Finite State Machine Minimization and Row Equivalence Application

Hassan Farhat
University of Nebraska at Omaha

Abstarct- *Finite state machines minimization (finite automata) is a well known problem in formal languages and computer design. For a given automata, effective procedures exist that converts the automaton to a unique equivalent automaton but with possibly fewer number of states.*

While several minimization procedures are devised, we show that some of the procedures do not yield the minimum automata. In particular, we show that, contrary to assumptions, the row equivalence method of minimization does not always yield the unique minimum. We explore the three common methods of minimizations. We then look at the need to employ minimization procedures other than row equivalence to achieve a unique minimum automaton.

1 Introduction

Finite automaton is fundamental in the study of the fields of computer science and computer engineering. It is found in the study of software algorithms as in the case of compilers [2, 3]. And it is important in the design of hardware units in computer engineering [1, 7].

In the study of formal languages, the languages recognized by finite automata are called regular languages. The languages are closed under the set operations: Kleene closure, union, complement and intersection. In addition, given the finite state machines, FSMs, of two regular languages, the automata for the union, complement, and intersection can be easily constructed (an effective procedure exists for the construction) [2]. Effective procedures exist, as well, that reduce the number of state (nodes) in an FSM. For a given FSM there exist a unique equivalent FSM with minimum number of states [2, 3].

In hardware realization, FSMs are mapped to Mealy or Moore machines [7]. In a Mealy machine, the output of the machine is associated with an input and a node (state) while in a Moore machine, the output is associated with a node only. The two machines are equivalent; a procedure exists that maps one machine to the other. In [8] we showed that a minimum automaton does not always yield the optimal hardware design and discussed automata partition [9].

While several procedures are proposed that yield minimum automata [4,5,6], we show that some do not always yield the minimum automaton. In particular we show the row method of minimization as presented in [4, 6] does not yield always a minimum. In [4], the row equivalence algorithm is expanded to include self-loops as a method of minimization. We show that even with this expansion, for some examples, the method does not yield minimum automata.

The paper is organized as follows. In section 2, we present two common methods of automata minimization. In section 3 we look at the row method of minimization. Section 4 presents a counter example and shows that other methods of minimization yield a smaller automaton. The conclusion is given in section 5.

2 Regular Languages and Corresponding Finite State Machines

The definitions of regular languages are found in [2,3][1]. The definition is based on the construction of regular expressions. The regular expression generates (describes) set of elements (words).

A finite automata, also called finite-state machine, M, is a recognizer for a regular language; i.e., for a given word, M will determine if the word is in the regular language. Formally, M, is defined as a quintuple $M = (Q, \Sigma, q0, \delta, A)$ where Q is a finite set of symbols (states), q0 is a special element of Q called the start state, A is a subset of Q (accepting states), Σ is a finite set of symbols (the alphabet), and δ is a function from $Q \times \Sigma$ to Q (the next state function).

The finite-state machine can be described as a directed graph or in tabular form. Example: consider the language over the alphabet {0, 1}. A word in the alphabet is any binary string of 0s and 1s. The language, L, where each word ends in two 0s is a regular language. Figure 1 gives the recognizer finite automat (automata), M, as a directed graph.

[1] This section is presented in [8]

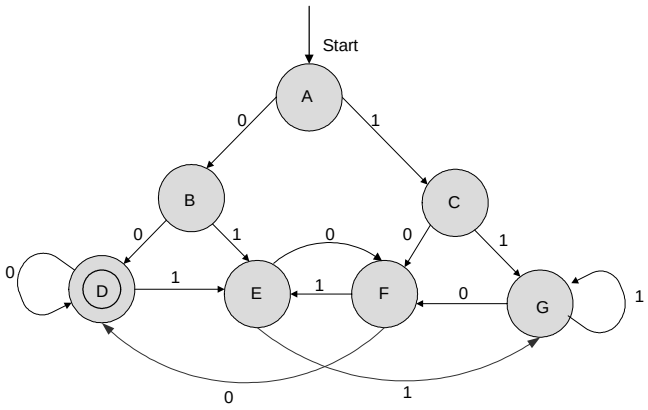

Figure 1: accept states have double circles, start state is labeled with start arrow

In hardware design, we describe the automata a little different. The automata can be described as a Mealy or Moore machine. The accepting states are removed from the definition of the automata. Instead, the machine is presented as a hardware block with inputs and outputs with states assigned binary codes and represented as memory elements. The conversion to Mealy or Moore is simple. Figure 2 shows the Moore conversion. Accepting states have an output of 1 while non-accepting states have an output of 0. For a given input (processed completely), if the final output is 1, the input is a word in the language. An output of zero means the input word is not the language.

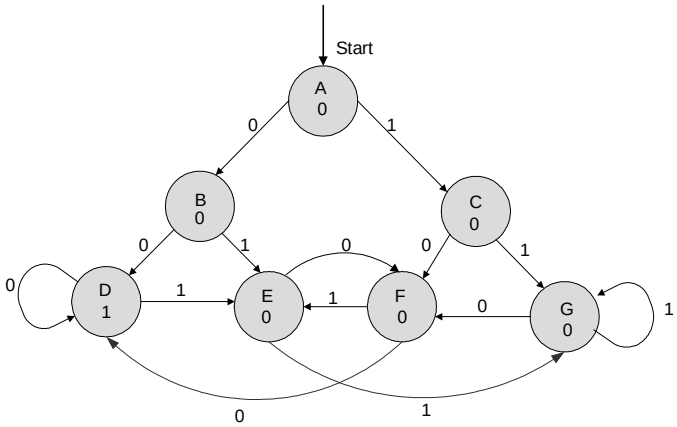

Figure 2: Moore realization, with each state we assign an output (bottom label) ; state D is accepting state (output 1 when in state D)

There are two main minimization procedures that produce the minimized automata: a) the partitioning method, and (b) the implication method [2]. We apply these to our example. The modified tabular representation is shown in Figure 3.

PS	X = 0	X = 1	X = 0, 1
A	B	C	0
B	D	E	0
C	F	G	0
D	D	E	1
E	F	G	0
F	D	E	0
G	F	G	0

Figure 3: Moore tabular form of automata, it includes the output column corresponding to present state

The partition method: Iteratively, place states in groups (partitions) based on the output response from a given state, call the partitions Pi, i corresponds to the iteration number. The 0 iteration contains all the automata states in a single group. P1 places the states in two groups based on the output; those with output zero and those with output 1. P1 : (A, B, C, E, F, G)(D).

Starting with P1 we form new groups based on next states on input 0 and input 1, call these 0-successor and 1 successor respectively. 0-successor(A, B, C, E, F, G) = (B, D, F, D, F). This part of the partition is split if the successor states belong to different previous groups. Split B and F since the next state is D; D and other successor states are in different previous groups. New P2 partition is P2: (A, C, E, G)(B, F)(D). To complete P2 we also form the 1-successors; 1-successor(A, B, C, E, F, G) = (C, E, G, G, E, G); no new groups formed.

Iteratively, we repeat the above to generate P3. The stopping criteria is a point where P(i+1) = Pi. When P3 is formed we find P2 = P3. The modified minimum automata contain three states (A, C, E, G), (B, F) and D.

The implication method of minimization: This procedure applies to incompletely specified automata as well (an automata where the transition function, δ, is not completely defined over the alphabet domain). It is based on a table construction as follows. The table contains columns for every state in the table except the last state; it also contains rows for every state except the first. For a column with label Sj and row Si, the corresponding entry is either X or two pairs of states. The entry is X if the outputs of Si and Sj are different. The table entries are the pairs (Snj0, Sni0) and (Snj1, Sni1) (next state of states Si and Sj on input 0 and 1 respectively). When this is applied to the original automata we obtain the table shown in Figure 4(a).

Pair of states with Xs are not equivalent. From this table we form additional tables by adding new Xs. The stopping criteria: when the new table generated is the same as the previous table. To add Xs, we examine each pair for states entries. We place an X if the entry contains a pair with an X entry already generated in the table.

Lets look at the above example, the first entry checked corresponds to states (A, B) column A and row B. This entry contains the pair (B, D) and (C, E). Since for the entry (B, D) in the current table, Figure 4(a), an X is

present, in the new table construction (Figure 4(b)), we place an X in entry (A, B).

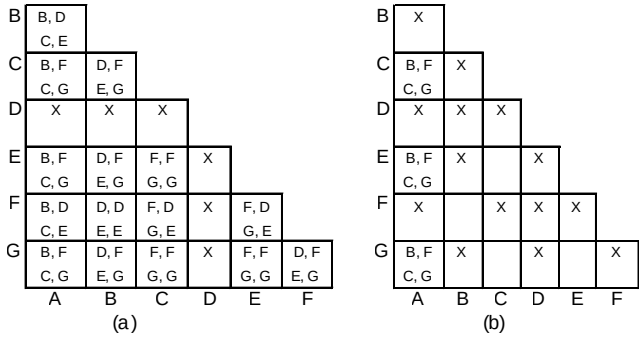

Figure 4: Implication method tables

Similar reasoning applies to other entries to obtain Figure 4(b). In the figure pairs of the form (x, x) are removed since it is not possible to replace them with Xs.

On starting with Figure 4(b) and repeating the iteration process, the new table generated is the same as Figure 4(b). Hence, the process stops.

The remaining entries with no Xs represent states that are equivalent. Hence states (A, C), (A, E) (A, G) (B, F) are equivalent. The automata minimization forms an equivalence relation, hence states A, C, E, F and G are equivalent. The final partition is (A, C, E, F, G)(B, F)(D). This is the same grouping obtained using the partition procedure above.

3 The row equivalence method of minimization

The row equivalence method of minimization relies on identification of equivalent rows in a state table as given in Figure 3. It is based on the following definition [6].

Definition: In a state table, two state are said to be equivalent if: a) the next states rows are identical, and b) if the output is the same from each state on each input combination.

In [4], the definition for part (a) was modified so as to allow for cycles in the state diagram. For two states A and B, with identical outputs on all input combinations starting in either state, the two states are equivalent if either or both properties 1 and 2 are satisfied. Property 1: the successor of state A on input x is A, and the successor of state B on input x is B (each state has an edge that loops back to itself on input x). Property 2: If on some input x the successor of state A is state B and the successor of state B is state A.

On repeated identification of equivalent states and reducing the table by keeping only one state from each class of equivalent states, the minimum can be obtained. We apply the method to the example state diagram above. From Figure 3, we note that state B and F are equivalent and states C, E and G are equivalent. On removing states F, E and G we obtain Figure 5.

	NS		OUTPUT
PS	X=0	X=1	X=0, 1
A	B	C	0
B	B	E	0
C	B	C	0
E	B	C	1

(A, C) Equivalent

Figure 5

Note that the states of the table are renamed in cases that reference states that are removed (example row with present state C).
By inspection of the table, we see that states A and C are equivalent. By removing state C we obtain the reduced table found in Figure 6.

	NS		OUTPUT
PS	X=0	X=1	X=0, 1
A	B	A	0
B	B	E	0
E	B	A	1

Rename E as C

Figure 6

On renaming state E, we obtain the minimum automaton, as was done using either method of minimization in the previous section.
We next show that even though the row equivalence method works for the previous example, it does not always yield a minimum. 5

4 A Counter Example

We consider the state table shown in Figure 7. By inspection of the table, there are no identical rows. We then look at two rows that satisfy properties 1 or 2 as stated in section 5. On input x = 0, rows D, E and F have loops (next state = present state). However, the next states are different on input x. Hence property 1 fails. Note that state N is not considered since the output from state N is 1.

	NS		OUTPUT
PS	X = 0	X = 1	X = 0, 1
A	F	B	0
B	E	C	0
C	D	A	0
D	D	M	0
E	E	L	0
F	F	K	0
K	N	C	0
L	N	B	0
M	N	A	0
N	N	N	1

Figure 7

We then check to see if property 2 holds true. For this property to hold true, we need to have two rows i and j (present states i and j) such that on some input x the next state from i is state j and the next state from j is state i. In addition, the remaining elements of the rows should be identical. By inspection, we find this property does not hold true, as well.

Based on the row equivalence method then, we conclude that the above table is already in reduced form. We next show this is not the case by forming a smaller automaton based on the partition method of minimization.

We use the partition method to show the minimum automaton yields fewer states than concluded in the row equivalence method. The 0 iteration contains all the automata states in a single group, P0 = (A, B, C, D, E, F, K, L, M, N). P1 places the states in two groups based on the output; those with output zero and those with output one; P1 = (A, B, C, D, E, F, K, L, M) (N).

From the P1 partition we look at 0-successor(A, B, C, D, E, F, K, L, M) = (F, E, D, D, E, F, N, N, N). Hence the new partition due to 0-successor is (A, B, C, D, E) (K, L, M) (N). The 1-successor(A, B, C, D, E, F, K, L, M) results in (B, C, A, M, L, K, C, B, A) which results in further partitions, P2 = (A, B, C) (D, E, F) (K, L, M) (N). From P2, when we consider the 0-partitions of (A, B, C), (D, E, F), and (K, L, M), we respectively obtain, (F, E, D), (D, E, F) and (N, N, N); no new partitions are generated. Similarly we consider the 1-partitions of (A, B, C), (D, E, F), and (K, L, M). We obtain (B, C, A), (M, L, K) and (C, B, A); no new partitions are generated. Since P3 = P2, the partition in P2 forms the minimum automaton with 4 states only.

5 Conclusion

In this paper we have considered a well known problem in formal languages; the problem of state minimization. While three algorithms are normally given to form the minimum automaton, we showed the row equivalence method does generally result in a minimum automaton. The row equivalence method can be chosen as a preprocessing step in the minimization. However, to guarantee minimum automata it needs to be appended by the implication method of minimization, or the partition method based on the state successor method.

References

[1] N. Weste and D. Harris. "CMOS VLSI design 3rd edition", Addison Wesley, 2004.

[2] J. Martin. "Introduction to Languages and the Theory of Computation". McGraw Hill, 1991.

[3] W. Barett, R. Bates, D. Gustafson, J. Couch. "Compiler Construction Theory and Practice, 2nd edition". SRA publishing, 1979.

[4] V. Nelson, H. Nagle, B. Carroll, J. Irwin. "Digital logic Circuit Analysis and Design". Prentice Hall, 1995.

[5] R. Katz and B. Gaetano. "Contemporary Logic Design, 2nd edition". Prentice Hall, 2005.

[6] M. Mano and C. Kime. "Logic and Computer Design Fundamentals, 3rd Edition". Prentice Hall, 2003.

[7] D. Gajski. "Principle of Digital Design". Prentice Hall, 1997.

[8] H. Farhat. "Minimization of State Diagrams and Realization at the Hardware Level". ICIAS, 2010.

[9] L. Benini, G. De Micheli, F. Vemzeulen. "Finite-State Machine Partitioning For Low Power", ISCAS, 1998.