MINIMAL VALID AUTOMATA OF SAMPLE SEQUENCES FOR DISCRETE EVENT SYSTEMS

Sheng-Luen Chung and Chung-Lun Li

ABSTRACT

Minimal valid automata (MVA) refer to *valid* automata models that fit a given input-output sequence sample from a Mealy machine model. They are *minimal* in the sense that the number of states in these automata is minimal. Critical to system identification problems of discrete event systems, MVA can be considered as a special case of the minimization problem for incompletely specified sequential machine (ISSM). While the minimization of ISSM in general is an NP-complete problem, various approaches have been proposed to alleviate computational requirement by taking special structural properties of the ISSM at hand. In essence, MVA is to find the minimal realization of an ISSM where each state only has one subsequent state transition defined. This paper presents an algorithm that divides the minimization process into two phases: first to give a reduced machine for the equivalent sequential machine, and then to minimize the reduced machine into minimal realization solutions. An example with comprehensive coverage on how the associated minimal valid automata are derived is also included.

KeyWords: Discrete even systems, minimization of sequential machines, minimal valid automata.

I. INTRODUCTION

System identification problems for discrete event systems (DES), defined in [2], refer to the inference of system model for an unknown system based on an externally observed input-output sequence. Figure 1 illustrates the basic system diagram of system identification for DEDS. The unknown DEDS is the target for system identification whose system dynamic is unknown to us. The Tester generates a test pattern of commands, as an input string $\alpha(k)$, to the unknown discrete-event dynamic system target, and then records the resulting output response sequence, as an output string $\beta(k)$, from the unknown target. We call the composed sequence as the input-output sample sequence $\omega(k)$, which records the inherent internal dynamic relation of the unknown system.

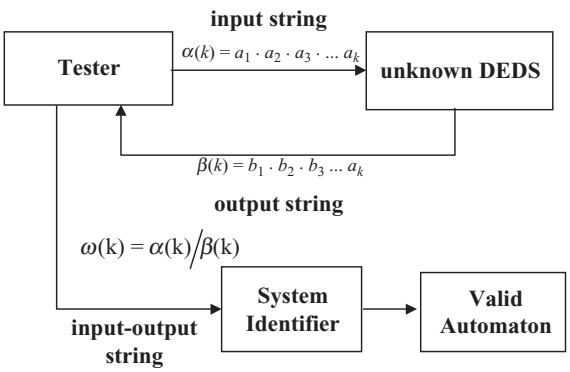


Fig. 1. System diagram of system identification for DEDS.

To best characterize the input-output relation of the observed sample path, [2] has proposed a new mathematical modeling paradigm, where the unknown target is modeled by a Mealy machine. The observed input-output sequence can then be considered as a sample path from the language generated by the underlying Mealy machine. Consequently, the identification problem is reduced to finding the underlying automata structure that generates the observed input-output sample path.

In effect, solution to the system identification problem, as a set of all possible model candidates, cor-

Manuscript received May 27, 2003; accepted October 23, 2003.

The authors are with Department of Electrical Engineering, National Taiwan University of Science and Technology, Taipei, Taiwan 106, R.O.C.

The research was supported in part by the grant NSC90-2213-E-011-020 by the National Science Council, Taiwan.

responds to *valid automata* that fit the externally observed input-output sequence sample. In particular, we are to find *minimal* valid automata (MVA) that are *minimal* in the sense that the number of states in these automata is minimal.

The problem of finding minimal valid automata can be considered as a special case of the minimization problem for incompletely specified sequential machine (ISSM), where transitions by some inputs lead to unspecified states or unspecified outputs. In the context of MVA, we are to find the minimal realization of an ISSM where each state only has one subsequent transition defined. While the minimization of ISSM in general is an NP-complete problem [4], various algorithms have been proposed that take advantage of the particular structure property of the underlined ISSM. These state minimization methods in general are composed of the following three steps:

(1) the derivation of all related compatibles;
(2) the reduction of the number of compatible classes to be considered;
(3) selection of a minimal closed covering from compatible classes.

Of these three steps, the third step of selecting a minimal close covering is most tedious. Given an ISSM, Kohavi [7] proposed a method that finds all maximal compatible with merger table, and uses compatibility graph to search all closed coverings. Then, the minimal closed coverings that correspond to the minimization solution are obtained by a "trial-and-error" procedure.

By taking the special property of a sequential machine where only one state transition is defined for each state other than the last state, Kella [8] proposed a method that improves upon the second step above by avoiding the generation of a complete set of maximal compatibles by adding new states recursively for sequential Mealy machines. Rao and Biswas [9] applied some deletion rules to the set of compatible classes in obtaining a relatively small set of symbolic compatible, from which a minimal solution was then obtained. Yamamoto [10] presented a method for obtaining a minimum form of a given ISSM by application of a directed tree graph. Avedillo *et al*. [11] proposed heuristic algorithm to build a closed covering for a given state table selecting maximal compatibles one by one until both covering and closure requirements are satisfied. Near-minimal solutions are thus incrementally generated. Puri [12] employed a tight lower bound and an ordered generating sequence to create a smaller search space of compatible sets. Efficient pruning criteria are developed to reduce further the search space. The first path on the search tree satisfying the covering and closure constraints is guaranteed to be a minimal FSM solution.

This paper presents an algorithm that divides the minimization process into two phases: one to give a reduced machine for the equivalent sequential machine, and then another to minimize the reduced machine into minimal realization solutions.

MVA is important not only to the solution of system identification problem, it is also important to the characterization of the complexity of information embedded in a sample path, as well as to the online modeling refinement problem and [3]. In this regard, this paper serves as a tutorial on the solution of Minimal Valid Automata by a detailed algorithm and a comprehensive coverage of an example.

Organization of this paper

The rest of this paper is organized as follows: Section II provides a background on the mathematical modeling of system identification of DES, and on the minimization of incompletely specified automata. Section III details on the proposed Algorithm of Minima Valid Automata, which is a revised minimization algorithm based on Kella's algorithm [8]. The derivation of a minimal closed covering is now formalized instead of being done by trial-and-error. Section IV is devoted the comprehensive derivation of minimal valid automata of an example step by step. Section V is the conclusion.

II. MATHEMATICAL PRELIMINARY

In this section, we first introduce some of the notation and definition from the system identification of DEDS. Then, we will also summarize key ideas about minimization of incompletely specified automata.

2.1 System identification for DES

A Mealy machine M is defined as a six-tuple: $M = \{I, O, S, \delta, \lambda, s_0\}$, where I is the finite set of input events; O is the finite set of output events; S is the finite set of states; $\delta : S \times I \to S$ is a state transition function; and $\lambda : S \times I \to O$ is an output function; and S_0 is the initial state. Given a Mealy machine $M = \{I, O, S, \delta, \lambda, s_0\}$ and an input sequence $\alpha(k) = a_1 \cdot a_2 \ldots a_k \in I^*$, we define the input-output sequence as:

$$\begin{aligned}\omega(k) &= (a_1, \lambda(s_0, a_1)) \cdot (a_2, \lambda(s_1, a_2)) \cdots (a_k, \lambda(s_{k-1}, a_k)) \\ &= (a_1, b_1)(a_2, b_2) \cdots (a_k, b_k) \\ &= a_1/b_1 ; a_2/b_2 ; \cdots ; a_k/b_k .\end{aligned}$$

Collectively, $\mathcal{L}(M)$ is the collection of all possible sample paths from the Mealy machine M. In [2], system identification problem of DES is defined as the solution of valid function defined below.

Definition 1. The *Valid Function* V is a function from $\mathcal{L}(M)$ to $2^{\mathcal{G}}$, where $\mathcal{G}$ is by definition the set of all Mealy machines. V will map any $\omega(k)$ in $\mathcal{L}(M)$ to the set of all Mealy machines G where $\omega(k) \in \mathcal{L}(G)$, i.e., $V(\omega(k)) = \{G \in \mathcal{G} : \omega(k) \in \mathcal{L}(G)\}$.

In words, valid automata $V(\omega(k))$ derived by valid function fit the input-output sample sequence in the sense when an input sequence $\alpha(k)$ is fed into any of the automaton in the set of Mealy machines $V(\omega(k))$, we should get the exact output string $\beta(k)$ as observed. Despite all the nice theoretic properties, the solution of *Valid Function* V in general contains infinite number of automaton candidates, that is $|V(\omega(k))| = \infty$, thus precluding the possibility of a computational result.

Alternatively, we are to find a limited version of Valid Automaton, which is called *Minimal* Valid Automata (MVA). The function is called *minimal* in the sense that the number of states for the automata derived from MVA is minimal among all valid automata. As a result of limiting legitimate solution constrained by minimal number of states, the set of minimal valid automata is finite. Before we present the formal definition for minimal valid automata, we need the following definition of *order*.

Definition 2. Let $\omega(k)$ be an input-output sequence. The *order* of the input-output sequence, written as *order* $(\omega(k))$, is defined as follows:

$$order(\omega(k)) := \min_{G \in V(\omega(k))} \|G\|$$

where $\|G\|$ denotes the cardinality of G, i.e., the number of states of G.

In words, the *order* that derived from a sample path of an unknown target can be considered as the lower bound of the cardinality of the unknown target.

Definition 3. The *Minimal Valid Function* V^m is a function from $\mathcal{L}(M)$ to $2^{\mathcal{G}}$, where $\mathcal{G}$ is by definition the set of all Mealy machines. V^m maps any $\omega(k)$ in $\mathcal{L}(M)$ to the set of all Mealy machines G where $\omega(k) \in \mathcal{L}(G)$ and the number of states of G is minimal, that is, the number of states is equal to $Order(\omega(k))$. In an equation, that is,

$$V^m(\omega(k)) = \{G \in \mathcal{G} : \omega(k) \in \mathcal{L}(G), \|G\| = order(\omega(k))\}$$

Properties of $V^m(\omega(k))$ are summarized as follows: Let $\omega(k)$ be a sample path from a Mealy machine M,

(i) $V^m(\omega(k)) \neq \emptyset$,
(ii) $|V^m(\omega(k))|$ = is finite,
(iii) $M \notin V^m(\omega(k))$ in general,
(iv) $V^m(\omega(k+1)) \subsetneq V^m(\omega(k))$ in general.

Properties (i) and (ii) ensure that minimal valid automata exist and the number of solution is finite. Property (iii) says that the model of the unknown target may not be in the minimal valid automaton set. This puts more constraint on the selection of a *good* sample path $\omega(k)$ if the target M is to be derived. Property (iv) indicates that there is no direct relation between the minimal valid automaton sets of successive sample paths, thus requiring more computation when identification process is conducted incrementally.

2.2 Minimization of incompletely specified sequential machines (ISSM)

In essence, the computation of minimal valid automata belongs to the problem of minimization of sequential machines. Depending on the sequential machines in question, the minimization process can be very different in the number of the resultant *solution* and the computational complexity required to derive the answer. For completely specified sequential machine, the solution is unique and the algorithm takes O (n log n) time [5] computational complexity. In contrast to the minimization of completely specified finite state machines where the notion of *equivalent* states is utilized, the minimization of incompletely specified sequential machines relies on the notion of *compatibility*. Equivalent relation implies a *partition* of the state space in resulting a unique minimal realization, whereas compatibility is not an equivalent relation. As a major consequence of this difference, the minimization of incompletely specified machines does not result in a unique solution in general, and the associated computational complexity is NP-complete [6].

In the following, we summarize on the key ideas for the minimization of ISSM, which relies on manipulations on collections of compatibles, called closed sets. To explore closed sets, one must compute maximal compatibles, prime compatibles, class sets of compatibles, and other related sets and subsets of sets of states. We will use an explanatory example to highlight these ideas.

Example 1. Given an incompletely specified sequence machine, whose state transition can be represented by the transition table as shown in Table 1 on next page.

Two states S_i and S_j are said to be *compatible*, if and only if, for every possible input sequence applicable to S_i and S_j, the same output sequence is produced, regardless of whether S_i or S_j is initial state. If the output sequences are different, then S_i and S_j are called *incompatible*. For the above example, states D and E, and states C and D are compatible, while states A and B, and states B and D are incompatible.

Table 1. Example of an incompletely specified sequential machine.

PS	NS, Z	
	Input = 0	Input = 1
A	A, 0	D, 1
B	E, 0	D, 0
C	–, –	B, 1
D	E, 0	B, 1
E	C, 0	E, –

Table 2. Merge table for Example 1.

B	×			
C	×	×		
D	×	×	√	
E	×	(C,E)(D,E)	(B, E)	(C,E)(B,E)
	A	B	C	D

A compatible class C_i *covers* compatible class C_j, if and only if, every state contained in C_j is also contained in C_i. For the example, one compatible class is (ABCD) and another compatible class is (AB). (ABCD) covers (AB).

There can be many compatible classes for a given machine, and a compatible class is said to be *maximal* if it is not covered by any other compatible class. Similarly, an incompatible class is said to be maximal if it is not covered by any other incompatible class.

Merge table [7] is used to find all the maximal compatibles for a given ISSM, as shown in Table 2 above:

For the given example, the maximal compatibles are: (A), (BE), (CDE), and Maximal incompatibles are: (ABC), (ABD), (AE).

A set of compatibles is *closed* if for every compatible contained in the set, all its implied compatible are also contained in the same set. For the example, the set of compatibles of {(A), (BE), (CDE)} is a closed compatible.

A set of compatibles (for machine M) is *closed* if for every compatible contained in the set, all its implied compatible are also contained in the same set. A closed set of compatibles which contains all the states of M is called a closed covering. For example, the set of compatibles of {(A), (BE), (CDE)} is a closed covering. Given an ISSM, there can be several closed covering. Each of them serves as a candidate of reduced automaton with the states in each compatible set being assigned as a new state in the reduced automaton. To find the minimization of the given ISSM, we are to find the closed coverings whose number of compatible sets is the smallest.

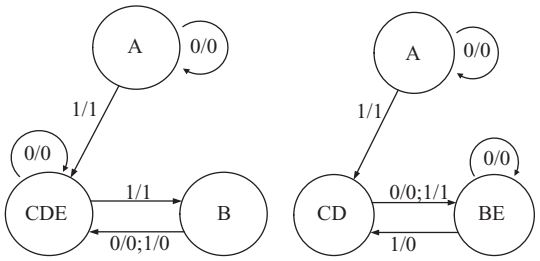


Fig. 2. Minimal realization of Example 1.

In general, the solution for the minimized automata may not be unique and requires trail-and-error. Below, however, is the upper and lower bounds for the number of states of the minimized automata: Upper bound: $min(k, n)$ where k is the number of states of machine, n is number of maximal compatibles. Lower bound: $max(q_1, q_2, \ldots, q_i, \ldots)$ where q_i is number of states of the ith maximal compatible.

For Example 1, the upper bound is $min(5, 3) = 3$, and the lower bound: $max(3, 3, 2) = 3$, thus we can conclude that the number of states of the minimization automata is 3. For this example, there are two minimal realizations as shown above in Fig. 2. The solution to the minimization problem for ISSM in general is not unique.

III. ALGORITHM FOR MINIMAL VALID AUTOMATA

The problem of minimal valid automata amounts to the problem of minimization of ISSM: the given input-output sample path can be modeled as an ISSM where each state, except the last one, only has one outgoing transition. By taking this special property, Kella [8] proposed a method that avoids the generation of a complete set of maximal compatibles by adding new states recursively for sequential Mealy machines. This method generates all possible reduced machines so that no machine with fewer states is overlooked.

By its nature the problem involves enumeration of solutions, Kella tried to reduce the enumeration as much as possible to derive a fast and efficient translation of input-output relations into the set of all minimization automata that fit the given input-output sample path. Enumeration is avoided unless there are a number of different solutions to the same problem and all of these solutions are equally valid and probable. In doing so, Kella's approach is to develop a fast state merging process for the specific type of incompletely specified machine defined by the input-output relation while all possible solutions are retained.

Despite the claimed advantage of state merging process that requires as least enumeration as possible in deriving the maximal compatibles, a trial-and-error process that tries out all combinations of maximal com

procedure Algorithm V^m

Input: input sequence ω(k)

Output: all valid minimal automata

Begin

$T_{ic}(\omega(k)) \leftarrow$ *Compatibility_check* $(\omega(k))$

for input symbol ***i*** first ***to*** last ***do***

$P_{i1}, P_{i2}, \ldots, P_{ij}, \ldots, P_{im} \leftarrow$ *Initial_blocks* $(i, T_{ic}(\omega(k)))$

for P_{i1} ***to*** P_{im} ***do*** /*augmenting states indexed by other input symbols*/

find all possible partitions P_{ij} of the total states by next block transitions where $P_{ij} = \{B_1, \ldots, B_k, \ldots, B_n\}$

$P_{co} = \{B_k \cup B_k', B_{-l}\} \leftarrow$ *Closed_covering* $(P_{ij}, \omega(k), T_{ic}(\omega(k)))$

minimal_valid_automata $V^m \leftarrow$ *Minimize_closed_covering* $(P_{co}, \omega(k))$

end partition of given input

end all input symbols

End Algorithm V^m

Fig. 3. Algorithm for the minimal valid automata V^m.

patibles to find all *minimal closed coverings* in deriving all the minimization automata is not dispensable. In this regard, we proposed a revised algorithm that takes into account of the upper and lower bond of the number of the states of final minimization automata. As a result, the enumeration of all combinations of maximal compatibles for minimal closed coverings is reduced.

Adapting Kella's approach, we develop the minimal valid automata algorithm V^m, where the maximal incompatible blocks are derived for *all* input symbols. Starting from each maximal incompatible block of merged states, we add other states into this block one by one, while maintaining compatibility of the block, until the set of augmented block constitute a *closed covering* of the given incompletely specified sequential machine. In our revised algorithm, valid automata with number larger than existing minimal valid automata are discarded, thus eliminating unnecessary trial-and-error computation.

Algorithm of V^m in pseudo code shown in Fig. 3 is composed mainly the following four steps:

(1) *Compatibility_check*: States of the sequential machine corresponding to the input sequence $\omega(k)$ is checked for compatibility.
(2) *Initial_blocks*: Compatible states for the same input symbol are grouped together as the initial blocks that will then include states corresponding to other input symbols. Initial blocks are considered for all input symbols.
(3) *Closed_covering*: Starting from the initial blocks, new states are to be included one by one. The closure property of the resultant merged states is checked and maintained to ensure a closed covering is derived. All possible combinations of merging states are checked against the input sequence $\omega(k)$ to ensure a valid closed covering.
(4) *Minimize_closed_covering*: For each of the valid closed covering, find the associated minimal realization. With the number of the merging states of the given closed covering reduced, the minimization takes less computation in general than would be required for the minimization of the sequential machine $\omega(k)$.

The procedure of *Compatibility_check* takes the input sequence of $\omega(k)$, and outputs the I/O state groups, the incompatibility listing table $T_{ic}(\omega(k))$, both of which are to be used later for the construction of *Initial_blocks*.

For the ***procedure*** of *Initial_blocks*, states for the same input are considered. Enumeration of all possible combinations of compatible states is carried out to yield all possible *partitions*.

Starting from each of the partition derived from the *Initial_blocks* procedure, the *Closed_covering* procedure is to add new states one by one until all states in $\omega(k)$ are included. In doing so, the next state transition for each state in the initial bock is derived according to input-output relation defined $\omega(k)$. Collectively, all the next state transitions form an initial block form a *next block*; states in the next block are to be checked for compatibility. In effect, if they belong to different originally blocks, meaning that closure property is not maintained, and the current block needs be split; otherwise, if the next block is a subset of an existing block, meaning that closure property is satisfied. Then process of generating next blocks continues until all states in $\omega(k)$ are included. At the end, blocks of merged states are indexed.

States that are not derived from the next block transition are indexed by consecutive negative integers. Collectively, these indexed blocks for a closed covering, which is to be minimized in the next procedure: *Minimize_closed_covering*.

The closed covering composed by the indexed blocks derived in the *Closed_covering* procedure is an incompletely specified sequential machinewhich is not minimized in general. Subsequently, the *Minimize_closed_covering* is to minimize the resultant ISSM whose states correspond to the indexed blocks that contain the merged states of $\omega(k)$. Rather then trial-and-error, our approach here is to enumerate all combinations of compatible states of the new ISSM. To eliminate the number of enumeration, only minimization candidates with state less then the upper bound derived in *Compatibility_check* and the Current_Order are considered. Minimization candidates are verified one by one by checking against the input sequence of $\omega(k)$.

We are now to give a detailed derivation steps on the algorithm of Minimal Valid Automata V^m.

3.1 Algorithm V^M: Minimal valid automata

Step 1. Given the input sequence $\omega(k)$, build the I/O state groups, and the incompatibility listing table and Merger table.

Step 2. Use the maximal incompatible sets obtained from the incompatible merge table to find the *lower_bound*, which is equal to the number of states in the maximal incompatible set.

Step 3. Initialize the output result: $V^m(\omega(k)) = \{\ \}$.

Step 4. *For* each input symbol i, generate all the maximal compatibles (MC's) for all the states defined under i.

Step 5. From the sets of MC's, form all possible partitions of the states defined under i. Each partition is then a set of compatible sets. Let the resulting partitions will be numbered P_{i1} through P_{im}, where i is input index, and m is the number of the generated partitions from each MC's.

Step 6. For each partition P_{ij}, find the associated next blocks as in the next step.

Step 7. Suppose $P_{ij} = \{B_1, \ldots, B_k, \ldots, B_n\}$. For each B_k in the partition P_{ij}, generate the associated next block N_{Bk} according to the mapping relation: given $B_k = [s_\alpha s_\beta \ldots s_\gamma]$ then, $NB_k = [s_{\alpha+1} s_{\beta+1} \ldots s_{\gamma+1}]$. In particular, note that s_n has no specified next state so that when it appears in B_k, no corresponding states appears in NB_k.

Step 8. Compare the NB_k derived above with all existing B_k, and do the following according which of the following three conditions (a), (b) and (c) meets:

(a) NB_k is a subset of an existing B_k, i.e., NB_k is completely included in any B_k: Delete NB_k from the NB_k list as it indicates that the closure condition on the originating B_k is satisfied.

(b) Some elements of NB_k already appeared in some block B_k:

(1) NB_k can be partially included in two different blocks B_k: This implies the closure condition is not met, and the associated $P_{ij} = \{B_1, \ldots, B_k, \ldots, B_n\}$ be dropped from further consideration, as it contradicts the closure condition, thus making it impossible for a closed covering.

(2) NB_k is partially included in a single B_k: Take the set difference of NB_k from B_k. It may happen that more than one next block is partially included in an existing B_k. Then, check all the set differences of the same subscript for compatibility. (a) If all of them are compatible, merge them all with B_k to form one single block B_k; (b) If the set difference sets are incompatible, this implies that the original block B_k has to split into two or more blocks. Then drop $P_{ij} = \{B_1, \ldots, B_k, \ldots, B_n\}$ from further consideration, as this might end up in a closed covering with number larger than other partitions.

(c) Elements in NB_k have never appeared in any B_k before; Retag NB_k as B_{-l} where l starts as 1 and counted up as new such cases are met. Here all states of NB_k are defined under other input symbol and closed conditions do not imply any connection to the block B_k.

Step 9. Repeat Step until all the states defined under the input symbol i are enumerated.

Step 10. Add all existing B_{-l} to the blocks $B_k \cup B_k'$ until all original states are included in the implied blocks. Since each of the blocks contains a set of states from the input sequence $\omega(k)$, we refer to each of the block a *mega state* hereafter. Use the input sequence $\omega(k)$ to build a reduced machine with states defined by the mega states. The problem now reduces to the minimization of the newly constructed state machine, which is in general an incompletely specified state machine. To do so, we are to construct the associated state transition table, merger table, and find the associated maximal compatibles of the reduced machines.

Step 11. By taking the upper bound and lower band on

the number of states in the minimized machine, we are to enumerate all closed coverings of the reduced machine derived in Step 10. In particular, Lower bound is equal to the largest number of states in the maximal compatible. We also keep track of Current_Order to be the number of sets of maximal compatibles those we have found so far, probably by other partition P_{ij}. For all closed covering with number of MC's being between Lower_bound and Current_Order, check its acceptance to the input sequence $\omega(k)$. If the closed covering can accept $\omega(k)$, it is considered as a minimization candidate.

Step 12. For all possible minimization candidates obtained above, build the corresponding state table and $V_\gamma^m := V_\gamma^m \cup$ {possible minimization candidate}, where γ is the number of state in the minimization candidate.

(a) ***IF*** $\gamma <$ Current_order, i.e., we can find solutions in smaller order, ***THEN*** do isomorphism for V_γ^m, let the new V_γ^m to replace the original V^m which we had found and update Current_order is equal to γ.

(b) ***IF*** $\gamma <$ Current_order, ***THEN*** do isomorphism for the union of V_γ^m and the original V^m.

(c) ***IF*** $\gamma <$ Current_order, it means that we can't find a better solution from Algorithm V^m.

Step 4 through Step 12 are done for each input symbol. Step 5 through Step 12 are done for each maximal compatibles for the states defined under symbol i. Step 6 through Step 12 are done for each initial block arrangement $P_{ij} = \{B_1, \ldots, B_k, \ldots, B_n\}$. Step 11 is done for all possible closed covering for the associated reduced machine.

IV. ILLUSTRATIVE EXAMPLE

With the algorithm detailed, an illustrative example is now in place with technical details carried out step-by-step. For simplicity of presentation, we assume the sample path taken from an unknown system of binary input symbols {0, 1} and binary output symbols {0, 1}.

Example 2. Given a finite-length input-output sample path $\omega(k)$ = 0/1; 0/0; 1/0; 1/1; 0/1; 1/0; 0/1; 1/1; 1/1; 0/1; 1/0; 1/1; 0/1, we want to find the associated minimal valid automata $V^m(\omega(k))$.

Solution: We first consider the given sample path as a sequential machine, as shown in Table 3, where only one state transition is defined for each state except the last one.

Table 3. Sequential machine for $\omega(k)$.

State	1	2	3	4	5	6	7	8
Input	0	0	1	0	1	0	0	1
Output	0	1	0	1	1	1	1	0

Table 4. I/O state groups table & incompatibility table.

Group	States
0/1	1
0/1	2, 4, 6, 7
1/0	3, 8
1/1	5

State	Incompatible States
8	–
7	–
6	–
5	8
4	7
3	5
2	4
1	2, 4, 6, 7

Step 1. Builds the I/O state groups table and the incompatibility table. From Table 4, the set of states defined under the input symbol "0" is {1, 2, 4, 6, 7}, and the set defined under the input symbol "1" is {3, 5, 8}.

Step 2. Given the whole set of states defined in Table 3, use the incompatible table to find Lower_bound of the order of $V^m(\omega(k))$. Lower_bound equals to the maximal number of states in the maximal incompatible set. What follows is the derivation for maximal incompatible sets.

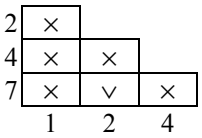
7: (7)(8)
6: (6)(7)(8)
5: (6)(7)(5,8)
4: (6)(4,7)(5,8)
3: (6)(4,7)(5,8)(3,5)
2: (6)(4,7)(5,8)(3,5)(2,4)
1: (1,6)(1,4,7)(5,8)(3,5)(1,2,4)

From above, the maximal number of states in the maximal incompatibles (1, 4, 7) and (1, 2, 4) is 3, therefore, Lower_bound = 3, which is the lower bound for the order of $V^m(\omega(k))$. This information would be used at Step 11 when we choose candidates for closed covering.

Step 3. Let the output initially set as $V^m(\omega(k)) = \{\ \}$.

Step 4. For all input symbols "0" and "1," we are to derive the initial blocks of maximal compatibles (MC's). Start with the input symbol "0," By merge table, we have the MC's for {1, 2, 4, 6, 7}:

$MC_1 = [1]$
$MC_2 = [2, 6, 7]$
$MC_3 = [4, 6]$



Step 5. From the MC's above, we find these two partitions for the states that are defined under the input symbol "0:"

$P_1 =$ [1] [2, 6, 7] [4] $P_2 =$ [1] [2, 7] [4, 6]

Step 6. For each of the partitions obtained above, we are to derive the associated next blocks for each of the blocks in the partition.

Step 7. Beginning with P1, we generate NB_k for each B_k.

$B_1 = [1] \rightarrow NB_1 = [2]$
$B_2 = [2, 6, 7] \rightarrow NB_2 = [3, 7, 8]$
$B_3 = [4] \rightarrow NB_3 = [5]$

Step 8. Compare each NB_k with all B_k, and classify it according to the condition (a, b or c) it meets.

$NB_1 = [2]$ → condition (a)
$NB_2 = [3, 7, 8]$ → condition (b) → $B_2' = [3, 8]$
$NB_3 = [5]$ → condition (c) → $B_{-1} = [5]$

Step 9. Repeat the next block transition in Step 7 to Step 8 on all B_k and B_{-l} until all original states are included in the blocks $B_k \cup B_k$ and B_{-l}.

$B_2' = [3, 8]$ →$NB_1 = [4]$ → condition (a)
$B_{-1} = [5]$ →$NB_2 = [6]$ → condition (a)

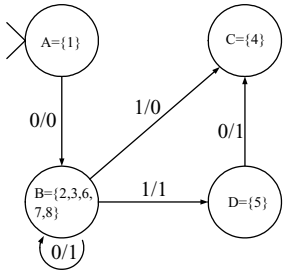
Step 10. Add all existing B_{-l} blocks to the blocks $B_k \cup B_k'$ in all possible combinations to obtain possible closed coverings.

Renumber the resultant blocks of and B_{-l}:

$B_1 = [1]$ →A
$B_2 + B_2' = [2, 6, 7]; [3,8]$ →B
$B_3 + B_3' = [4]$ →C
$B_{-1} = [5]$ →D

Below is the corresponding reduced machine that reduces the original 8-state machine into a four-state machine as shown in Fig. 4.

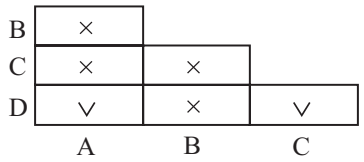
To further minimize this reduced machine, we find the associated compatible sets by the merge table in Table. 5.



	Input = 0	Input = 1
A	B, 0	–, –
B	B, 1	C, 0
C	D, 1	–, –
D	–, –	B, 1

Fig. 4. Reduced machine of P_1 under input symbol 0.

Table 5. Merge table for the reduced machine of P_1 under input symbol "0".



C: (CD)
B: (CD)(B)
A: (B)(CD)(AD)

Step 11. Because Lower_bound = 3 and that the upper bound, being equal to the number of maximal compatible sets, is 3 in this case, we know that the order of the minimal realization for the reduced machine must be 3. Permutation on the maximal compatibles leaves the following two possibilities:

1. AD = [1]; [5]
   B = [2, 3, 6, 7, 8]
   C = [4]
2. A = [1]
   B = [2, 3, 6, 7, 8]
   CD = [4]; [5]

Both accept the input sequence of $\omega(k)$, thus satisfying the closed covering property.

Step 12. Two minimal realization solutions corresponding to the two closed coverings are found, as shown in Fig. 5.

With isomorphism check, we have two minimal realizations for the reduced machine for P1 under input symbol "0" as shown in Fig. 6.

Now, update Current_Order to 3, which is the smallest number of states we have found in our minimal realization so far.

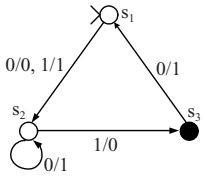
Go to Step 6.

Step 6. We now derive the associated next blocks for P_2.

Step 7. In P_2, we generate next block NB_k for each B_k.

$B_1 = [1] \rightarrow NB_1 = [2]$
$B_2 = [2, 7] \rightarrow NB_2 = [3, 8]$
$B_3 = [4, 6] \rightarrow NB_3 = [5, 7]$

Present State	Next State, Output	
	Input = 0	Input = 1
State 1 = [1];[5] State 2 = [2, 3, 6, 7, 8] State 3 = [4]	State 2, 0 State 2, 1 State 1, 1	State 2, 1 State 3, 0 –, –



Present State	Next State, Output	
	Input = 0	Input = 1
State 1 = [1] State 2 = [2, 3, 6, 7, 8] State 3 = [4];[5]	State 2, 0 State 2, 1 State 3, 1	–, – State 3, 0 State 2, 1

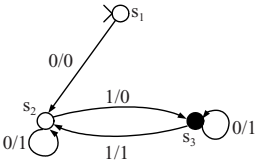


Fig. 5. Two minimal realizations for the reduced machine for P_1 under input symbol "0".

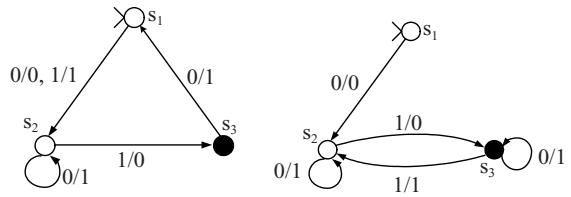


Fig. 6. Minimization for the reduced machine for P_1 under input symbol "0".

Step 8. Compare each NB_k with all B_k, and then classify it according to the condition (a, b or c) it meets.

$NB_1 = [2]$ → condition (a)
$NB_2 = [3, 8]$ → condition (c) → $B_{-1} = [3, 8]$
$NB_3 = [5, 7]$ → condition (b) → $B_2' = [5]$

Step 9. Repeat Step 7 to Step 8 on all B_k and B_{-l} until all original states are included in the blocks $B_k \cup B_k'$ and B_{-l}.

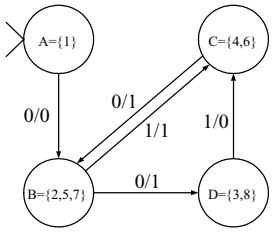
$B_{-1} = [3, 8]$ →$NB_1 = [4]$ → condition (a)
$B_2' = [5]$ →$NB_2 = [6]$ → condition (a)

Step 10. Add all existing B_{-l} blocks to the blocks $B_k \cup B_k'$ in all possible combinations to obtain possible closed coverings.

Renumber the resultant blocks of $B_k \cup B_k'$ and B_{-l}.

$B_1 + B_1' = [1]$ →A
$B_2 + B_2' = [2, 7] + [5]$ →B
$B_3 + B_3' = [4, 6]$ →C
$B_{-1} = [3, 8]$ →D

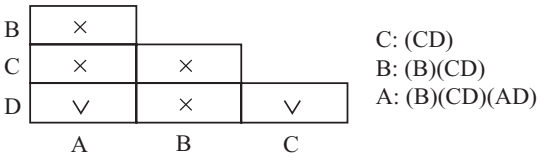
Figure 7 shows the corresponding reduced machine that reduced the original 8-state machine into a four-state machine. To further minimize this four-state machine, we are to find the associated maximal compatible sets from Table 6.



	Input = 0	Input = 1
A	B, 0	–, –
B	D, 1	C, 1
C	B, 1	–, –
D	–, –	C, 0

Fig. 7. Reduced machine for P2 under input symbol "0".

Table 6. Merge table for the reduced machine of P2 under input symbol "0".



Step 11. With Current_Order = 3, the number of states in minimal realization cannot be larger that 3. From the MC's = (B)(CD)(AD), permutation on the maximal compatible sets leaves us two choices: {(AD), (B), (C)} and {(A), (B), (CD)}: both satisfy the closed covering condition by checking against $\omega(k)$. The corresponding closed covering solutions are listed as following:

1. AD = [1];[3, 8]
   B = [2, 5, 7]
   C = [4, 6]
2. A = [1]
   B = [2, 5, 7]
   CD = [4, 6]; [3, 8]

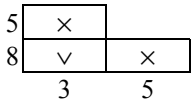
Step 12. Two minimal realization solutions corresponding to the two closed coverings are found, as shown in Fig. 8.

With isomorphism check, we have two minimal realizations for the reduced machine for P_2 under input symbol "0" as shown in Fig. 8.(*)

Now, the updated Current_Order is still 3, as we cannot find a minimization with less number of states yet.

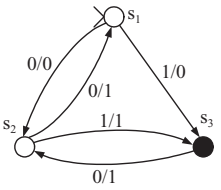
Step 4. We then do for input symbol "1," for the set of states defined under input symbol "1" {3, 5, 8}. By merge table, we find the set of MC's:

$MC_1 = [3, 8]$
$MC_2 = [5]$



(*) Overall, there are four minimal realizations under input symbol "0"

Present State	Next State, Output	
	Input = 0	Input = 1
State 1 = [1]; [3, 8]	State 2, 0	State 3, 0
State 2 = [2, 7]; [5]	State 1, 1	State 3, 1
State 3 = [4, 6]	State 2, 1	–, –



Present State	Next State, Output	
	Input = 0	Input = 1
State 1 = [1]	State 2, 0	–, –
State 2 = [2, 7]; [5]	State 3, 1	State 3, 1
State 3 = [4, 6]; [3, 8]	State 2, 1	State 3, 0

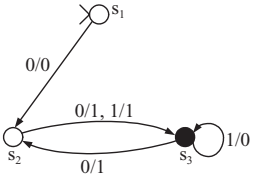


Fig. 8. Two minimal realizations for the reduced machine for P_2 under input symbol "0".

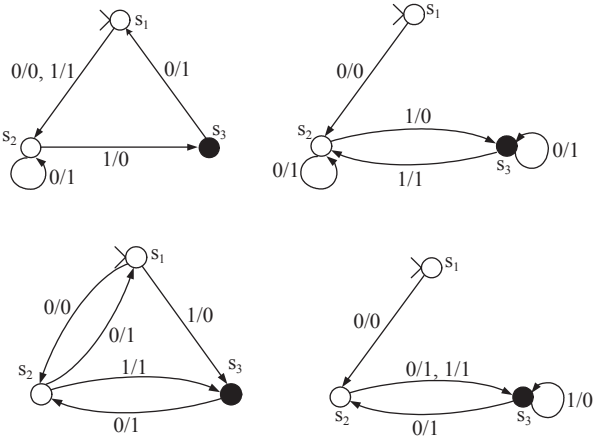


Fig. 9. All the minimization solutions under input symbol "0".

Step 5. There is only one partition derived from the MC's.

P1 = [3, 8]
[5]

Step 6. We begin with P_1.

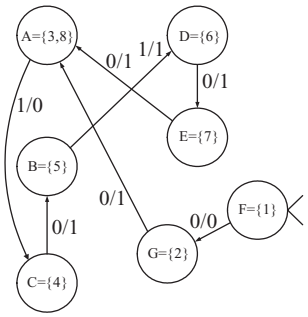
Step 7. Beginning with P1, we generate NB_k for each B^k.

$B_1 = [3, 8] \rightarrow \quad NB_1 = [4]$
$B_2 = [5] \quad \rightarrow \quad NB_2 = [6]$

Step 8. Compare each NB_k with all B_k, and then classify it according to the condition (a, b or c) it meets.

$NB_1 = [4] \quad \rightarrow$ condition (c) $\rightarrow \quad B_{-1} = [4]$
$NB_2 = [6] \quad \rightarrow$ condition (c) $\rightarrow \quad B_{-2} = [6]$

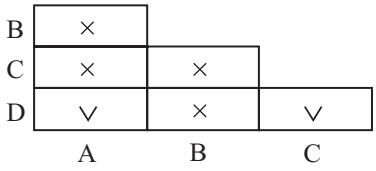
Step 9. Repeat Step to Step on all B_k and B_{-l} until all original states are included in the blocks $B_k \cup B_k'$ and B_{-l}.



	Input = 0	Input = 1
A	–, –	C, 0
B	–, –	D, 1
C	B, 1	–, –
D	E, 1	–, –
E	A, 1	–, –
F	G, 0	–, –
G	A,1	–, –

Fig. 10. Reduced machine for P1 under input symbol "1".

Table 7. Merge table for the reduced machine of P_1 under input symbol "1"



F: (F)(G)
E: (F)(EG)
D: (F)(DEG)
C: (F)(DEG)(CD)
B: (BF)BDEG)(BCD)
A: (BF)(BDEG)(BCD)(AF)(ADEG)(ACD)

$B_{-1} = [4] \rightarrow NB_1 = [5] \rightarrow$ condition (a)
$B_{-2} = [6] \rightarrow NB_2 = [7] \rightarrow$ condition (c) $\rightarrow$
$B_{-3} = [7]$
$B_{-3} = [7] \rightarrow NB_1 = [8] \rightarrow$ condition (a)

Step 10. Add all existing B_{-l} blocks to the blocks $B_k \cup B_k'$ in all possible combinations to obtain possible closed coverings.

Renumber the resultant blocks of $B_k \cup B_k'$ and B_{-l}.

$B_1 = [3, 8] \rightarrow A$
$B_2 = [5] \rightarrow B$
$B_{-1} = [4] \rightarrow C$
$B_{-2} = [6] \rightarrow D$
$B_{-3} = [7] \rightarrow E$
$B_{-4} = [1] \rightarrow F$
$B_{-5} = [2] \rightarrow G$

The corresponding reduced machine with seven-state machine is shown in Fig. 10.

To further minimize this seven-state machine, we are to find the associated maximal compatible sets from Table 7 above.

Step 11. From the result computed for the input symbol "0," we know that Current_Order = 3. The MC's we have above include: (BF)(BDEG)(BCD)(AF)(ADEG)(ACD). The order of minimization candidates must be no larger than 3. Permutation of the MC's, with the constraint

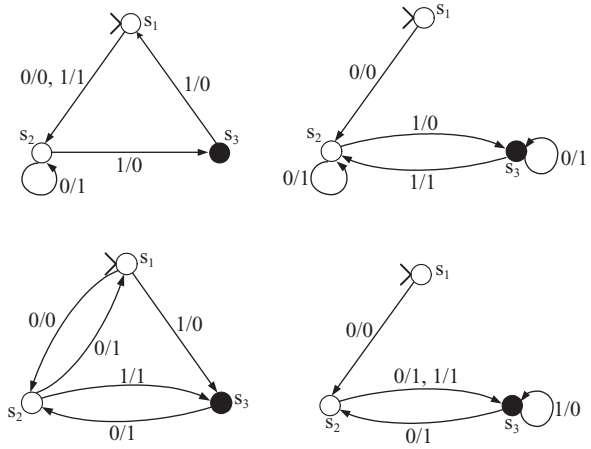


Fig. 11. Minimal Valid Automata $V^m(\omega(k))$.

on the order no larger than 3 and with the constraint on closed covering by $\omega(k)$, yields four minimal realization solutions: {(BF), (ADEG), (C)}; {(F), (ADEG), (BC)}; and {(AF), (BEG), (CD)}; {(F); (BEG), (ACD)}.

However, minimization solution corresponding to the first two are exactly the two in Fig. 6, and the next two are those in Fig. 8. For this case, the minimal realization derived from the input symbol "1' is the same as that derived for the input symbol of "0."

Step 12. Finally, we obtain the minimal valid automata $V^m(\omega(k))$ as the set of the four minimal nealizations in Fig. 11.

V. CONCLUSION

Given an input-output sequence of an unknown DEDS, this paper presents an algorithm of minimal valid automata that yields minimal realization of reconstructed target. Critical to system identification problems of discrete event systems whose behavior are best modeled by Meanly machines, MVA can be considered as a special case of the minimization problem for incompletely specified sequential machine (ISSM). This paper presents an algorithm that divides the minimization process into two phases: one to give a reduced machine for the equivalent sequential machine, and then another to minimize the reduced machine into minimal realization solutions.

REFERENCE

1. Lee, D. and M. Yannakakis, "Principles and Methods of Testing Finite State Machines—a Survey," *Proc. IEEE*, Vol. 84, No. 8, pp. 1090-1123 (1996).
2. Chung, S.-L., J.-C. Wu, and C.-L Li, "System Identification of Discrete Event Systems," *J. Chinese Inst. Eng.*, Vol. 27, No. 2, pp. 203-210 (2004).
3. Chung, S.-L., J.-C. Wu, W.-Y. Chen, and C.-L Li, "Online Modeling Refinement for Discrete Event Systems," Accepted and to Appear in *J. Chinese Inst. Eng.* Vol. 28 (2005).
4. Gold, E. M., "Complexity of Automaton Identification from Given Data," *Inform. Contr.*, Vol. 37, pp. 302-320 (1978).
5. Hopcroft, J., "An n log n Algorithm for Minimizing States in a Finite Automaton" In: *Theory of Machines and Computations*, Academic Press, New York, pp. 189-196 (1971).
6. Pleeger, C. P., "State Reduction in Incompletely Specified Finite State Machines," *IEEE Trans. Comput.*, Vol. 22, pp. 1099-1102 (1973).
7. Kohavi, Z., "Switching and Finite Automata Theory," 2nd Ed., McGraw-Hill, New York(1978).
8. Kella, J., "Sequential Machine Identification," *IEEE Trans. Comput.*, Vol. C-20, No. 3, pp. 332-338 (1971).
9. Rao, C. V. S. and Nripendra N. Biswas, "Minimization of Incompletely Specified Sequential Machines," *IEEE Trans. Comput.*, Vol. c-24, No. 11, pp. 1089-1100 (1975).
10. Masaru Yamamoto, "A Method for Minimizing Incompletely Specified Sequential Machines," *IEEE Trans. Comput.*, Vol. c-29, No. 8, pp. 732-736 (1980).
11. Avedillo, M. J., J. M. Quintana, and J. L. Huertas, "A New Method for the State Reduction of Incompletely Specified Finite Sequential Machines," in *Proc. Workshop Logic Arch. Synthesis*, pp. 107-115 (1990).
12. Puri, R. and J. Gu, "An Efficient Algorithm to Search for Minimal Closed Covers in Sequential Machines," *IEEE Trans. Comput. Aid. Design Integr. Circuits Syst.*, Vol. 12, No. 6, pp. 37-745 (1993).

Sheng-Luen Chung received his B.S. degree in electronic engineering department from the National Chiao-Tung University, Taiwan in 1985, and the M.S.E. and Ph.D. degrees from the Department of Electrical Engineering and Computer Science, University of Michigan, Ann Arbor, in 1990 and 1992, respectively. Since 1992, he has been with the Electrical Engineering Department of the National Taiwan University of Science and Technology, Taiwan, where he is now an Associate Professor. As a pioneer in Taiwan semiconductor manufacturing automation industry, he has been an active consultant

working with MITRI, Consilium, Applied Material, IBM, and HP in several advanced semiconductor fabs and TFT/LCD fabs in Taiwan. His current research interests include system identification, embedded system design and applications of supervisory control. He is granted Research Award by National Science Council from 1994 to 2001. He was a recipient of IEEE 1994 George E. Axelby Outstanding Paper Award from the Control System Society of the IEEE in 1994 for a paper co-authored with Stephane Lafortune and Feng Lin.

Chung-Lun Li received his B.S. degree in automatic control engineering from the Feng Chia University, Taiwan, R.O.C. in 2001, and the M.S. degree in electrical engineering from the National Taiwan University of Science and Technology, Taiwan, R.O.C. in 2003. He is currently a Ph.D. student at the department of electrical engineering of National Taiwan University of Science and Technology. His research interests include: embedded system design, discrete event dynamic systems, and applications of supervisory control.