Discrete Math - Mini Project Draft Report

TurboTurists

December 9, 2021

Abstract

Finite State Machine (FSM) is a mathematical model of computation that is made of one or more states. The FSMs store a sequence of different unique states and transitions between them depend on the values of the inputs and the current state of the machine. FSMs are commonly used to control and represent execution flow.

Our project is to create a program that allows users to construct the FSM diagrams online. The platform visualizes the dynamic states as well as any process and flow, and displays in an interactive browser. It is featured with many powerful tools such as drag and drop, alignment guide, save and share, or options for fonts and custom shapes, which will help users easily perform certain actions and design any diagrams.

While the front-end of the web app will be developed through the use of HTML, CSS, Java Script, our back-end will rely on SQL, Java.

1 Introduction

Our team TurboTurists is composed of 4 aspiring CS students: Cuong, Hiep, Khoi, Vy.

For any computer science majors, the finite state machine (FSM) is an important concept to know because it directly relates to the underlying concepts of how computers work, as it represents a computational model used to simulate sequential logic. As a result, when designing any computational hardware system, the FSM is usually the ideal starting place, acting as a blueprint upon which rigorous systems shall be built. However, currently there are little to no practical FSM simulation systems available to the public, and the existing solutions are either limited in their supporting features, or simply lacking in interactivity.

For the mini-project, we are implementing an FSM simulation. The entire program will be hosted on a website, which users can go to, input their desired attributes of the FSM, watch as it is being built in real time on the web, and play around with different interactive elements, including but not limited to a user-friendly drag-n-drop control, customizable fonts and colors, smart notation systems,... to design their machine's diagram.

2 Mathematical Background / Implementation

In response to some inputs, our finite state machine can shift from one state to another. It can minimize the number of execution pathways across the code, simplify the criteria examined at each branching point, and make transitions between modes of execution simpler. As it turns out, most reactive systems' behavior can be separated into a reasonably small number of non-overlapping states, with event reactions inside each individual chunk depending simply on the current event and no longer on the past events. The model is built based on re-implementing Deterministic Finite State Machines and Non-deterministic Finite State Machines.

2.1 Automata Theory

Automata Theory is an area of computer science dealing with the construction of abstract self-propelled computing systems that automatically follow a predetermined sequence of operations. An automaton with a finite number of states is called a **Finite Automaton**. (**?**)

The term "Automata" comes from the Greek word "automata", meaning "self-acting." An automaton (plural: automata) is an abstract self-propelled computing system that automatically follows a predetermined series of operations.

An automaton having a finite number of states is called a Finite Automaton (FA) or Finite State Machine (FSM).

An automaton can be represented by a 5-tuple (Q, Σ, δ, q_0, F), where: (1)

- Q is a finite set of states.
- Σ is a finte set of symbols, called alphabets of the automaton
- δ is the transition function
- q_0 is the initial state from where any input is processed
- F is a set of final state(s) of Q

(Tuteja, 2019) The outputs of the finite automata might be linked to each transition. There are two different types of output-generating FSM, including Mealy type, and Moore type.

2.1.1 Mealy Machine

A Mealy Machine is an FSM whose output is determined by the current state as well as the current input.

It can be described by a 6 tuple (Q, Σ, O, δ, X, q_0) where:(1)

- Q is a finite set of states.
- Σ is a finte set of symbols, called alphabets of the automaton
- O is a finite set of symbols called the output alphabet.
- δ is the input transition function where δ: Q $\times$ Σ $\rightarrow$ Q
- X is the output transition function where X: Q $\times$ Σ $\rightarrow$ O
- q_0 is the initial state from where any input is processed

(Tuteja, 2019)

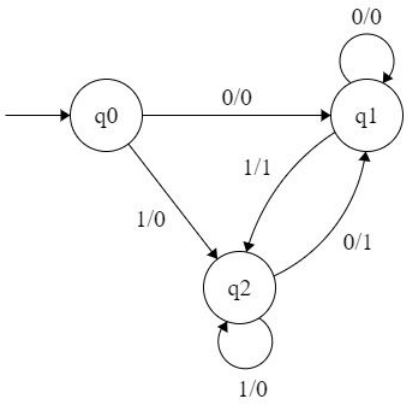

Figure 1: Mealy Machine Sample

The output of this mealy machine is expressed by separating each input symbol for each state with a /. The output length of the mealy machine is similar to the input length. (1)

- Input: 11
- Transition: $\delta(q_0,11) \rightarrow \delta(q_2,1) \rightarrow q_2$
- Output: 00 (q_0 to q_2 transition has Output 0 and q_2 to q_2 transition also has Output 0)

2.1.2 Moore Machine

Moore machine is an FSM whose outputs depend on only the present state.

A Moore machine can be described by a 6 tuple (Q, Σ, O, δ, X, q_0) where: (1)

- Q is a finite set of states.
- Σ is a finte set of symbols called input alphabets of the automaton
- O is a finite set of symbols called the output alphabet.
- δ is the input transition function where δ: Q × Σ → Q
- X is the output transition function where X: Q → O
- q_0 is the initial state from where any input is processed

(Tuteja, 2019)

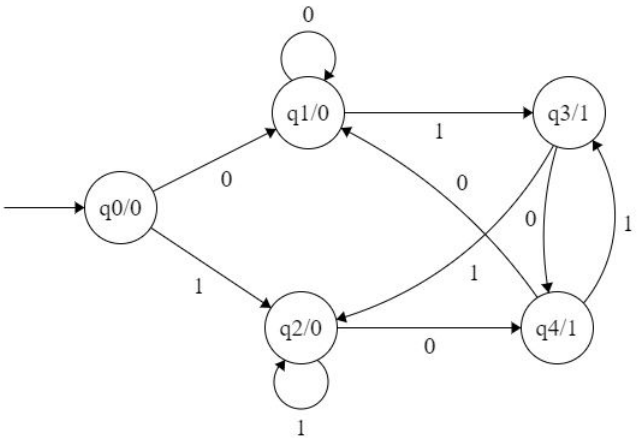

Figure 2: Moore Machine Sample

The output of the moore machine is displayed with each input state separated by /. A moore machine's output length is greater than input by 1. (1)

- Input: 11
- Transition: δ (q_0,11) → δ(q_2,1) → q_2
- Output: 000 (0 for q_0, 0 for q_2 and again 0 for q_2)

2.1.3 Example

Task: Implement a finite state machine which detects each occurrence of the string "101" in any given binary input string. For example, the following input should produce the following output:

INPUT: 11101101001

OUTPUT: 00001010000

Figure 3: Task

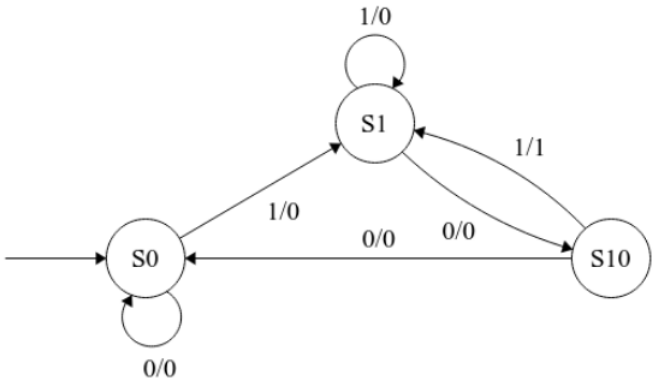

Figure 4: Mealy Solution

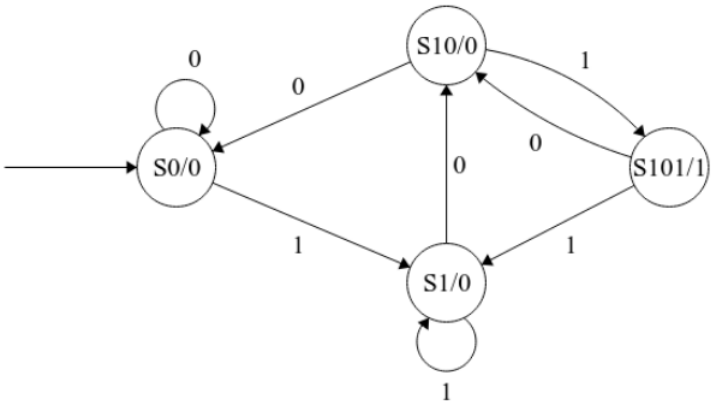

Figure 5: Moore Solution

2.2 Implementation - Tech stack

Our project is built using HTML to structure the page that receives data from the machine described by the user, CSS to design the page that reads the data, and JavaScript (together with the jQuery library) to process the data, construct the state machine, and perform the click events.

The code includes the HTML implementation of the form that receives the user's input, as well as the JavaScript code that selects and saves the data entered by users as variables. The critical part of the code is to generate the circles and arrows from the number of states and their future states. In addition, it has the event of the input buttons, which traverses an array containing the table with states and future states from the current state and the input clicked, and finally decides which state to transition to.

2.2.1 Circles and Arrows generation

While seemingly complex and difficult, the generation of the circles and arrows used for visualisation of the FSMs are, on the contrary, quite simple to implement.

The positions of the centers of circles (with fixed radii) have been planned out, for each of the 8 cases (depending on N - the number of states, from 1 to 8). Each case will have its own pre-defined layouts of the circles (consisting of all the (x, y) coordinates of the N circles). With this, all it takes is some simple if-else statement and extracting information from user's inputs, in order to come up with the corresponding circles formation.

Generations of the arrows also work in a similar manner. Each circles' center will be the starting and ending points of arrows corresponding to it. Then, drawing the arrows is achieved also through simple conditional statements from the user's inputs.

2.3 Results

Our product: s4shanull.github.io/vinuni-fsm

2.3.1 Machine type: Mealy

Number of states: 3

Truth table:

Current State	Input	Next State	Output
1	0	1	0
1	1	2	0
2	0	3	0
2	1	1	0
3	1	1	1
3	0	2	1

Result:

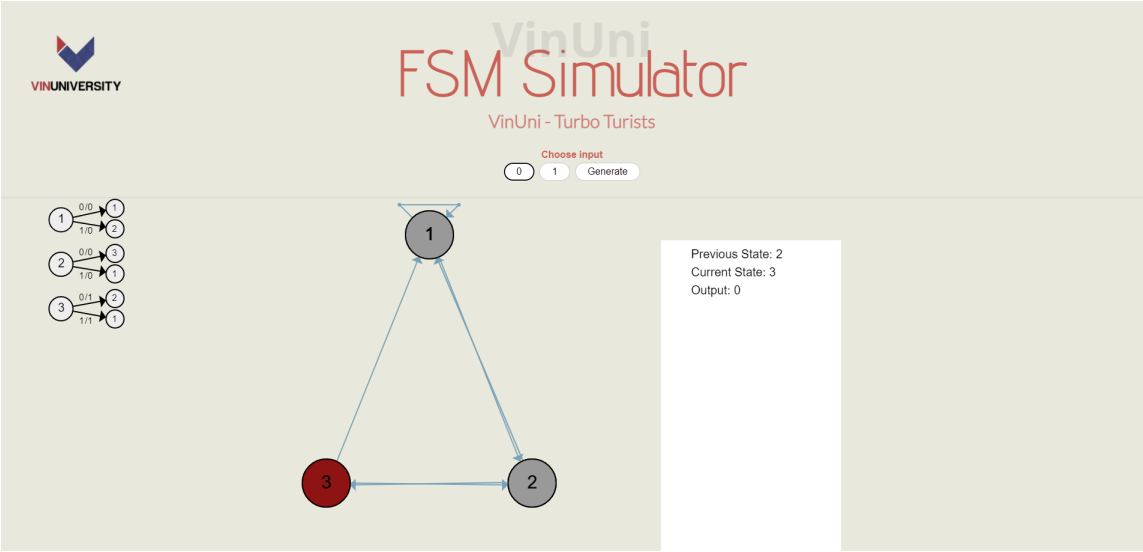

Figure 6: Mealy Machine Sample

2.3.2 Machine type: Moore

Number of states: 5

Truth table:

Current State	Input	Next State	Output
1	0	3	1
1	1	1	1
2	0	4	0
2	1	4	0
3	0	4	0
3	1	5	1
4	0	3	1
4	1	3	1
5	0	1	1
5	1	2	0

Result:

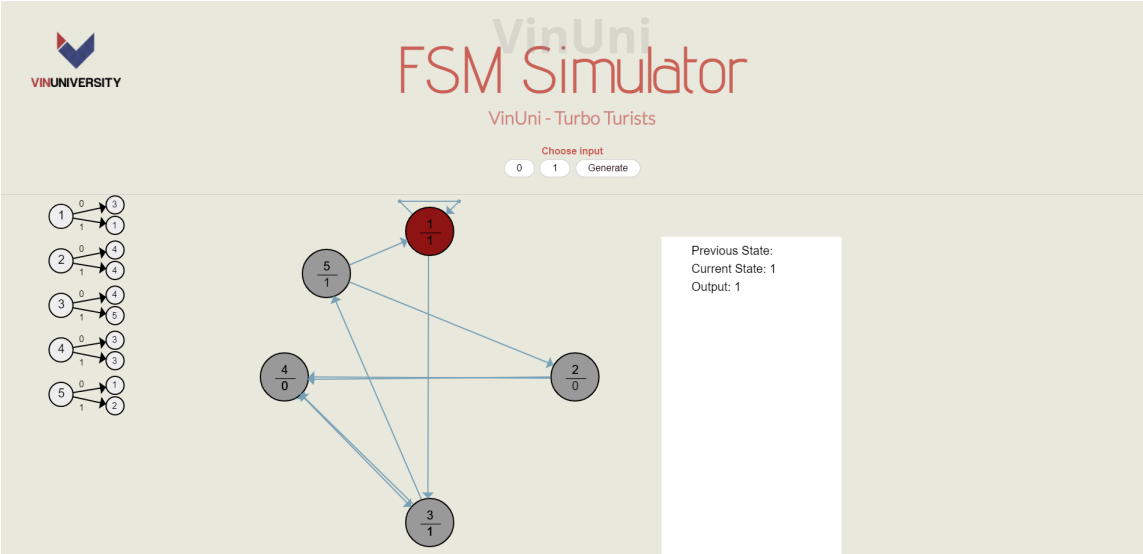

Figure 7: Moore Machine Sample

2.4 Discussion

2.4.1 Limitations - Circles and Arrows generation

Implementation of the generations of circles and arrows are done by pre-defining the layouts of the circles, then generate the corresponding elements. Because of this, currently the number of states is capped at 8, and coding an additional state (9, 10, ...) becomes linearly more difficult, as we have to go through the process of creating a layout for each circle in the 8+ circle formation, then for each of them create the necessary conditional statements for their (and their arrows) generations.

2.4.2 Limitations - Web Responsiveness

We have not been able to apply responsiveness to the web. All positions (from HTML elements like titles, texts,...) to the actual circles and arrows,... are all hard-coded, with specific pixel values and coordinates. Because of this, the web achieves little scalability when it comes to mobile / tablet usage, and even usage on computers of different screen sizes.

References

[1] Ankit, "Answer for 'What is automata theory?" - Quora," Quora, 05-Oct-2017. [Online]. Available: https://www.quora.com/What-is-automata-theory. [Accessed: 09-Dec-2021].

[2] "Moore and Mealy Machines," tutorialspoint, 2020. [Online]. Available: https://www.tutorialspoint.com/automatatheory/mooreandmealy_machines.htm. [Accessed: 09-Dec-2021].

[3] S. Tuteja, "Mealy and Moore Machines in TOC," GeeksforGeeks, 20-Nov-2019. [Online]. Available: https://www.geeksforgeeks.org/mealy-and-moore-machines-in-toc/. [Accessed: 09-Dec-2021].