

CSC615M MP2: Equivalence of Two Finite State Machines

Ryan Austin Fernandez
CS-ST Student
2401 Taft Avenue,
Manila, Philippines 1004
(+63)917-621-6469
ryan_fernandez@dlsu.edu.ph

ABSTRACT

Finite State Machines are the simplest formal machines. They represent a set of transitions between a set of states producing a particular output or recognizing a particular regular language. In implementing these actual machines in hardware, it may be desirable to select a more simply designed machine. However, it is necessary to ensure that the simpler design is equivalent to the original design. This is done by using the equivalence algorithm in the previous project on the two start states of the machines. The algorithm was coded in Python and tested using equivalence class black box testing and passed all tests. This project can then be said to be successful in determining equivalent machines and can be used in future projects regarding machine equivalence and modelling a Turing Machine.

I. INTRODUCTION

Finite State Automata are the simplest formal machines taught in undergraduate Theory of Computation classes. Their operation simply consists of a finite number of states, stimuli, and responses, transitions from states on different stimuli, and output mappings (Hopcroft, J.E. et al, 2006). There are three basic types of finite state machines: Mealy machines, where output is assigned to transitions; Moore machines, where output is assigned to states; and Finite State Accepters, where the output is only produced after the entire input string is read and the output is either acceptance or rejection (Denning, P.J., et al., 1978).

In implementing FSM's, it may be necessary to select a simpler design for the same machine. However, it is necessary to ensure that the simpler designed machine is equivalent to the original design.

A machine M1 is equivalent to another machine M2, if they produce the same output for every input string, when it comes to transducers, or the machines both accept or reject the same strings, when talking about accepters (Denning, P.J. et al., 1978). This is easily determined for small machines, but for larger, more complicated machines, it may be desirable to automate this process.

This project aims to automate the process of determining whether two machines are equivalent. This project will only handle finite state machines of a definite type. This project will not cover hybrid accepters/Mealy machines/Moore machines. For finite state accepters, this project can only handle the deterministic type.

II. DESIGN

The algorithm used was an extension of the algorithm found in Hopcroft, J.E. et al (2006). In this textbook, to prove two states are equivalent, they must both be either accepting or rejecting and any state the two states transition into must be equivalent as well. To extend this into Mealy and Moore machines, one can consider a finite state accepter as a Moore

machine with $R = \{0, 1\}$, 0 meaning rejecting state and 1 meaning accepting state. One can further extend this to Mealy machines by considering all output mappings on all stimuli symbols.

Proving two states equivalent by the basic definition is impossible, however, since it is impossible to test all possible input strings as there are an infinite number of strings. What can be done instead is to prove that certain paths in the FSM would lead to distinguishable (non-equivalent) states.

Given that pairs of states in a single automaton can be represented as vertices and transitions can be represented as edges, this becomes a simple graph search problem. The goal of the search is to locate a vertex that contains two distinguishable nodes. If the entire graph is exhausted and a goal vertex is not found, it can be concluded that the nodes are distinguishable.

Table 2.1. is a depth-first search algorithm adapted from Russel, S. & Norvig, P. (2010).

Table 2.1. – DFS Algorithm for State Equivalence

DFS Search
DFS-equiv(currNode, explored, stimuli) returns TRUE if currNode's states are equivalent and FALSE otherwise
currNode – has s1 and s2, the states to compare
explored – list of explored nodes
stimuli – list of input symbols
check all transition outputs for Mealy Machine, and static state output for Moore and FSA
if currNode's state's outputs are different
return FALSE
else
add currNode to explored
for each input symbol s in stimuli
expand gets the two states that the states in currNode transition into
newStates = expand(currNode, s)
if newStates not in explored
if DFS-equiv(newStates, explored, stimuli) returns FALSE
return FALSE
if all target states are visited but none are distinguishable
return TRUE

This algorithm is simply run on the start states of two finite state machines to determine if the two machines are equivalent. It is also run on each group of states in each machine to provide the one-to-one correspondence of states of a machine to another machine.

III. IMPLEMENTATION

The system was implemented in Python. The project had four modules: FSM.py for modelling the actual machine, State.py to model the states, FSMReader.py to read the file, and Main2.py to act as the driver.

The State class used a Python dictionary to store the mappings from input symbols to states and outputs. The FSM class used a dictionary to map the names of the states to the states themselves. Each state, when added to the FSM, has the FSM's name prepended to the state name. The FSM adapts the name of the file it originated from.

IV. TESTING

Four major test cases were written. This is in three sets of two cases, one where there are equivalent states present and one where there are none. The three sets are if there is one input symbol, if there are two input symbols, and if there are more than two.

The first test case involves two single input FSAs that accepts odd numbers. The one with no redundant states as shown in Figure 4.1. This was stored in file odd.txt.

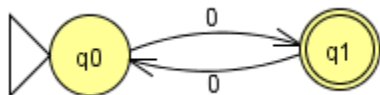


Figure 4.1. – FSA accepting odd number of 0's

The second machine was the same FSA with redundant states as seen in Figure 4.2. This was stored in file oddRedundant.txt.

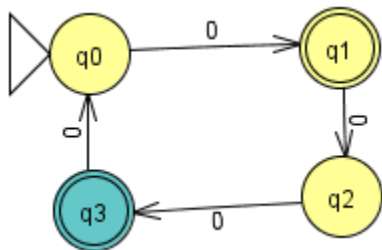


Figure 4.2. – FSA accepting odd number of 0's with redundant states

The outputs for these two cases are shown in Figure 4.3.

```

Machine Equivalence Checker
Leave input empty to quit
Enter filename for machine 1: odd.txt
Enter filename for machine 2: oddRedundant.txt
odd and oddRedundant are equivalent.
Equivalent Group 1:
  From odd: q0
  From oddRedundant: q0,q2
Equivalent Group 2:
  From odd: q1
  From oddRedundant: q1,q3
  
```

Figure 4.3. – Outputs for first Test Cases

For the first set, the program outputted the expected output.

The second set of test cases had Moore Machines. The machine outputs A if there have been no two symbols that are adjacent and equal yet and B if there has been. Figure 4.4. shows the minimized version and Figure 4.5. shows the redundant version.

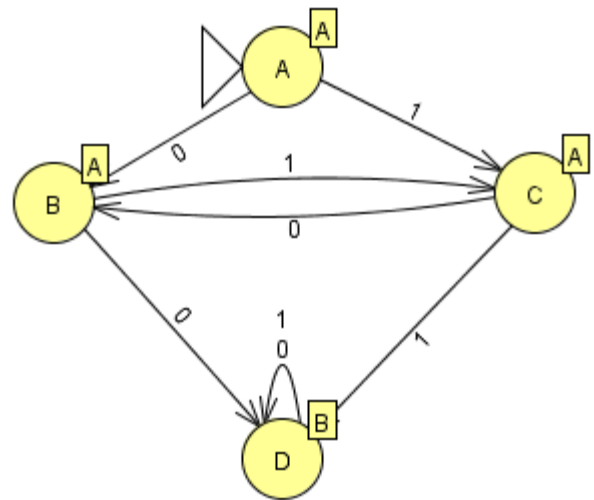


Figure 4.4. – Consecutive Symbol Detector Moore Machine

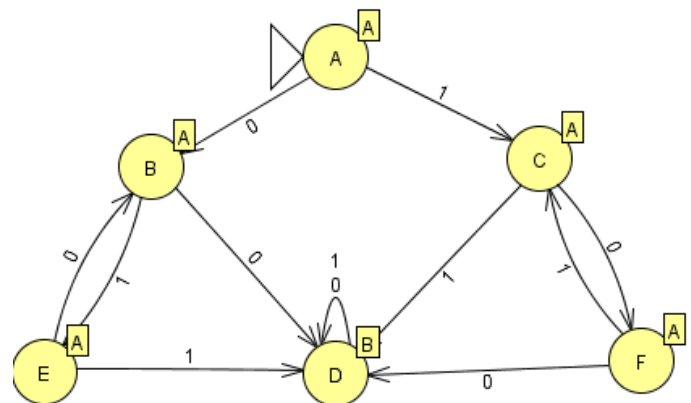


Figure 4.5. – Consecutive Symbol Detector Moore Machine with Redundant States

The program's outputs for the two are in Figure 4.6.

```

Machine Equivalence Checker
Leave input empty to quit
Enter filename for machine 1: consecSimple.txt
Enter filename for machine 2: consecRedundant.txt
consecSimple and consecRedundant are equivalent.
Equivalent Group 1:
  From consecSimple: A
  From consecRedundant: A
Equivalent Group 2:
  From consecSimple: B
  From consecRedundant: B,F
Equivalent Group 3:
  From consecSimple: C
  From consecRedundant: C,E
Equivalent Group 4:
  From consecSimple: D
  From consecRedundant: D
  
```

Figure 4.6. – Output for Second Set of Test Cases

For the second set, the program outputted the expected output.

For the last set, a Mealy machine with inputs 0, 1, and X was created. The machine would start out echoing the input until an X is encountered. It outputs the X but will then complement the input until another X is encountered. Figure 4.7. shows this machine minimized and Figure 4.8. shows it with redundant states.

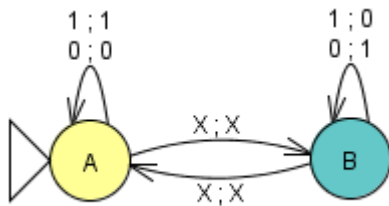


Figure 4.7. – Echo/Complement Machine

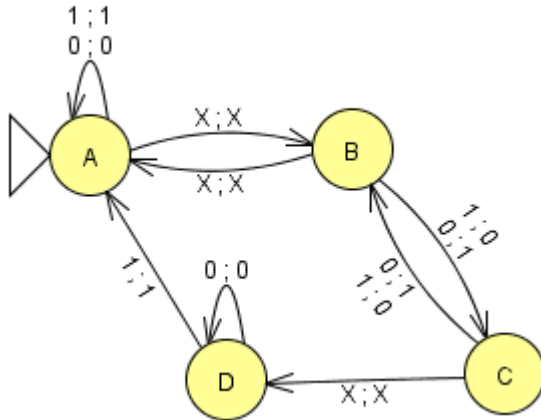


Figure 4.8. – Echo/Complement Machine with Redundant States

Figure 4.9 shows the program's output, which is the expected output.

```
Machine Equivalence Checker
Leave input empty to quit
Enter filename for machine 1: echoReverse.txt
Enter filename for machine 2: echoReverseRedundant.txt
echoReverse and echoReverseRedundant are equivalent.
Equivalent Group 1:
From echoReverse: A
From echoReverseRedundant: A,D
Equivalent Group 2:
From echoReverse: B
From echoReverseRedundant: B,C,E
```

Figure 4.9. – Output for Third Set of Test Cases

By equivalence class testing, these test cases should prove that the program is in an acceptable state.

For testing the negative response, since equivalence of state has two requirements: that the outputs are the same and the successor states are equivalent (Denning, P.J. et al., 1978), each of these were tested.

Consider a machine that outputs the 2's complement of a number fed in reverse as shown in Figure 4.10.

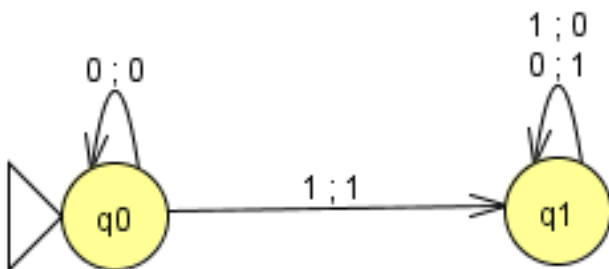


Figure 4.10. – Mealy Machine for Reverse 2's Complement

Also consider a machine that outputs the 1's complement of a number in reverse, as shown in Figure 4.11.

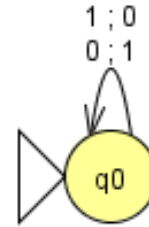


Figure 4.11 – Mealy Machine for Reverse 1's Complement

The start states are not equivalent because they produce different outputs. Figure 4.12 shows the output of the program.

```
Machine Equivalence Checker
Leave input empty to quit
Enter filename for machine 1: 2sComplement.txt
Enter filename for machine 2: 1sComplement.txt
2sComplement and 1sComplement are not equivalent.
```

Figure 4.12 – Output for Fourth Set of Test Cases

Now comparing the 2's complement machine to an echo machine, as shown in Figure 4.13, the start states produce the same output but the successor state for input 1 are not equivalent.

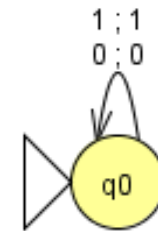


Figure 4.13 – Mealy Machine that Echoes its Input

The output is shown in figure 4.14. In the last two cases, the program produced the correct output.

```
Machine Equivalence Checker
Leave input empty to quit
Enter filename for machine 1: 2sComplement.txt
Enter filename for machine 2: echo.txt
2sComplement and echo are not equivalent.
```

Figure 4.14. – Output for Fifth Set of Test Cases

V. CONCLUSION

In conclusion, the project was successfully implemented. The program can successfully determine whether two finite state automata are equivalent, whether it be an acceptor, Mealy, or Moore Machine.

REFERENCES

- [1] Denning, P.J., Dennis, J.B., & Qualitz, J.E. (1978). *Machines, languages, and computation*. New Jersey: Prentice Hall.
- [2] Hopcroft, J.E., Motwani, R., and Ullman, J. D., (2006), *Introduction to automata theory, languages and computation 3rd edition*, Addison-Wesley Publishing Company. Reprinted by Jemma Inc.
- [3] Russell, S. & Norvig, P. (2010). *Artificial Intelligence: A Modern Approach*, 3rd Ed. New Jersey: Prentice-Hall.