An Efficient Algorithm to Search for Minimal Closed Covers in Sequential Machines

Ruchir Puri, *Student Member, IEEE,* and Jun Gu, *Senior Member, IEEE*

***Abstract*—The problem of state reduction in a finite state machine (FSM) is important to reduce the complexity of a sequential circuit. In this paper, we present an efficient algorithm for state minimization in incompletely specified state machines. This algorithm employs a tight lower bound and a fail-first heuristic, and generates a relatively small search space from the prime compatibles. It utilizes efficient pruning rules to further reduce the search space and finds a minimal closed cover. The technique guarantees the elimination of all the redundant states in a very short execution time. Experimental results with a large number of FSM's including the MCNC FSM benchmarks, are presented. The results are compared with other recent work in the area.**

I. Introduction

THE REMOVAL of redundant states is important to reduce the circuit complexity in the design of sequential circuits. In a completely specified finite state machine (FSM) with n states, a solution can be achieved successfully in $O(n \log n)$ time [22]. In such FSM's, the merging of equivalent states yields a unique minimal solution [27]. A state machine where transitions under some inputs lead to unspecified states or unspecified outputs is called an incompletely specified finite state machine (ISSM) [16], [23]. The state minimization problem for such a machine is, unfortunately, NP-complete [11], [30]. A minimal solution obtained in such a case may not be unique [13]. This problem has been studied for many years [12], [13], [27], [31]. In recent years, because of the development of automated FSM synthesis systems [1], [9], [10], [28], there has been a renewed interest in this problem [2], [17], [20], [29], [32].

In this paper, we present a new algorithm for the state minimization problem. This algorithm constructs a search tree from prime compatibles. It efficiently builds up a relatively small search space by utilizing a tight lower bound derived from maximal incompatibles. Efficient pruning criteria are developed to further prune the search space. The algorithm is capable of removing all the redundant states in a given FSM and guarantees a minimal FSM solution. This algorithm can generate all the minimal FSM solutions with almost no time overhead. It then selects the FSM having minimum implementation area. Our experiments with practical FSM's, including MCNC FSM benchmarks, show that the technique is effective in reducing time and memory requirements for large size FSM's. These results are compared with recent experimental results by Kannan and Sarma [20] and by Hachtel *et al.* [17].

The rest of this paper is organized as follows. In Section II, we briefly overview the previous work in the area. Section III gives some basic definitions and notations that simplify our discussion. In Section IV, we describe in detail an efficient algorithm for state reduction. Experimental results with industrial FSM's, including MCNC benchmarks, are illustrated in Section V. Section VI concludes this paper.

Manuscript received June 1, 1991; revised June 26, 1992. This work was supported in part by the NSERC Operating Grant OGP0046423 and NSERC Strategic Grant MEF0045793. This paper was recommended by Associate Editor R. K. Brayton.

The authors are with the Department of Electrical and Computer Engineering, University of Calgary, Alta., Canada T2N 1N4.

IEEE Log Number 9206846.

II. Previous Work

Paul and Unger [27] and Unger [33] developed a general theory of ISSM's and presented a systematic approach for generating maximum compatibles and minimal closed covers. Since then many researchers have studied the reduction of this enumerative tabular procedure proposed by Paul and Unger. Grasselli and Luccio [13] solved the binate covering (i.e., state minimization) problem by an integer linear programming approach. Luccio [24] further extended the definition of prime compatibility classes to simply the procedure. The chain generating method developed by Meisel [25] generates all the paths and chains unconditionally. This leads to an unacceptably large search space in case a machine has a large number of prime compatibles. DeSarkar *et al.* [8] derived all the irredundant prime closed sets and chose the minimal one that covers the machine. This method becomes inefficient when a large number of irredundant prime closed sets exists.

Kella [21] proposed a method that avoids the generation of a complete set of maximal compatibles by adding new states recursively. This method generates all possible reduced machines so that no machine with fewer states is overlooked. House and Stevens [19] and Curtis [7] developed reduction rules to reduce the size of a covering-closure table proposed by Grasselli and Luccio [13]. Bennetts [3] developed a method of deriving prime compatibles and corresponding implications without deriving the maximal compatibles. Pager [26] proposed a method that dealt with only a very special class of FSM's (i.e., where a minimal closed cover can be derived from maximal compatibles only). Yang's method [35] eliminates all the implication unrelated and superseded compatibles, which substantially reduces the number of compatibles under

0278-0070/93$03.00 © 1993 IEEE

consideration. His method does not guarantee that at least one minimal closed cover can be derived from the remaining set of compatibles [31]. Biswas [4] proposed a technique based on implication trees. The approach is suitable for small FSM's, where the minimal closed cover contains at least one maximal compatible. Biswas later modified this method to account for machines where none of the solutions contain a maximal compatible [5]. Rao and Biswas [31] applied some deletion rules to the set of compatible classes and obtained a relatively small set of symbolic compatibles. A minimal solution was then obtained from these symbolic compatibles.

Perkowski and Nguyen [29] gave a backtracking algorithm that checks partially generated solutions using dynamic rules. This algorithm employs an optimum variant to find the optimality of the solutions. For large size FSM's, the method requires excessive computing time. Avedillo *et al.* [2] derived a reduced FSM by applying a sequence of transformations to internal states in a given FSM. Kannan and Sarma [20] proposed a simple heuristic algorithm to find a minimal closed cover. Their approach yields a suboptimal solution in some cases and suffers from excessive computing time in the case of large FSM's. Recently, Hachtel *et al.* [17] described the exact and heuristic algorithms for minimizing incompletely specified finite state machines which yield optimal results in most of the cases.

Following Meisel's approach [25] in generating a search tree, our algorithm employs a tight lower bound and an ordered generating sequence to create a smaller search space. Efficient pruning criteria are developed to reduce further the search space. Compared to the Perkowski–Nguyen backtracking algorithm [29], which employs an optimum variant to prove the optimality, in our algorithm, the first path on the search tree satisfying the covering and closure constraints is guaranteed to be a minimal FSM solution.

III. Preliminaries

The external behavior of an FSM can be described by a state transition graph (STG). An STG can be represented by a flow table or a cube table. The flow table representation is a two-dimensional array where columns correspond to the input states and rows correspond to the internal states. The entries are ordered pairs representing the next internal state and the output. In a cube table representation (e.g., MCNC FSM benchmarks), each row corresponds to a transition edge of the STG. Thus each entry specifies an input, the present state, the next state, and the output. Each nontrivial entry of a flow table can be mapped as a corresponding entry in the cube table. In the following discussion, we will use a simple state machine example to illustrate our algorithm. The flow table of the FSM example *ungerex* is shown in Table I [33].

For a finite state machine $\mathfrak{M}$ with n states, let C_i denote the ith compatible class and let $\mathcal{S} = \{C_1, C_2 \cdots\}$ denote a set of compatible classes. Following basic notations in [23], [24], [33], we now give some basic definitions.

TABLE I
FLOW TABLE OF FSM EXAMPLE *ungerex*

Present State	I_0	I_1	I_2	I_2
a	b/0	d/0	-/0	g/-
b	c/0	e/0	-/0	-/-
c	a/0	f/0	-/0	-/-
d	-/-	b/1	-/1	a/-
e	-/-	-/-	-/1	c/0
f	b/1	-/-	-/1	-/-
g	-/0	-/1	h/-	-/-
h	-/1	-/0	i/-	-/-
i	-/1	-/1	-/-	a/1

Definition 1: States p and q are *compatible*, if and only if, for every possible input sequence applicable to p and q, the same output sequence is produced, regardless of whether p or q is an initial state. If the output sequences differ, then p and q are called *incompatible*.

Definition 2: A set of states Q is *compatible*, if and only if, for every possible input sequence, no two conflicting output sequences will be produced, regardless of which state is the initial state.

Definition 3: 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.

Definition 4: 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*, or a *maximal incompatible* (MI), if it is not covered by any other incompatible class.

In the FSM example *ungerex*, there are five maximal compatibles, i.e., {hfe, ifd, ged, fed, cba}, and twelve maximal incompatibles, i.e., {ihgc, gfc, iec, hdc, ihgb, gfb, ieb, hdb, ihga, gfa, iea, hda}.

Definition 5: Under input i, a set of states Q *implies* another set of states R, if R is a set of next states for Q. If Q is a compatible, then R is called an *implied compatible* of Q under input i.

Definition 6: A *closure class* of compatible C_i, denoted as $\Phi(C_i)$, is a set of the implied compatibles of C_i such that

1) Each implied compatible has more than one state
2) Each implied compatible is not contained in C_i
3) Each implied compatible is not contained in any other member of the closure class

A closure class is obtained by the repeated application of implication transitivity for all inputs.

Definition 7: A compatible class C_i is said to be *prime* if there exists no other compatible class C_j such that

1) C_j covers C_i
2) Every member of closure class $\Phi(C_j)$ is contained in at least one member of closure class $\Phi(C_i)$

For the FSM example *ungerex*, the prime compatibles and their corresponding closure classes are shown in Table II.

Definition 8: A compatible C_i in a set of compatibles $\mathcal{S}$ is said to be *unimplied* if neither C_i nor any of its subsets are implied by any member of $\mathcal{S}$.

Definition 9: An *extended closure class* of $\mathcal{S}$, denoted as $\Phi(\mathcal{S})$, is a set of the implied compatibles of $\mathcal{S}$ such that

1) Each implied compatible has more than one state

TABLE II
PRIME COMPATIBLES AND CORRESPONDING CLOSURE CLASSES

C_i	$\Phi(C_i)$
hfe	∅
ifd	∅
ged	ba, df, bc, ef, ac
fed	ba, bc, ac
cba	fed
gd	∅
ge	∅
ba	df, bc, de, ca, ef
ca	bc, de, ef, ba, fd
cb	ba, df, de, ac, fe
a	∅
b	∅
c	∅

2) Each implied compatible is not contained in any compatible of $\mathcal{S}$
3) Each implied compatible is not contained in any other member of the extended closure class

An extended closure class is obtained by the repeated application of implication transitivity for all inputs.

Definition 10: A set of compatibles $\mathcal{S}_j$ *dominates* ($\leq$) another set of compatibles $\mathcal{S}_i$ if every compatible of $\mathcal{S}_i$ is contained in at least one compatible of $\mathcal{S}_j$.

In the FSM example, set $\mathcal{S}_1$ = {ifd, hfe, ged, a} and set $\mathcal{S}_2$ = {ifd, hfe, ged, cba} have the extended closure class $\Phi(\mathcal{S}_1) = \varnothing$ and $\Phi(\mathcal{S}_2)$ = {fed}, respectively.

The set of compatibles $\mathcal{S}_2$ dominates another set of compatibles $\mathcal{S}_1$, i.e., $\mathcal{S}_1 \leq \mathcal{S}_2$, since every member of $\mathcal{S}_1$ is contained in at least one member of $\mathcal{S}_2$.

Definition 11: A set of compatibles *covers* machine $\mathfrak{M}$ if it contains all the states of the machine.

The *weight* of $\mathcal{S}$, denoted as $w(\mathcal{S})$, is the number of distinct internal states covered by compatibles in $\mathcal{S}$. If $\mathcal{S}$ covers the machine $\mathfrak{M}$, then $w(\mathcal{S})$ is equal to n.

Definition 12: A set of compatibles is *closed* if for every compatible contained in the set, all its implied compatibles are also contained in the same set. The extended closure class of a closed set of compatibles $\mathcal{S}$ is empty, i.e., $\Phi(\mathcal{S}) = \varnothing$.

For example, the set of compatibles $\mathcal{S}$ = {ifd, hfe, ged, cba} covers FSM example *ungerex*, since every state of *ungerex* is covered by at least one of the compatibles in $\mathcal{S}$. Thus, the weight of set $\mathcal{S}$, i.e., $w(\mathcal{S}) = 9$.

The extended closure class of the set $\mathcal{S}$ = {ifd, hfe, ged, cba, fed} is empty, i.e., $\Phi(\mathcal{S}) = \varnothing$. Thus, the set of compatibles $\mathcal{S}$ is closed.

Definition 13: A set of k compatibles $\mathcal{S}$ is called a *minimal closed cover* if and only if $\mathcal{S}$ satisfies

1) *Covering condition:* $\mathcal{S}$ covers the machine $\mathfrak{M}$, i.e., $w(\mathcal{S}) = n$.
2) *Closure condition:* $\mathcal{S}$ is closed, i.e., $\Phi(\mathcal{S}) = \varnothing$.
3) *Minimal condition:* A set of $k - 1$ or less compatibles does not satisfy both covering condition and closure condition.

A minimal closed cover is a minimal FSM solution, i.e., a reduced FSM.

In FSM example *ungerex*, $\mathcal{S}$ = {ifd, hfe, ged, cba, fed} is a minimal closed cover, since it satisfies the covering condition, the closure condition, and the minimal condition.

The goal of *state minimization* is to find an equivalent FSM, i.e., a minimal closed cover, that simulates the given external behavior with a reduced number of states. The state minimization process normally consists of the following steps:

- Detection of redundant states which can be merged, producing pairwise compatibles.
- Derivation of prime compatibles from pairwise compatibles (in the case of completely specified FSM's, the solution can be found only from maximal compatibles).
- Selection of a set of compatibles from prime compatibles which satisfies the covering constraint, the closure constraint, and the minimal constraint.
- Ensuring that the functional behavior of the reduced FSM is equivalent to the original one.

IV. Efficient Algorithm for Finding a Minimal FSM

The algorithm starts by generating all pairwise compatibles and incompatibles [23] from FSM cube table representation (e.g., the KISS format [36]). We employ Bennetts's approach [3] to derive compatible classes directly from the list of pairwise compatibles. Some of the compatible classes are then removed by utilizing Luccio's deletion rules [24]. The remaining compatible classes are called *prime compatibles*, which are used to generate the search tree.

In the following discussion, we will describe the lower bound on the number of states in a reduced FSM, the generating sequence for the construction of the search tree, search tree generation and pruning, and the algorithm to search for a minimal FSM solution.

4.1. Lower Bound

A set of k incompatible states $q_1, q_2 \cdots, q_k$ must be covered by k compatible classes, since a single compatible class cannot cover two incompatible states. In addition, the compatibles in a minimal closed cover must cover all the internal states of the given FSM. Hence, a maximal incompatible containing a maximum number of states determines the limit to further state reduction. The *lower bound*[1] on the size of the reduced FSM, denoted as Ω, is equal to the number of states contained in a maximal incompatible with the maximum number of states. The maximum incompatibles (i.e., maximal cliques) are derived from the incompatibility graph using Carraghan's maximal clique algorithm [6].

Example: In FSM example *ungerex*, there are three maximal incompatibles having a maximum number of states, i.e., *ihgc*, *ihgb*, and *ihga*. This implies that the maximum number of states in a maximal incompatible is four. Thus, the lower bound Ω on the size of the minimal closed cover is four.

[1]This lower bound does not account for the closure constraint; thus, in some cases, a closed covering of this minimal size may not be possible.

4.2. Generating Sequence

4.2.1. Selecting a Maximal Incompatible as a Generating Sequence

A generating sequence is an ordered sequence of states in a maximal incompatible that contains the maximum number of states and generates a minimum number of nodes in the search tree. The generating sequence, denoted as $s_1, s_2, s_3 \cdots s_\Omega$, is used to determine the priority of the levels of the search tree. Each level l corresponds to a state s_l in the generating sequence. The nodes at level l of the search tree are determined by the prime compatibles that cover the state s_l in the generating sequence.

The weight of a state q_j, denoted as $w(q_j)$, in a maximal incompatible is equal to the number of prime compatibles covering state q_j. The total number of nodes, which is also referred to as the weight of maximal incompatible, denoted as $w(\mathrm{MI})$, equals $\Pi_{j=1}^{|\mathrm{MI}|} w(q_j)$.

It is possible that the number of states in more than one maximal incompatible equals the lower bound Ω. In such a case, the maximal incompatible having the least weight $w(\mathrm{MI})$ (generating the minimum number of nodes) is chosen as the generating sequence. If the weights of two or more maximal incompatibles are the same, then the ties are broken arbitrarily and any one of them is chosen as the generating sequence.

Example: For FSM example *ungerex*, the maximal incompatibles having maximum number of states, and their corresponding total number of nodes are shown in Table III. Three maximal incompatibles, i.e., *ihgc*, *ihgb*, and *ihga*, can be chosen as the generating sequence. Each will generate an equal number of nodes, i.e., twelve. In such a case, an arbitrary choice is made, and the maximal in compatible *ihga* is chosen as the generating sequence.

4.2.2. Ordering States in the Generating Sequence

Pruning is more effective at a high level of a search tree. Thus, the states of the generating sequence are ordered ascendingly in terms of their weights, so that a node with fewer children nodes is selected first to construct the search tree (fail-first heuristic [14], [15], [18]). If two states in the ordered generating sequence have equal weights and if the second state has a higher chance of pruning the nodes, then we interchange their order. In most instances, the swapping of the order results in a substantial reduction in the number of nodes in the search tree.

Example: For FSM example *ungerex*, the weight of states in the generating sequence *ihga* are

$$w(i) = w(h) = 1, \quad w(g) = 3, \quad w(a) = 4.$$

State i being least in weight is ordered at first position, followed by state h, state g, and state a in that order. States i and h have equal weights, i.e., one. But none of them can prune the nodes, since each generates only one node, i.e., *ifd* and *hfe*, respectively. Thus, the order of state i and state h remains unchanged. The ordered generating sequence for FSM example *ungerex* is *ihga*.

The obtained sequence of state variables in the maximal incompatible is called an *ordered generating sequence* (OGS), denoted as $s_1 s_2 s_3 \cdots s_\Omega$. The algorithm *generate_sequence*, shown in Fig. 1, takes the set of maximal incompatibles and the set of prime compatibles as inputs and produces an OGS.

TABLE III
MAXIMAL INCOMPATIBLES HAVING MAXIMUM NUMBER OF STATES

Maximal Incompatible (MI)	Weight of Each State $w(q_j)$	Weight $w(MI)$
ihgc	$w(i)=1, w(h)=1, w(g)=3, w(c)=4$	12
ihgb	$w(i)=1, w(h)=1, w(g)=3, w(b)=4$	12
ihga	$w(i)=1, w(h)=1, w(g)=3, w(a)=4$	12

4.3. Search Tree Generation

In this section we describe the search tree generation process. The search tree is constructed to find a minimal solution. The OGS and the lower bound on the size of a reduced FSM are used to generate the search tree. In the rest of the paper, let

- *A node* on the search tree represent a compatible. The kth node on level l is denoted by $n_{l,k}$.
- *A path* on the search tree represent a set of compatibles. A path consisting of l nodes (from the root node to the kth node on level l) is denoted by $path_{l,k}$.

Nodes in the search tree are comprised of prime compatibles. The search tree is generated level by level in terms of the OGS ($s_1 s_2 s_3 \cdots s_\Omega$), where $w(s_1) \le w(s_2) \le \cdots \le w(s_\Omega)$. There are Ω states in the generating sequence, and each state s_l is associated with the corresponding level l of the search tree. A search tree generated by such a method has fewer children nodes attached to a node at the higher levels than at the lower ones, facilitating efficient pruning. The tree generation process starts at root level ($l = 0$). Each node at the current level l is expanded to the next lower level ($l + 1$) by choosing prime compatibles covering the state s_{l+1} of the generating sequence as its children. The nodes are expanded level by level until all Ω state variables in the generating sequence are used. Thus, each node at level Ω spans a path from the root node consisting of exactly Ω compatibles.

Example: For FSM example *ungerex*, the generating sequence derived in the previous section was *ihga*. Thus, states i, h, g, and a of the generating sequence are associated with the first, second, third, and fourth levels of the search tree, respectively. The prime compatibles containing states i, h, g, and a are shown in Table IV.

We start with an empty node at root level ($l = 0$). There is only one node at the first and second levels of the search tree, namely *ifd* and *hfe*. There are three nodes, i.e., *ged*, *gd*, and *ge*, at the third level. Each of these nodes will have four children nodes, i.e., *cba*, *ba*, *ca*, and *a*, at the fourth level. Thus, in total there are twelve nodes at the fourth level of the search tree. The tree generation process stops as the four states in the generating sequence, i.e., i, h, g and a, have been used. The search tree generation process for this machine *ungerex* is illustrated in Fig. 2.

```
Input  : A set of maximal incompatibles.
       : A set of prime compatibles.
Output : An ordered generating sequence (OGS).

Procedure generate_sequence( ) {
  find the lower bound Ω on the size of the reduced FSM ;
  for each maximal incompatible MI with number of states equal to Ω {
     compute the weight of each state w(q_j) in MI ;
     compute the weight of MI as w(MI)=∏_{j=1}^{Ω} w(q_j) ;
  }
  find a maximal incompatible (MI) having Ω states and minimum weight w(MI);
  sort the states of this maximal incompatible in the ascending order of their weights ;
  if two states q_i and q_j have equal weights {
     /* compare the chances of q_i and q_j to prune the nodes */
     find the sets of prime compatibles P_i and P_j containing states q_i and q_j ;
     assign weights to P_i and P_j, equal to the number of prime compatibles,
           that are contained in the rest of their members ;
     if the first weight is larger
        then interchange the order of state q_i and state q_j ;
  }
  return the states of ordered maximal incompatible MI
     as the states of OGS (s_1 s_2 s_3 ... s_Ω) ;
}
```

Fig. 1. Algorithm to derive an ordered generating sequence.

TABLE IV
PRIME COMPATIBLES CONTAINING THE STATES IN THE GENERATING SEQUENCE

s_i : i_{th} State in OGS	Prime Compatibles Containing State s_i
i	ifd
h	hfe
g	ged, gd, ge
a	cba, ba, ca, a

In the following discussion we present the rules for search tree pruning.

4.4. Pruning Criteria

The search space generated by the tree generation process described in the previous section can be relatively large in case of a large FSM. In this section, we describe the pruning criteria used to reduce the search space. Some redundant nodes in the search tree can be deleted by comparing subsets of compatibles. The pruning rules ensure that the covering condition and closure condition of a deleted node are covered by some other nodes.

Let $\mathcal{S}_r = \{C_1, C_2, \cdots, C_{r-1}, C_r\}$ be a set of compatibles, which is shown in Fig. 3 as $path_{l,r}$. Assume that the end node C_r of $path_{l,r}$ has two children nodes, C_i and C_j. The compatible sets $\mathcal{S}_i$ ($\{C_1, C_2, \cdots, C_{r-1}, C_r, C_i\}$) and $\mathcal{S}_j$ ($\{C_1, C_2, \cdots, C_{r-1}, C_r, C_j\}$) form $path_{l+1,i}$ and $path_{l+1,j}$, respectively. Node C_i *deletes* C_j with respect to the compatible set $\mathcal{S}_r$ if all the conditions in either of the following two rules are satisfied.

Rule 1

Condition 1.1: C_i covers all the states contained in C_j, i.e., $C_j \subset C_i$.

Condition 1.2: The extended closure class of $\mathcal{S}_j$ dominates the extended closure class of $\mathcal{S}_i$. That is,

$$\Phi(\mathcal{S}_i) \leq \Phi(\mathcal{S}_j)$$

$$\Phi(\mathcal{S}_r \cup C_i) \leq \Phi(\mathcal{S}_r \cup C_j)$$

so,

$$\Phi(C_i) \leq \Phi(\mathcal{S}_r \cup C_j) \cup \mathcal{S}_r.$$

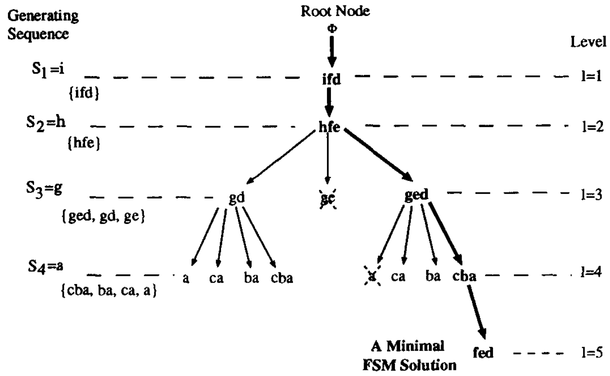


Fig. 2. Search tree for FSM example *ungerex*.

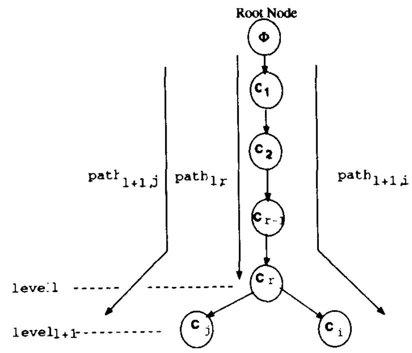


Fig. 3. Pruning a node in the search tree.

Proof: Let $\mathcal{S}$ be a set of compatibles containing all the compatibles in $\mathcal{S}_j$, i.e., $\{\mathcal{S}_r\} \cup C_j \subseteq \mathcal{S}$. Assume that $\mathcal{S}$ is a minimal closed cover, i.e., weight $w(\mathcal{S}) = n$ and the extended closure class $\Phi(\mathcal{S}) = \emptyset$ (Definition 13). If C_j in $\mathcal{S}$ is replaced[2] by a compatible class C_i such that $C_j \subset C_i$ (Condition 1.1), then the covering condition of $\mathcal{S}$ will not be affected, i.e., $w(\{\mathcal{S}\backslash C_j\} \cup C_i) = n$. If each number of the extended closure class of $\{\mathcal{S}_r\} \cup C_i$, is contained in at least one member of the extended closure class of $\{\mathcal{S}_r\} \cup C_j$, i.e., $\Phi(\mathcal{S}_i) \leq \Phi(\mathcal{S}_j)$ (Condition 1.2), then replacing C_j by C_i will not affect the closure condition of $\mathcal{S}$, i.e., $\Phi(\{\mathcal{S}\backslash C_j\} \cup C_i)$ remains empty. The number of compatibles in $\mathcal{S}$ remains unchanged if C_j is replaced by C_i, thus the minimal condition will not be affected. In summary, compatible C_j can be deleted without affecting the chances of finding the minimal closed cover, if both Conditions 1.1 and 1.2 are satisfied. □

Rule 2

Condition 2.1: If C_i does not cover all the states in C_j then these uncovered states should be covered by the set of compatibles $\mathcal{S}_r$ (i.e., $path_{l,r}$).

Condition 2.2: C_j is an unimplied compatible.

Condition 2.3: The extended closure class of $\mathcal{S}_j$ domi-

[2] $\mathcal{S}\backslash C_j$ is the set of compatibles obtained by removing compatible C_j from compatibles in $\mathcal{S}$.

nates the extended closure class set of $\mathcal{S}_i$. That is,

$$\Phi(\mathcal{S}_i) \leq \Phi(\mathcal{S}_j).$$

Proof: Let $\mathcal{S}$ be the set of compatibles containing all the compatibles in $\mathcal{S}_j$, i.e., $\{\mathcal{S}_r\} \cup C_j \subseteq \mathcal{S}$. Assume that $\mathcal{S}$ is a minimal closed cover, i.e., $w(\mathcal{S}) = n$ and $\Phi(\mathcal{S}) = \varnothing$ (Definition 13). If C_j in $\mathcal{S}$ is replaced by a compatible class C_i such that all the states of C_j not covered by C_i are covered by $\mathcal{S}_r$ (Condition 2.1), then $\mathcal{S}_i$ covers the same states as $\mathcal{S}_j$. This ensures that the covering condition of $\mathcal{S}$ will not be affected, i.e., $w(\{\mathcal{S} \backslash C_j\} \cup C_i) = n$. If the extended closure class of $\{\mathcal{S} \backslash C_j\}$ contains C_j (or its subset) as its member, then replacing C_j by C_i may affect the closure condition of $\mathcal{S}$, i.e., the extended closure class $\Phi(\mathcal{S})$ may no longer be empty. Compatible C_i being unimplied (Condition 2.2) ensures that neither C_i nor any of its subsets are implied by any of the prime compatibles (Definition 8). Thus, Condition 2.2 ensures that closure condition $\Phi(\mathcal{S}) = \varnothing$ will not be violated. In addition, if each member of the closure class of the set of compatibles $\{\mathcal{S}_r\} \cup C_i$ is contained in at least one member of the closure class of $\{\mathcal{S}_r\} \cup C_j$, i.e., $\phi(\mathcal{S}_i) \leq \Phi(\mathcal{S}_j)$ (Condition 2.3), then replacing C_j in $\mathcal{S}$ by C_i will not affect the closure condition of $\mathcal{S}$, i.e., $\Phi(\{\mathcal{S} \backslash C_j\} \cup C_i)$ remains empty. The number of compatibles in $\mathcal{S}$ remains unchanged when replacing C_j by C_i. Thus, the minimal condition will also be unaffected. This implies that compatible C_j can be deleted from the search tree without affecting the chances of finding the minimal closed cover, if Conditions 2.1, 2.2, and 2.3 are satisfied. □

The pruning is carried out in conjunction with tree generation as described in the previous section. When a node is expanded to the next level, its children are tested for the pruning criteria. Those satisfying the pruning criteria are deleted from the search tree. The search tree generation and pruning algorithm is shown in Fig. 4. Procedure *tree_generation* takes the ordered generating sequence and the set of prime compatibles as its input and generates a search tree with its depth equal to Ω.

Example: In the search tree shown in Fig. 2, consider the children nodes of node *hfe* at the second level, namely, *ged*, *gd*, and *ge*. Node *gd* can delete node *ge* with respect to the path consisting of nodes *ifd* and *hfe* because it satisfies all the conditions of pruning rule 2 as explained below.

1) State *e* of compatible *ge* is not covered by *gd*, but is already covered by *hfe* (the parent node), thereby satisfying Condition 2.1.
2) The compatible *ge* is unimplied. Thus Condition 2.2 is satisfied.
3) Extended closure classes $\Phi(ifd, hfe, ge)$ and $\Phi(ifd, hfe, gd)$ are empty. They satisfy Condition 2.3.

Similarly, node *a* at the fourth level (i.e., a child of compatible *ged* at level 3) is also deleted. The deleted nodes are marked in the search tree (Fig. 2).

In the following section, we describe an algorithm that finds a minimal closed cover satisfying covering and closure constraints.

```
Input  : A set of prime compatibles .
       : An ordered generating sequence (s1, s2, s3, ... sΩ).
Output : A search tree with its depth equal to Ω.

Procedure tree_generation( ) {
  initialize the root node ;
  while level l is less than the lower bound Ω {
    expand each node at level l to level l + 1 by choosing
      prime compatibles covering the state s(l+1) as its children nodes
    prune the children of each node at level l ;
    increment l to level l+1 ;
  }
  return the search tree
}
```

Fig. 4. Algorithm for search tree generation and pruning.

4.5. Searching for a Minimal FSM Solution

A path in the search tree is a minimal FSM solution if it satisfies the closure condition, the covering condition, and the minimal condition (Definition 13). An efficient heuristic that is able to reduce the search effort further is to rank the weights for nodes at the same level of the search tree. The *weight* of a node is the number of distinct states covered by compatibles on its path. To gain search efficiency, a node having a large weight is expanded before a node having a smaller weight (again, fail-first heuristic [15], [18]). This heuristic increases the chances of finding a minimal solution at an earlier stage of the search and eliminates the redundant computational effort of expanding the rest of the nodes.

For each node at level Ω, denoted as $n_{\Omega,k}$, the algorithm tests each $path_{\Omega,k}$ for covering and closure conditions. If the path satisfies both constraints, then the compatibles on the path form a closed covering. If the path does not satisfy either constraint, then node $n_{\Omega,k}$ is expanded to the next level. $path_{\Omega,k}$ and its children are then tested for a minimal solution. If the new path satisfies the covering and closure conditions, the algorithm stops and returns compatibles on the path as a minimal closed cover. A node is expanded to the next level as follows.

The covering condition can be satisfied by adding prime compatibles (i.e., children nodes at level $\Omega + 1$) that cover an uncovered state in $path_{\Omega,k}$. The closure condition can be satisfied by adding prime compatibles (i.e., children nodes at level $\Omega + 1$) that cover a member of the extended closure class of compatibles on $path_{\Omega,k}$. If $path_{\Omega,k}$ has more than one uncovered state, the rest of them are covered in the subsequent levels. Similarly, if the extended closure class has more than one member, the rest of them are covered in the subsequent levels. The nodes at lower levels can be expanded similarly. If none of the nodes at level $\Omega + 1$ satisfy the constraints, then each node $n_{\Omega+1,k}$ at level $\Omega + 1$ is expanded to next lower level, and its children are tested for a minimal solution. This process of expanding the nodes and testing the children is continued until a path satisfying the covering and closure conditions is found.

Proposition: A set of compatibles on the first path that satisfies covering and closure constraints also satisfy minimal constraint.

Proof: A path having k compatibles forms a minimal closed cover, if it satisfies covering and closure condi-

```
Input  : A set of prime compatibles.
       : Nodes at level Ω : n_{Ω,k} (k = 1,2,...,|n_Ω|).
Output : A minimal FSM solution.

Procedure search_minimal_solution( ) {
  for each node n_{Ω,k} (k = 1,2,...,|n_Ω|) {      /* test nodes at level Ω */
    S_k = compatibles on path_{Ω,k} ;
    if S_k is a closed covering (solution found)
      return path_{Ω,k} ;     stop and quit;
  }
  start from level l = Ω ;
  while no solution {
    sort nodes at level l in the ascending order of their weights ;
    for each node n_{l,k} (k = 1,2,...,|n_l|) {
      S_k = compatibles on path_{l,k} ;
      if S_k satisfies covering condition {
        if S_k satisfies closure condition (solution found)
          return path_{l,k} ;     stop and quit;
        else { /* if closure condition not satisfied, try to satisfy it at next level */
          expand node n_{l,k} to level l + 1 ;
          prune the children nodes ;
          if path of the children satisfy both constraints (solution found)
            return the path_{l,k} and the child ;     stop and quit;
        }
      }
      else {/* if covering condition not satisfied, try to satisfy it at next level */
        expand node n_{l,k} to level l + 1 ;
        prune the children nodes ;
        if path of the children satisfy both constraints (solution found)
          return the path_{l,k} and the child ;     stop and quit;
      }
    } /* for loop */
    increment l to level l+1 ;
  } /* while loop */
}
```

Fig. 5. Algorithm to find a minimal FSM solution.

tions and if there is no closed covering of $k - 1$ or less compatibles. In the search for a minimal solution, a node at level l is expanded to the next $l + 1$ level if and only if none of the paths at level l satisfy the closure and covering constraints. Thus we test a path having k compatibles for a minimal solution only after we have tested all the paths having $k - 1$ or less compatibles. This guarantees that the first path that satisfies both the covering and the closure constraints will also satisfy the minimal constraint. □

The procedure *search_minimal_solution*, illustrated in Fig. 5, takes the set of prime compatibles and the nodes generated by the *tree_generation* algorithm. It finds a minimal FSM solution, or proves that no solution exists. A complete path obtained from the procedure *search_minimal_solution* is a minimal FSM solution of the given state machine equals the number of nodes on the solution path. The search tree generation process ensures that a complete path containing l compatible must be at the lth level of the search tree. This implies that all the minimal FSM solution paths must be at the same level of the search tree as the first minimal closed covering path. Thus all the minimal FSM solutions can be generated with almost no time overhead.

Example: Fig. 2 shows the search tree for the FSM example. The paths of nodes at the fourth level, i.e., $path_{\Omega,k}$, are shown in Table V, where $\mathcal{S}_k$ denotes the set of compatibles on $path_{\Omega,k}$. The nodes are arranged in the descending order of their weights $w(\mathcal{S}_k)$. Each $path_{\Omega,k}$ in Table V is then tested for a minimal solution. None of them satisfies both covering and closure constraints. The uncovered states and the extended closure class $\Phi(\mathcal{S}_k)$ on each path are shown in the table. Since $path_{\Omega,1}$, i.e., $\mathcal{S}_1 = \{ifd, hfe, ged, cba\}$, satisfies the covering constraint,

TABLE V
PATHS AND CONSTRAINTS AT THE FOURTH LEVEL OF SEARCH TREE (FSM EXAMPLE)

$path_{\Omega,k}$ (Path of Compatibles at Level Ω)	States Covered by $\mathcal{S}_k$ (Covering Condition)	$\Phi(\mathcal{S}_k)$ (Closure Condition)
ifd, hfe, ged, cba	abcdefghi	fed
ifd, hfe, gd, cba	abcdefghi	fed
ifd, hfe, ged, ba	abdefghi	bc, ac
ifd, hfe, ged, ca	acdefghi	bc, ba
ifd, hfe, gd, ba	abdefghi	bc, ca
ifd, hfe, gd, ca	acdefghi	bc,ba
ifd, hfe, gd, a	adefghi	∅

TABLE VI
REDUCED FSM OF THE EXAMPLE STATE MACHINE *ungerex*

Present state	I_0	I_1	I_2	I_2
A	B/1	B/1	-/1	B/0
B	B/1	A/0	-/0	C/-
C	-/0	B/1	D/1	B/0
D	B/1	-/0	E/1	B/0
E	B/1	B/1	-/1	B/1

it covers all the states in *ungerex* ($w(\mathcal{S}_1) = 9$). The extended closure class of $\mathcal{S}_1$ is not empty and it contains a compatible *fed*. Thus, we expand the first node at level four to level five, to satisfy the closure condition. This requires supplementing $path_{\Omega,1}$ with compatible *fed*. Subsequently, we test the path of child *fed*, i.e., $\{\mathcal{S}_1\} \cup (fed) = \{ifd, hfe, ged, cba, fed\}$, for a minimal solution. This new path satisfies both the covering and closure conditions. It is therefore a minimal closed covering path, as shown in the highlighted lines in Fig. 2. Compatibles $\{ifd, hfe, ged, cba, fed\}$ form a minimal closed cover for FSM example *ungerex*. The reduced state machine is derived by merging states f, e, d into state A, states c, b, a into state B, states g, e, d into state C, states h, f, e into state D, and states i, f, d into state E, as shown in Table VI.

V. EXPERIMENTAL RESULTS

We have tested a large number of FSM's for state minimization. The majority of the FSM's tested were from MCNC FSM benchmark set [36]. All experiments were preformed on a SUN SPARC 1 + workstation. We have compared our results with recent state minimization results from Kannan and Sarma [20] (experiments performed on a SUN 3/60) and Hachtel *et al.* [17] (experiments performed on a DEC 3100).

Table VII illustrates the experimental results, where the second, third, and fourth columns show the number of inputs N_I, number of outputs N_O, and number of states N_S in the benchmark FSM's. The fifth and sixth columns show the lower bound L_B and the memory reduction M_R obtained using our algorithm. The last several columns give the number of states in the reduced FSM, $N_S(R)$, and the corresponding execution time (seconds), obtained using our algorithm, Kannan *et al.*'s algorithm, and Hachtel *et al.*'s algorithm.

It was observed from Table VII that approximately half the MCNC benchmarks do not have any compatible states and thus could not be reduced. FSM's whose number of states were reduced have been underlined in Table VII. In four of the MCNC FSM's, i.e., *donfile*, *modulo12*,

TABLE VII
STATE REDUCTION RESULTS WITH BENCHMARK FSM's[3]

FSM	FSM Specifications			Our Method				Kannan et al., [20]		Hachtel et al., [17]	
Name	N_I	N_O	N_S	L_B	M_R	$N_S(R)$	Time	N_S(R)	Time	N_S(R)	Time
bbara	4	2	10	7	–	7	0.00	7	2.00	7	0.02
bbsse	7	7	16	13	–	13	0.02	x	x	13	0.09
bbtas	2	2	6	6	–	6	0.00	6	0.05	x	x
beecount	3	4	7	4	11%	4	0.01	4	1.00	4	0.02
cse	7	7	16	16	–	16	0.01	16	0.73	x	x
dk14	3	5	7	7	–	7	0.01	7	0.12	x	x
dk15	3	5	4	4	–	4	0.01	4	0.10	x	x
dk16	2	3	27	27	–	27	0.02	27	2.06	x	x
dk17	2	3	8	8	–	8	0.01	8	0.10	x	x
dk27	1	2	7	7	–	7	0.01	7	0.07	x	x
dk512	1	3	15	15	–	15	0.01	15	0.25	x	x
donfile	2	1	24	1	–	1	0.01	1	x	x	x
ex1	9	9	20	18	–	18	0.01	x	x	18	0.14
ex3*	2	2	10	2	11%	4	2.51	x	x	4	1.34
ex4	6	9	14	14	–	14	0.01	14	0.17	x	x
ex5*	2	2	9	2	22%	3	0.23	4	1.00	3	0.14
ex6	5	8	8	8	–	8	0.01	8	0.10	x	x
ex7*	2	2	10	3	25%	3	0.26	4	1.00	3	0.35
keyb	7	2	19	19	–	19	0.02	19	1.97	x	x
kirkman	2	6	16	16	–	16	0.02	16	2.88	x	x
lion	2	1	4	4	–	4	0.01	4	1.00	x	x
lion9	2	1	9	4	17%	4	0.01	x	x	4	0.00
mark1	5	6	15	12	74%	12	0.02	12	1.00	12	0.09
mc	3	5	4	4	–	4	0.01	4	0.03	x	x
modulo12	1	1	12	1	–	1	0.01	1	x	x	x
opus	5	6	10	9	–	9	0.01	9	1.00	9	0.01
planet	7	9	48	48	–	48	0.11	16	15.0	x	x
planet1	7	9	48	48	–	48	0.11	x	x	x	x
pma	8	8	24	24	–	24	0.06	x	x	x	x
s1	8	6	20	20	–	20	0.02	20	0.70	x	x
s1a	8	6	20	1	–	1	0.01	x	x	x	x
s27	4	1	6	5	–	5	0.01	x	x	x	x
s8	4	1	5	1	–	1	0.01	1	x	x	x
sand	1	9	32	32	–	32	0.02	32	4.22	x	x
scf	27	56	121	97	89%	97	0.99	97	998.	97	1.75
shiftreg	1	1	8	8	–	8	0.02	8	0.05	x	x
sse	7	7	16	13	–	13	0.02	13	1.00	13	0.09
styr	9	10	30	30	–	30	0.04	30	4.10	x	x
tav	4	4	4	4	–	4	0.01	4	0.10	x	x
tbk	6	3	32	16	99%	16	1.64	16	35.0	16	4.60
tma	7	6	20	18	17%	18	0.02	x	x	18	0.05
train11	2	1	11	4	47%	4	0.01	4	1.00	4	0.02
train4	2	1	4	4	–	4	0.01	4	0.05	x	x
unger61-3	2	1	10	7	–	7	0.00	x	x	x	x
unger61-2*	2	1	9	4	31%	5	0.03	x	x	x	x
unger62-1	2	1	9	4	–	4	0.00	x	x	x	x
yang-ex1*	2	1	9	2	–	4	5.59	5	1.00	x	x
paul-ex4*	2	1	9	2	–	2	0.00	x	x	x	x
grasselli*	3	1	8	3	–	4	0.01	x	x	4	0.02
unger47-1*	2	1	8	3	31%	3	0.01	x	x	3	0.02
unger62-3*	2	1	8	2	31%	3	0.08	x	x	x	x

s1a, *s8*, all the states were compatible and thus they were merely reduced to one state machine. The execution times required to reduce the FSM's, of which all or none of the states are compatible, were negligible. For all except nine FSM benchmarks that are marked with an asterisk, the lower bound L_B on the size of the reduced FSM was the same as the number of states in the reduced state machine obtained only from maximal compatibles. In such a case prime compatibles were not required. The maximal compatibles (i.e., cliques in merger graph) were derived using Unger's implication table [33].

Statistics regarding the memory space reduction were obtained directly from the percentage reduction in number of nodes in the search tree due to pruning. The average memory space reduction increases with the search tree size. In case of FSM's with a very large number of prime compatibles (e.g. *ex2*), pruning reduces the memory requirements by 90–95%, but requires a substantial amount of time. Thus such a case involves a trade-off between memory space and execution time. For MCNC FSM benchmark *scf* having 121 states, a memory reduction of 89% was achieved, and an optimal solution was obtained in 0.99 s of CPU time. In comparison, for the same machine, it took Kannan *et al.*'s algorithm 998 s to obtain a solution (on a SUN 3/60). Also, Kannan *et al.*'s algorithm does not guarantee an optical solution and requires a large amount of time for all the benchmark FSM's.

Our algorithm compares favorably with Hachtel *et al.*'s exact and heuristic algorithms. It produced optimal results for all examples in a lesser time for almost all cases. For FSM's *scf* and *tbk*, it took our algorithm 0.99 and 1.64 s, as compared to 1.75 and 4.60 s with Hachtel *et al.*'s algorithm (on a DEC 3100). In these two cases, only one maximal incompatible having a maximum number of states (maximal clique) was derived to obtain the OGS. Because of efficient pruning criteria, the reduction of FSM's with our approach is accompanied by a substantial memory reduction for all large examples. For example, for nontrivial FSM's *scf* and *tbk*, memory reductions of 89% and 99% were achieved.

The minimal FSM solution of an incompletely specified FSM is not unique (e.g., FSM's *ex7*, *tma*, and *lion9*). These minimal FSM's may have different implementation areas. For example, FSM *ex7* has two minimal solutions with a two-level implementation area of 84 and 72 (obtained using NOVA [34]). Our algorithm selects the best solution with a minimum area of 72. In comparison, Kannan *et al.*'s algorithm generated a sub-optimal solution with an area of 108, and Hachtel *et al.*'s algorithm generated a solution with an area of 84.

VI. Conclusion

An efficient algorithm to search for the minimal closed covers in sequential machines is presented. This algorithm eliminates all the redundant states in a given state machine and guarantees to produce an optimal solution for a reduced FSM. This performance is achieved because of the application of fail-first heuristics in the search tree level and nodal ordering and taking advantage of efficient search space pruning criteria in search tree generation and in the search process.

Acknowledgment

Robert Brayton and reviewers provided constructive comments that improved the quality of this paper. Rettig Oliver and Michel Crastes sent us papers [17], [20], and [2].

References

[1] P. Ashar, S. Devadas, and A. R. Newton, "Irredundant interacting sequential machines via optimal logic synthesis," *IEEE Trans. Computer-Aided Design*, vol. 10, pp. 311–325, Mar. 1991.

[2] M. J. Avedillo, J. M. Quintana, and J. L. Huertas, "State reduction of incompletely specified finite sequential machines," in *Proc. Workshop on Logic and Architecture Synthesis*, 1990, pp. 107–115.

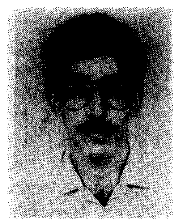
[3] R. G. Bennetts, "An improved method for prime C-class derivation in the state reduction of sequential networks," *IEEE Trans. Computers*, vol. 20, pp. 229–231, Feb. 1971.

[4] N. N. Biswas, "State minimization of incompletely specified sequential machines," *IEEE Trans. Computers*, vol. 23, pp. 80–84, Jan. 1974.

[5] N. N. Biswas, "Implication trees in the minimization of incompletely specified sequential machines," *Int. J. Electronics*, vol. 55, pp. 291–298, Jan. 1983.

[3] Execution times for all methods are given for computing the minimal closed cover and mapping the closed cover into a reduced FSM.

[6] R. Carraghan and P. M. Pardalos, "An exact algorithm for maximum clique problem," *Operations Research Letters*, vol. 9, pp. 375-382, Nov. 1990.
[7] H. A. Curtis, "The further reduction of CC tables," *IEEE Trans. Computers*, vol. 20, pp. 454-456, Apr. 1971.
[8] S. C. DeSarkar, A. K. Basu, and A. K. Choudhury, "Simplification of incompletely specified flow tables with the help of prime closed sets," *IEEE Trans. Computers*, vol. 18, pp. 953-956, Oct. 1969.
[9] S. Devadas, H. K. T. Ma, A. R. Newton, and A. Sangiovanni-Vincentelli, "Irredundant sequential machines via optimal logic synthesis," *IEEE Trans. Computer-Aided Design*, vol. 9, pp. 8-17, Jan. 1990.
[10] X. Du, G. Hachtel, B. Lin, and A. R. Newton, "MUSE: a multilevel symbolic encoding algorithm for state assignment," *IEEE Trans. Computer-Aided Design*, vol. 10, pp. 28-38, Jan. 1991.
[11] M. R. Garey and D. S. Johnson, *Computers and Intractability—A Guide to the Theory of NP-Completeness*. New York: W. H. Freeman, 1979.
[12] S. Ginsburg, "A synthesis technique for minimal state sequential machines," *IRE Trans. Electronic Computers*, vol. 8, pp. 13-24, Mar. 1959.
[13] A. Grasselli and F. Luccio, "A method for minimizing the number of internal states in incompletely specified sequential networks," *IEEE Trans. Computers*, vol. 14, pp. 350-359, June 1965.
[14] J. Gu, *Constraint-Based Search*. New York: Cambridge University Press, 1993.
[15] R. M. Haralick and G. Elliot, "Increasing tree search efficiency for constraint satisfaction problems," *Artificial Intelligence*, vol. 14, pp. 263-313, 1980.
[16] J. Hartmanis and R. E. Stearns, *Algebraic Structure Theory of Sequential Machines*. New York: Prentice-Hall, 1966.
[17] G. D. Hachtel, J. K. Rho, F. Somenzi, and R. Jacoby, "Exact and heuristic algorithms for the minimization of incompletely specified state machines," in *Proc. European Design Automation Conf.*, 1991, pp. 184-189.
[18] E. Horowitz and S. Sahni, *Fundamentals of Computer Algorithms*. Rockville, MD: Computer Science Press, 1978.
[19] R. W. House and D. W. Stevens, "A new rule for reducing CC tables," *IEEE Trans. Computers*, vol. 19, pp. 1108-1111, Nov. 1970.
[20] L. N. Kannan and D. Sarma, "Fast heuristic algorithms for finite state machine minimization," in *Proc. European Design Automation Conf.* 1991, pp. 192-196.
[21] J. Kella, "State minimization of incompletely specified sequential machines," *IEEE Trans. Computers*, vol. 19, pp. 342-348, Apr. 1970.
[22] Z. Kohavi and A. Paz, *Theory of Machines and Computations*. New York: Academic Press, 1971.
[23] Z. Kohavi, *Switching and Finite Automata Theory*. New York: McGraw-Hill, 1978.
[24] F. Luccio, "Extending the definition of prime compatibility classes of states in incompletely specified sequential machine reduction," *IEEE Trans. Computers*, vol. 18, pp. 537-540, June 1969.
[25] M. S. Meisel "A note on internal state minimization in incompletely specified sequential networks," *IEEE Trans. Computers*, vol. 16, pp. 508-509, Aug. 1967.
[26] D. Pager, "Conditions for existence of minimal closed covers composed of maximal compatibles," *IEEE Trans. Computers*, vol. 20, pp. 450-452, Apr. 1971.
[27] M. C. Paul and S. H. Unger, "Minimizing the number of states in incompletely specified sequential switching functions," *IRE Trans. Electronic Computers*, vol. 8, pp. 356-367, Sept. 1959.
[28] M. A. Perkowski and J. Liu, "Generation of finite state machines from parallel program in DIADES," in *Proc. Int. Sym. Circuits and Systems*, 1990, pp. 1139-1142.
[29] M. A. Perkowski and N. Nguyen, "Minimization of finite state machine in system SuperPeg," in *Proc. Midwest Sym. Circuits and Systems*, 1985, pp. 139-147.
[30] C. P. Pfleeger, "State reduction in incompletely specified finite state machines," *IEEE Trans. Computers*, vol. 22, pp. 1099-1102, Dec. 1973.
[31] C. V. S. Rao and N. N. Biswas, "Minimization of incompletely specified sequential machines," *IEEE Trans. Computers*, vol. 24, pp. 1089-1100, Nov. 1975.
[32] B. Reusch and W. Merzenich, "Minimal covering for incompletely specified sequential machines," *Acta Information*, vol. 22, pp. 663-678, 1986.
[33] S. H. Unger, *Asynchronous Sequential Circuits*. New York: Wiley-Interscience, 1969.
[34] T. Villa and A. Sangiovanni-Vincentelli, "NOVA: state assignment of finite state machines for optimal two-level logic implementation," *IEEE Trans. Computer-Aided Design*, vol. 9, pp. 905-924, Sept. 1990.
[35] C. Yang, "Closure partition method for minimizing incomplete sequential machines," *IEEE Trans. Computers*, vol. 22, pp. 1109-1122, Nov. 1973.
[36] S. Yang, "Logic synthesis and optimization benchmarks user guide version. Technical report version 3.0," Microelectronics Center of North Carolina, Jan. 15, 1991.



Ruchir Puri (S'91) received the B.S. degree with honors in electronics and communication engineering from the Regional Engineering College, Kurukshetra, India, in 1988, and the M.S. degree in electrical engineering from the Indian Institute of Technology, Kanpur, India, in 1990, both in electrical engineering.

He joined the Department of Electrical Engineering at the University of Calgary during 1990, where he is presently a Ph.D. candidate. He has been working on logic synthesis, design and verification of self-timed circuits, and efficient search techniques for VLSI circuit design.

Mr. Puri is a member of the IEEE Computer Society. He is a recipient of a 1993 ACM/IEEE Design Automation Scholarship and the 1992 and 1993 Alberta Microelectronics graduate research scholarships.

Jun Gu (S'86-M'90-SM'90) received the B.S. degree in electrical engineering from the University of Science and Technology of China in 1982 and the Ph.D degree in computer science from the University of Utah in 1989, where he was twice awarded the ASC Fellowship, twice awarded the University Research Fellowships, and twice awarded the ACM/IEEE academic scholarship awards.

He joined the University of Calgary in late 1989. Since 1990, he has been an Associate Professor with the Department of Electrical and Computer Engineering at the University of Calgary, where he has received substantial funding from federal governmental agencies and the industrial sector. His research interests include operations research and combinatorial optimization, applied artificial intelligence, computer-aided manufacturing, computer architecture and parallel processing, and structured techniques for VLSI circuit design. He developed many efficient search algorithms dealing with practical constrained optimization problems involving a million variables. In 1987, he provided several discrete and continuous local search algorithms for the satisfiability (SAT) problem and other NP-hard problems. He authored the book *Constraint-Based Search* (Cambridge Univ. Press, 1993).

Dr. Gu is Vice Program Chair of the 1993 IEEE International TAI Conference, Boston, MA, and an Adjunct Professor of the National Research Center for Intelligent Computing Systems and the Academy of Sciences of China. He is a member of the ACM, AAAI, International Neural Network Society, and *Sigma Xi*.