**CSC615M - Machine Project 1 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 first learning output for the course of CSC615M – Automata Theory, Computability, Formal Languages, the students were tasked to design a software which would accept a Finite State Machine (FSM) and determine which of its states are considered equivalent. Among the three types of FSMs discussed in class, the formatting of the machine 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, FSM’s may be reduced to a minimized form by decreasing the number of states within the machine by combining states which are considered as equivalent in order to minimize costs. In order to do this, one must first determine which states are equivalent in a given finite state machine.

Two states are considered equivalent as according to these definitions discussed in class:

“**Definition**: Two states qa and qb of a Mealy machine M = (Q, S, R, f, g, qI) are equivalent states if and only if the machines Ma = (Q, S, R, f, g, qa) and Mb = (Q, S, R, f, g, qb) are equivalent.

If qa and qb are equivalent states, we write qa ~ qb.”

“**Theorem**: States qa and qb of a Mealy machine M = (Q, S, R, f, g, qI) are equivalent if and only if:

**a)** For all s ∈ S, g (qa, s) = g (qb, s).

**b)** For all s ∈ S, f (qa,s) ~ f (qb, s).”

Keeping these definitions in mind, this project aims to automate the process of reducing given FSAs.

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

As the project aims to output the equivalent states of a given Mealy Machine, the inputs to be accepted will be the machine’s corresponding components, namely:

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

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, 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.

Sample input text file components:

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

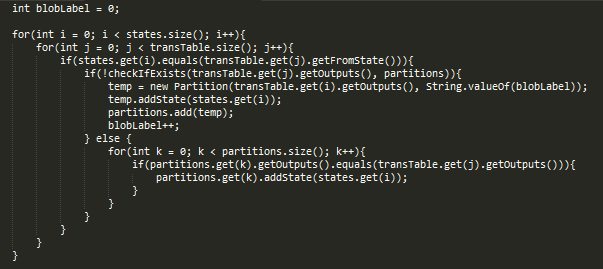
* 1. **Algorithm**

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

The steps of the Partitioning Algorithm and their corresponding implementation in the system are as follows:

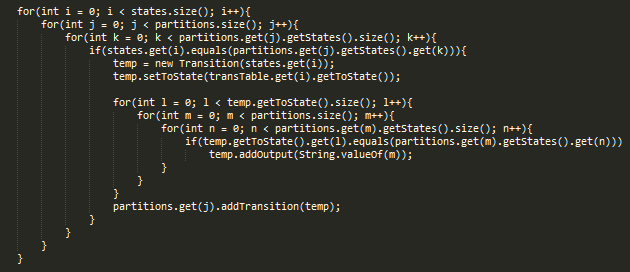
* + 1. **Form an initial partition P1 of Q by grouping together all states that are 1-equivalent**

For the first step, an initial partition is performed where every listed state in the 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.



*Figure 2.1. Code for the Initial Partition*

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.



*Figure 2.2. Code for Setting the New Transitions*

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

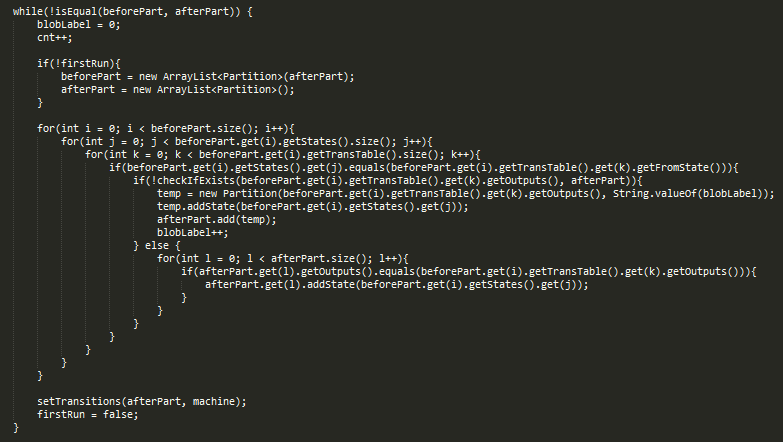
* Blob #1: A, C, F
* Blob #2: B, D, E, G
* Blob #3: H, J
  + 1. **Obtain Pk+1 from Pk - Refinement of Pk.**

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 2.2.1.*, 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.

**

*Figure 2.3. Code for Reducing the Previous Partition*

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

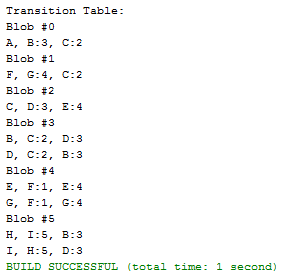
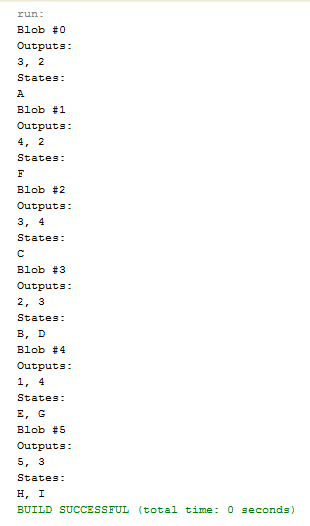
The third and final step of the Partitioning Algorithm is where the iteration takes place. As seen in *Figure 2.3.,* the entire partition reduction algorithm is enclosed within a while condition in which it checks whether the newly created partition is equivalent to its previous both in terms of blob 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. **Output**

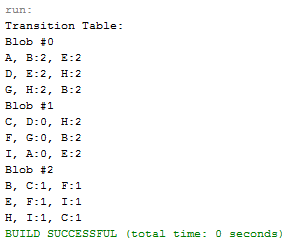
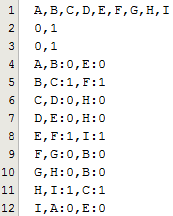
The system outputs the final partition and the corresponding groups of equivalent states of a given Mealy Machine. Using the machine listed in *Section 2.1.* as an example, it’s corresponding output once it is run in the system is seen in *Figure 2.5.*



*Figure 2.5. Final Partition Figure 2.6. Transition Table*

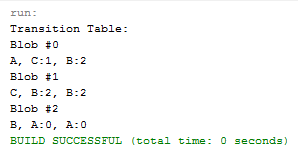
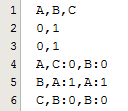
The corresponding transition table, as seen in *Figure 2.6.*, of the final partition can be viewed as well in order to prove the succeeding partition, if generated, will also contain the same components.

1. **TEST CASES**
   1. **Mealy Machine with No Unique States**

****

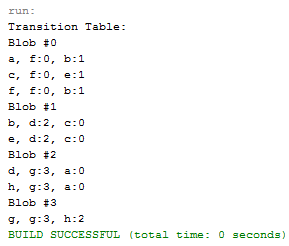
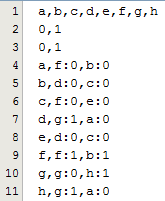
*Figure 3.1. Input Figure 3.2. Output*

* 1. **Mealy Machine with no Equivalent States**

****

*Figure 3.3. Input Figure 3.4. Output*

* 1. **Mealy Machine with Unique and Equivalent States**

****

*Figure 3.5. Input Figure 3.6. Output*

1. **CONCLUSION**

Overall, the system was successfully implemented. As proven by the test cases of varying scenarios, the system can successfully determine the equivalent states of a Mealy Machine FSA.

By testing a machine with no unique states, the system should be able to determine if all of the states belong to a group with more than one equivalent state. By testing a machine with no equivalent states, the system should be able to determine if all of the states belong to their own groups by themselves. And finally, by testing a machine with contains a mix of both unique and groups of equivalent states, the system should be able to properly form each state’s corresponding grouping.

This system will also be useful for the succeeding machine project as it involves determining whether two machines are equivalent.

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.