**Deterministic Finite Automata Project**

**Team members:**

1. **Ahmed Alaa El-Dein Ahmed.**
2. **Diab Saudi Dab Mansour.**
3. **Ramez Nasser Fayez Bakheet.**

First of all, we should not jump to the details of the project until we discuss two main definitions. First, automata are a field of computer science that deals with the study of abstract machines. It is a set of algorithms used to create an automaton that is capable of recognizing patterns and making decisions without explicit programming. Automata have many practical applications, from analyzing data to recognizing expressions. Automata can provide complex systems, making them indispensable tools for developers.

Automata is built using regular expressions. These regular expressions are used to define the syntax and structure of the language. Automata is flexible that can produce a lot of solutions to various types of problems.

Second, deterministic finite automata (DFA) are the most basic type of automata. A DFA is a machine composed of states, input symbols, and transition functions. The DFA reads an input string of symbols and transitions from one state to another for each symbol in the string, until arriving at a final state. If the final state is a member of the set of accepted states, the input string is accepted otherwise, it is rejected.

DFA is a set of finite states that can be used to define a language. They are a type of abstract machine that consists of a finite set of states, a set of transitions, and an input string. The transitions are defined by a transition function that takes as input a pair of states and symbols and produces a new state as output. The input string is used to determine the behavior of the machine, and the output string is the result of running the machine on the input.

Finally, we have known about automata and DFA, now we can jump to details.

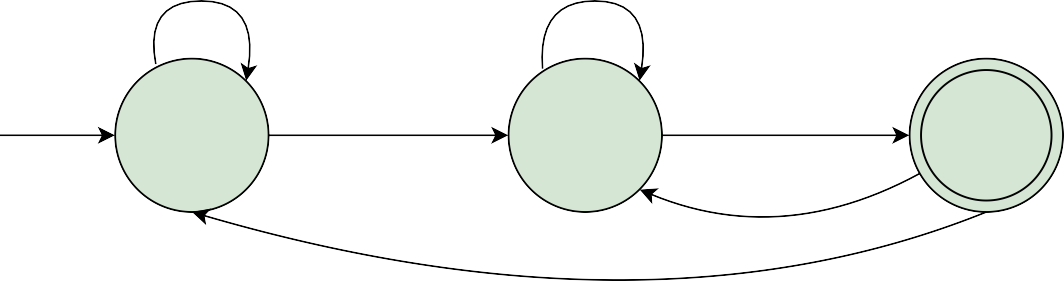
**Machine Design (M1):**

* M = {over (0,1) | strings that end with 01}
* Ex: (01, 101, 1101, 0001, 10101, 0101)

1

0

1



q0

0

q1

1

q2

0

1. **Q** = set of states = {q0, q1, q2}.
2. **∑** = alphabet = {0, 1}.
3. ∂ = transition function (Table).

|  |  |  |
| --- | --- | --- |
| States | 0 | 1 |
| q0 | q1 | q0 |
| q1 | q1 | q2 |
| q2 | q1 | q0 |

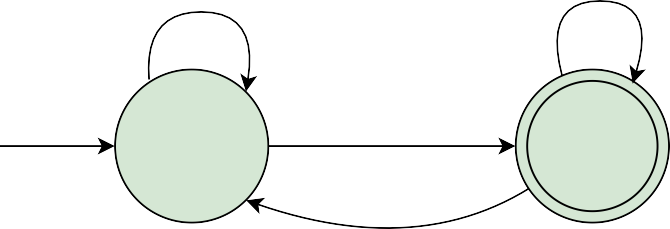
1. **qO** = start state = q0.
2. **F** = set of final states = {q2}.

**Machine Design (M2):**

* M = {over (0,1) | strings that end with 1}
* Ex: (01, 101, 1101, 0001, 10101, 0101)

1

0



q0

1

q1

0

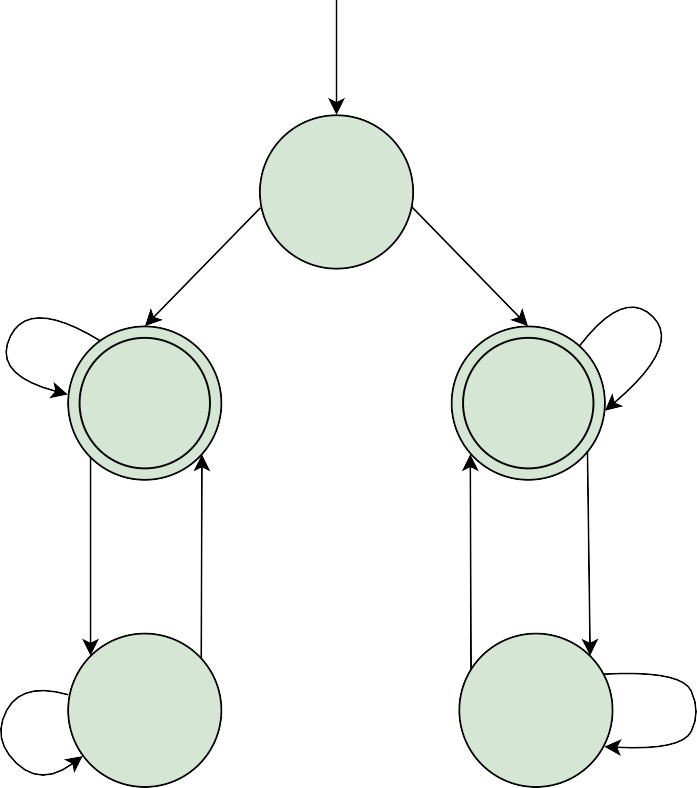
1. **Q** = set of states = {q0, q1}.
2. **∑** = alphabet = {0, 1}.
3. ∂ = transition function (Table).

|  |  |  |
| --- | --- | --- |
| States | 0 | 1 |
| q0 | q0 | q1 |
| q1 | q0 | q1 |

1. **qO** = start state = q0.
2. **F** = set of final states = {q1}.

**Machine Design (M3):**

* M = {over (0,1) | strings that end with the same symbol}
* Ex: (aba, bbab, babab, bbb, babbab)



q0

a

b

b

q1

q3

b

a

b

a

q2

q4

a

a

a

b

1. **Q** = set of states = {q0, q1, q2, q3, q4}.
2. **∑** = alphabet = {a, b}.
3. ∂ = transition function (Table).
4. **qO** = start state = q0.
5. **F** = set of final states = {q1, q3}

**Documentation about our code**

Big picture:

How do we build our code? In fact, we were confused about how can we build a code that can solve any DFA we give to it. and we think lowly if we can give the final code -after developing it- the graphical representation of the DFA, and finally, we have reached out to ask the user for entering the number of the states and take all states' information from him.

Every state has some main points the program should know, the name of the state, the arrows that leave the state, and where these arrows will get you.

Jumping to details:

Now we will discuss a bit more details about our code and how it works. First, we create a class called State and used it to represent our states in our machine. Every state will be represented in code as a name and two arrows get out from the state and every arrow will be represented as an alphabet and the name of the state that it will go to. So, we created a structure that represented the arrow objects. and also, we create a method to take all these inputs from a user called set\_Attributes. also, we created a get\_name method that returns the name of the state as we will see in the main function and finally -in this class- we created the most important method called Regular\_Function which takes one character and returns the name of the state that we will go to.

Here we should stop and discuss with more care what this method takes as arguments. In fact, we used the current state object to call this method with it to know which state we are in, and by comparing the character that the method takes with the character that the state object contains we will know which state we will go to.

Now in our main, we will ask the user to enter the number of states, and using for loop and vector we will take every state's data and push the object in our vec vector indicating him to enter the start state as the first one.

Also, we will ask the user to enter the number of final states and then enter them all using for loop and save them all in the final\_states vector. Finally, we will take the string that we want to make test on it.

How the code will work? Our code will take every character in this string and call the method Regular\_Function as we mentioned. The variable j we used to represent the index of the state and we used for loop to search for the state that equals the found variable that the Regular\_Function returned. Finally, we checked if the name of the last state object that we reached equals any name that the final\_states vector contains to know if we are accepted or not.

**References**

For the paragraph on automata and DFA machines we depended on our understanding of the course and our study from the book.

For The Machine Design and The Examples of Running the Developed Code: We used one of your examples that are on slides, the second two we get them from the book pages 37 and 38.

For our code we build it depending on ourselves, not depending on an outside source.

Finally, if you want to run our code you can find it in the team leader repository.

<https://github.com/AhmedAlaa4611/Atometa.git>

Other References

<https://www.geeksforgeeks.org/introduction-of-pushdown-automata/>

<https://www.javatpoint.com/automata-tutorial>

<https://en.wikipedia.org/wiki/Deterministic_finite_automaton>