**CSC615M - Machine Project 2 Documentation**

Aquino, Kurt Neil

De La Salle University, Manila

254 Sitio Bagong Anyo, Pagsawitan, Santa Cruz, Laguna

(+63) 9273597974

kurt\_aquino@dlsu.edu.ph

1. **INTRODUCTION**

For the second learning output for the course of CSC615M – Automata Theory, Computability, Formal Languages, the students were tasked to design a software which would accept two Finite State Machines (FSM) and determine if the both of them are considered equivalent. Among the three types of FSMs discussed in class, the formatting of the machines to be used as input for this project will be based on Mealy machines, in which that outputs are assigned to transitions.

As discussed in class, two finite machines are considered equal if they either produce the same output for every input string, or when both machines accept or reject the same strings. The equivalence of two finite machines are formally defined from the given handout as follows:

“**Definition**: Two machines M1 and M2 are equivalent if and only if for each stimulus, M1 and M2 produce identical responses, i.e. if s1(t) = s2(t), then r1(t) = r2(t), for all t ≥1.

*If M1 and M2 are equivalent machines, we write M1 ~ M2.*”

Keeping the definition in mind, this project will make use of MP01 in order to determine which states of both machines are considered equivalent by reducing both of the machines into a single resulting machine. Once reduced, it should be tested that the initial states of both machines should be in the same blob of a partition, or in other terms equivalent, for both machines to be considered equivalent (Hopcroft et al, 2006).

Overall this project aims to automate the process of determining the equivalence of two given Finite State Machines.

1. **SOFTWARE DESIGN**
   1. **Inputs**

As the project aims to determine the equivalence of two Finite State Machines, the inputs to be accepted will be both of the machines’ corresponding components, namely:

* Q = set of states
* S = set of inputs
* R = set of output
* Transition Table = set of transitions which determine where a state goes given an input and its corresponding output.

Each respective machine’s name will be included as an additional attribute in order to differentiate their corresponding states and components during the reduction process.

For the user’s ease of access, the inputs will be saved in a text file, which follows the same formatting of the listed components of the machine. This is done in order to avoid the repetitive process of asking the user to type in the necessary components every single time the user would want to run the system. Since two machines will be compared, two separate text files will be used as inputs as well.

Sample input text file components:

Machine 1

A,B,C,D,E,F,G,H,I

0,1

0,1

A,B:0,C:0

B,C:1,D:1

C,D:0,E:0

D,C:1,B:1

E,F:1,E:1

F,G:0,C:0

G,F:1,G:1

H,I:1,B:0

I,H:1,D:0

It should be noted that the first state transition listed in each of the machine’s transition tables are considered as each respective machine’s initial state.

* 1. **Algorithm**

In order to check the equivalence of both machines, both of machines’ components will first be combined into a single new machine, which will then be reduced. This is done in order to determine the equivalent states of machine 1 in machine 2.

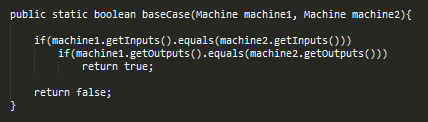
The algorithm used in the system for reducing the number of states of the combined machine is discussed by Hopcroft (1971), which is based on the Partitioning Refinement Algorithm.

The steps and implementation of checking the equivalence of both machines and the Partitioning Algorithm are as follows. Note that the code for the Partitioning Algorithm can be referred from the documentation of MP01 and will no longer be included in this paper.

* + 1. **Combine the components of M1 and M2**

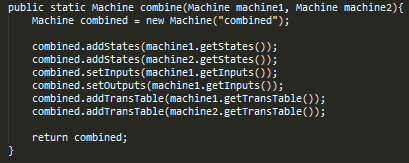
Prior to combining the components of M1 and M2 into a new machine and reducing them, it should first be checked whether or not both machines have the same set of inputs and outputs. Otherwise, both machines might not produce the same response for a given input string which goes against the definition of equivalent machines.

The checking of both machines’ inputs and outputs can be considered as a base case before performing any of the partitioning and equivalence processes.



*Figure 2.1. Code for checking the equivalence of the inputs and outputs of two given machines*

Once the set of inputs and outputs of both machines are considered equivalent, the components of both machines can now be combined into a singular machine.



*Figure 2.2. Code for combining two given machines*

* + 1. **Form an initial partition P1 of Q by grouping together all states that are 1-equivalent**

For the next step, an initial partition is performed where every listed state in the newly combined machine is assigned to a group, called a “Blob”, depending on the state’s corresponding set of response outputs. If the group does not exist yet, a new group will be generated. If the group already exists, then the state will be added to that group. Each newly generated group is assigned a “Label” in order to differentiate it from the other groups for the succeeding partitions to identify.

After sorting the states into their corresponding groups for the initial partition, each state will then be assigned a new transition table as according to the newly assigned groups of their corresponding response states. This will be used as reference for the succeeding partitions to identify whether the newly created partition is equivalent to its previous.

Given the sample machine listed in *Section 2.1.*, its corresponding initial partition will contain:

* Blob #1: A, C, F
* Blob #2: B, D, E, G
* Blob #3: H, J
  + 1. **Obtain Pk+1 from Pk - Refinement of Pk.**

For this step, succeeding partitions will be generated for as long as previous partitions still contain blobs with states of differing response outputs. The response outputs for the corresponding response states of every state will be based on the labels of the newly created groups. Using the machine listed in *Section 2.1.* as an example, State A’s response states are B and C, and their corresponding response outputs are originally 0 and 0. After performing the initial partition, the response outputs of the response states B and C will change depending on which group each of them has been assigned, which are groups 2 and 1 in this case.

Considering the newly created transition table assigned to the sample initial partition listed in *Section 2.2.2.*, which is listed as follows:

* Blob #1:
* A, B:2, C:1
* C, D:2, E:2
* F, G:2, C:1
* Blob #2:
* B, C:1, D:2
* D, C:1, B:2
* E, F:1, E:2
* G, F:1, G:2
* Blob #3:
* H, J:3, B:2
* J, H:3, D:2

It is observable that there are groups with states who have differing sets of response outputs. Once this has been identified, a new partition will be made as according to the previous partition’s blob groups and transition table, which is the initial partition in this case, and will be compared whether or not they are equal.

* + 1. **Repeat Step 2 until Pm+1 = Pm for some m – Final Partition of Q.**

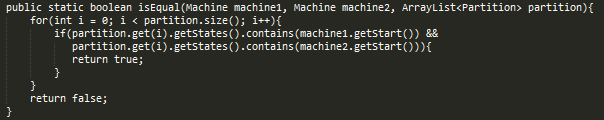
The third and final step of the Partitioning Algorithm is where the iteration takes place. The entire partition reduction algorithm is enclosed within a while condition in which it checks whether the newly created partition is equivalent to its previous both in terms of blob groups and transition table.

The original function used for the comparison is Java’s own ArrayList.equals() function, but in order to represent the manual checking of both partitions’ components, a custom function is used as a placeholder.

The code simply checks if both partition’s label are the same; if it’s corresponding groups/blobs have the same components, such as the states and their corresponding response states and outputs; and finally, if each of their states corresponding transition tables are the same as according to the previous partition.

* + 1. **Testing the equivalence of the initial states of M1 and M2**

The final step after reducing the combined machine is checking if the initial states from M1 and M2 are within the same blob of a partition as this determines whether or not they are equivalent, which in turn proves that their respective machines are equivalent as well.

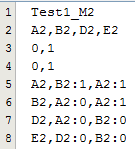
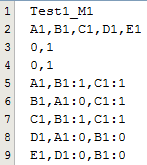
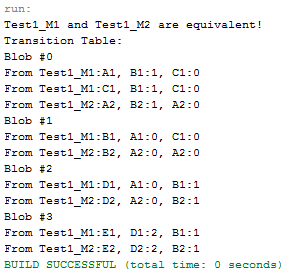


*Figure 2.3. Code for checking if a blob contains both of the initial states of M1 and M2*

* 1. **Output**

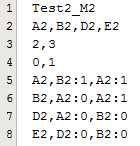
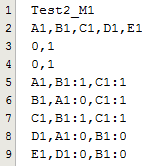
The system outputs a statement whether the two given machines are equivalent or not, along with the final partition and the corresponding groups of equivalent states of the combined components of the two machines.

1. **TEST CASES**
   1. **Equivalent machines, equal inputs / outputs**

**** **

*Figure 3.1. Test Case 1 Inputs**Figure 3.2. Test Case 1 Output*

* 1. **Non-equivalent machines, unequal inputs**

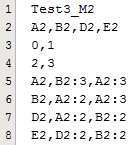
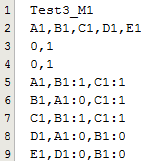
**

*Figure 3.3. Test Case 3 Inputs*

**

*Figure 3.4. Test Case 3 Output*

* 1. **Non-equivalent machines, unequal outputs**

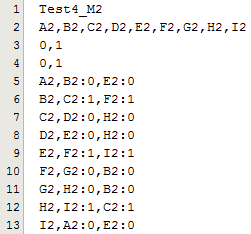
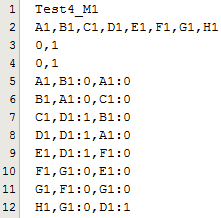
****

*Figure 3.5. Test Case 3 Inputs*

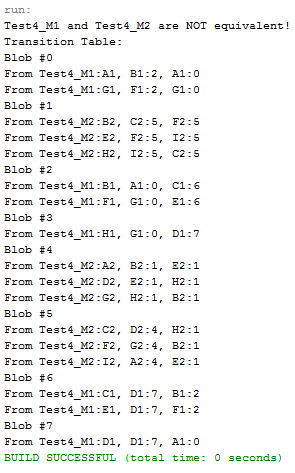
**

*Figure 3.6. Test Case 3 Output*

* 1. **Non-equivalent machines, equal inputs / outputs**

**

*Figure 3.7. Test Case 4 Inputs*

****

*Figure 3.8. Test Case 4 Output*

1. **CONCLUSION**

Overall, the system was successfully implemented. As proven by the test cases of varying scenarios, the system can successfully determine the equivalence of two given Finite State Machines.

1. **REFERENCES**

[1] Hopcroft, John (1971), "An n log n algorithm for minimizing states in a finite automaton", Theory of machines and computations (Proc. Internat. Sympos., Technion, Haifa, 1971), New York: Academic Press, pp. 189–196.

[2] Hopcroft, John, Motwani, R., and Ullman, J.D., (2006), Introduction to Automata Theory, Languages and Computation 3rd edition, Addison-