }
82.     vector<int>deadState;
83.     for (auto i : DFA)
84.     {
85.         if (i.second.first == i.second.second && i.second.first == i.first && accept.find(i.first) == accep
    t.end())
86.             deadState.push_back(i.first);
87.     }
88.     if (deadState.size() > 0)
89.     {
90.         cout << "Number of dead states = " << deadState.size() << '\n';
91.         cout << "Dead State are : ";
92.         for (auto i : deadState)
93.         {
94.             cout << i << ' ';
95.         }
96.         cout << '\n';
97.         printLine();
98.     }
99.     else
100.    {
101.        cout << "There is not dead state in the DFA.\n";
102.        printLine();
103.    }
```

Figure 3.6

```
    while (true)
    {
        cout << "Do you want to enter string !?\nEnter y or Y for yes or enter anything to Exit\n";
        char coun; cin >> coun;
        if (coun != 'y' && coun != 'Y') break;
        printLine();
        cout << "Please enter a String :\n";
        string s; cin >> s;
        if (!AcceptedString(s))
        {
            cout << "This invaled string\n";
            printLine();
            continue;
        }
        if (is_Accept(DFA_path(s))) cout << "The string : " << s << " is Accepted one.\n";
        else cout << "The string : " << s << " is Rejected one.\n";
        printLine();
    }
    printLine();
    return 0;
}
```

Figure 3.7

In figure 3.5, 3.6, 3.7 there is a main function which print the input instructions and call the functions which were explained in pervious figures.

- Firstly, we make user to input number of nodes in DFA.
- Then enter the alphabetical of DFA.
- Also make user to input the initial state in DFA, the number of accepted state in this machine.
- Also make user to specify each char will go for which state (transition table).
- If there is a dead state we will output the number of them and their name to user if is not, we will output to user there is no dead state in the DFA.
- The user will input the string then the code will check if each char in this string is following the alphabet of machine and if this follow we will output this string is valid.

3.2 Machine Examples:

In this section, we will show examples of the string that should be accepted or rejected and see how the machine acts with the example.

3.2.1 Machine (1):

A machine that accepts any string that has even 0's and odd 1's.

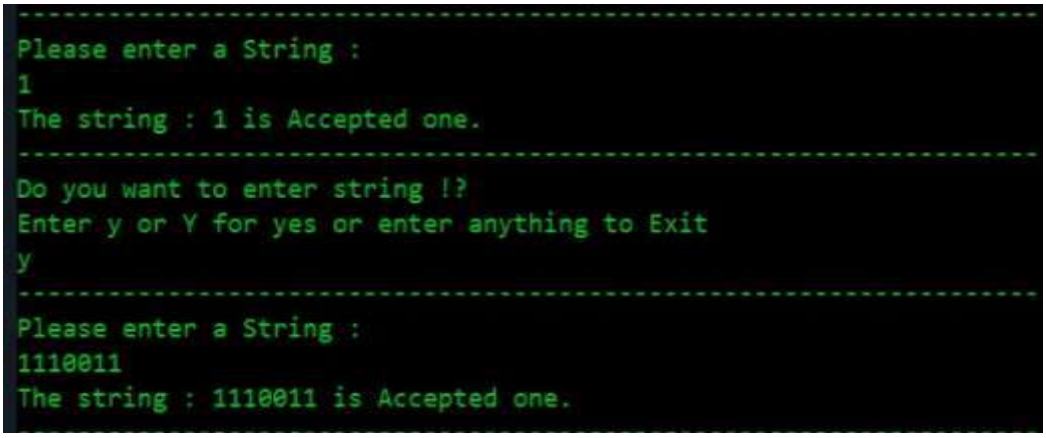

Figure 3.8 Accept example for Machine 1

Figure 3.9 Reject example for Machine 1

3.2.2 Machine (2):

A machine that Accepts strings that start with 1 and end with 0 or vice versa.

Figure 3.10 Accept example for Machine 2

Figure 3.11 Reject example for Machine 2

3.2.3 Machine (3):

A machine that accepts the set of all strings with three consecutive 0's.

Figure 3.12 Accept example for Machine 3

Figure 3.13 Reject example for Machine 3

6. Appendix:

```
*/
#include<iostream>
#include<map>
#include<vector>
#include<set>
#include<algorithm>
#include<cmath>
using namespace std;
set<int>accept;
char alpha1, alpha2;
map<int, pair<int, int>>DFA;// <state number ,< alpha1 go to , alpha2 go to  >
int nodesNumber, finalNodesNumber, initialState;
int DFA_path(string s)
{
    int cur_node = initialState;
    for (int i = 0; i < s.size(); i++)
    {
        if (s[i] == alpha1) cur_node = DFA[cur_node].first;
        else cur_node = DFA[cur_node].second;
    }
    return cur_node;
}
bool is_Accept(int node)
{
    for (auto i : accept) if (node == i) return 1;
    return 0;
}
bool AcceptedString(string str)
{
    for (int i = 0; i < str.size(); i++)
        if (str[i] != alpha1 && str[i] != alpha2)
            return 0;
    return 1;
}
void printLine() { cout << "------------------------------------------------------------------------\n\a"; }
```

[B6tz1e - Online C++0x Compiler & Debugging Tool - Ideone.com]

```
41.  int main()
42.  {
43.      printLine();
44.      cout << "Please enter number of nodes in your DFA: \n";
45.      cin >> nodesNumber;
46.      printLine();
47.      cout << "Please enter two input alphabetical OF DFA :\n";
48.      cin >> alpha1 >> alpha2;
49.      printLine();
50.      cout << "Nodes ( States ) created  and it equal = \n";
51.      for (int i = 1; i <= nodesNumber; i++)
52.      {
53.          DFA[i];
54.          cout << i;
55.          (i == nodesNumber) ? cout << '\n' : cout << ' ';
56.      }
57.      printLine();
58.      cout << "What is the initial state in this DFA :\n";
59.      cin >> initialState;
60.      printLine();
61.      cout << "What is the number of final states in this DFA :\n";
62.      cin >> finalNodesNumber;
63.      printLine();
64.      for (int i = 0; i < finalNodesNumber; i++)
65.      {
66.          cout << "Please enter " << i + 1 << " final state : " << endl;
67.          int finalState; cin >> finalState;
68.          if (finalState <= nodesNumber) accept.insert(finalState);
69.      }
70.      printLine();
71.      for (int i = 1; i <= nodesNumber; i++)
72.      {
73.          int n1, n2;
74.          cout << "Node " << i << " at alphabet " << alpha1 << " = ";
75.          cin >> n1;
```

B6tz1e - Online C++0x Compiler & Debugging Tool - Ideone.com

```
74.          cout << "Node " << i << " at alphabet " << alpha1 << " = ";
75.          cin >> n1;
76.          cout << "Node " << i << " at alphabet " << alpha2 << " = ";
77.          cin >> n2;
78.          printLine();
79.          DFA[i].first = n1;
80.          DFA[i].second = n2;
81.      }
82.      vector<int>deadState;
83.      for (auto i : DFA)
84.      {
85.          if (i.second.first == i.second.second && i.second.first == i.first && accept.find(i.first) == accep
     t.end())
86.              deadState.push_back(i.first);
87.      }
88.      if (deadState.size() > 0)
89.      {
90.          cout << "Number of dead states = " << deadState.size() << '\n';
91.          cout << "Dead State are : ";
92.          for (auto i : deadState)
93.          {
94.              cout << i << ' ';
95.          }
96.          cout << '\n';
97.          printLine();
98.      }
99.      else
100.     {
101.         cout << "There is not dead state in the DFA.\n";
102.         printLine();
103.     }
104.     while (true)
105.     {
106.         cout << "Do you want to enter string !?\nEnter y or Y for yes or enter anything to Exit\n";
107.         char coun; cin >> coun;
```

B6tz1e - Online C++0x Compiler & Debugging Tool - Ideone.com

```
95.         }
96.         cout << '\n';
97.         printLine();
98.     }
99.     else
100.    {
101.        cout << "There is not dead state in the DFA.\n";
102.        printLine();
103.    }
104.    while (true)
105.    {
106.        cout << "Do you want to enter string !?\nEnter y or Y for yes or enter anything to Exit\n";
107.        char coun; cin >> coun;
108.        if (coun != 'y' && coun != 'Y') break;
109.        printLine();
110.        cout << "Please enter a String :\n";
111.        string s; cin >> s;
112.        if (!AcceptedString(s))
113.        {
114.            cout << "This invaled string\n";
115.            printLine();
116.            continue;
117.        }
118.        if (is_Accept(DFA_path(s))) cout << "The string : " << s << " is Accepted one.\n";
119.        else cout << "The string : " << s << " is Rejected one.\n";
120.        printLine();
121.    }
122.    printLine();
123.    return 0;
124. }
```

B6tz1e - Online C++0x Compiler & Debugging Tool - Ideone.com

6. Reference

- [(1.1) introduction: theory of automata] https://cs.stanford.edu/people/eroberts/courses/soco/projects/2004-05/automata-theory/basics.html
- [(1.2) introduction: DFA Machines] https://www.javatpoint.com/deterministic-finite-automata
- [(2.1) Machine design: DFA Definition] https://www.tutorialspoint.com/automata_theory/deterministic_finite_automaton.htm
- [(2.2) Machine Design: Case Studies] https://www.javatpoint.com/examples-of-deterministic-finite-automata
- [(3.1) Machine Implementation] B6tz1e - Online C++0x Compiler & Debugging Tool - Ideone.com