Exact and Heuristic Algorithms for the Minimization of Incompletely Specified State Machines

June-Kyung Rho, Gary D. Hachtel, *Fellow, IEEE*, Fabio Somenzi, *Member, IEEE*, and Reily M. Jacoby

***Abstract*—In this paper we present two exact algorithms for state minimization of FSM's. Our results prove that exact state minimization is feasible for a large class of practical examples, certainly including most hand-designed FSM's. We also present heuristic algorithms, that can handle large, machine-generated, FSM's. The possibly many different reduced machines with the same number of states have different implementation costs. We discuss two steps of the minimization procedure, called state mapping and solution shrinking, that have received little prior attention in the literature, though they play a significant role in delivering an optimally implemented reduced machine. We also introduce an algorithm whose main virtue is the ability to cope with very general cost functions, while providing high performance.**

I. Introduction

STATE MINIMIZATION is an important step in the design of FSM-based circuits. Though the problem has received considerable attention in the past [1]–[7] (see [8] for an extensive bibliography), it is the recent development of sequential synthesis systems that has created the need for efficient algorithms that can minimize large FSM's.

In this paper we present two exact algorithms for state minimization that we have implemented. Our results prove that, contrary to common belief, exact state minimization is feasible for a large class of practical examples, certainly including most hand-designed FSM's. However, FSM's generated by sequential synthesis systems may have many states and, in particular, many compatible states. Heuristic techniques are therefore of interest. The ones we present in this paper have been very successful in reducing time and memory requirements, without appreciably affecting the optimality of the solution.

Normally a reduction in the number of states is attempted in the hope of reducing the complexity of the resulting FSM, as measured, for instance, by the gate count of a multilevel implementation after technology mapping[1]. However, solutions with the same number of states may have different gate counts. Several steps in the algorithms influence the cost of the resulting implementation. We analyze two of them in detail, namely the mapping step (the choice of some next state entries for which multiple options exist) and the shrinking of the solution to nonprime compatibles (sets of compatible states that can be disregarded if the minimum number of states is the only concern; they are defined in Section II). We show how careful choices in these phases reduce the cost of the state-minimal machine with respect to the original machine in most cases.

We also briefly discuss the impact of more general cost functions on the optimization process. One of the two exact algorithms we present may be used, without changes, for different types of cost functions. It represents an attractive combination of generality and efficiency.

After the preliminaries of Section II, the paper discusses the exact algorithms in Section III. Section IV describes the heuristic intended to reduce the CPU and memory requirements. Section V covers the mapping problem, and Section VI is devoted to the shrinking of the solution. Finally, Sections VII and VIII present experimental results and conclusions.

II. Preliminaries

A *finite state machine* (FSM) is defined as a quintuplet $M = (I, O, S, \delta, \lambda)$, where I is a finite nonempty set of inputs, O is a finite nonempty set of outputs, S is a finite nonempty set of states, $\delta : I \times S \to S$ is the next state function, and $\lambda : I \times S \to O$ (for a Mealy machine), or $\lambda : S \to O$ (for a Moore machine) is the output function. An FSM is *incompletely specified* if either δ or λ is not defined for one or more elements of its domain.

An FSM can be described in several ways. In a *flow-table* description of an FSM, one row of the table corresponds to a state and a column corresponds to an element of I (the input alphabet). By contrast, in a *cube table* representation, one row of the table corresponds to one edge of the state transition graph, i.e., it specifies for a given present state, and a given input value, the next state and the output value. We will denote a cube table by $F = \{F_i\}$, where F_i is a cube with four fields: the input state $\mathrm{IS}(F_i)$, the present state $\mathrm{PS}(F_i)$, the next state $\mathrm{NS}(F_i)$, and the output state $\mathrm{OS}(F_i)$. We shall use both representations in the sequel.

The flow table of Fig. 1 specifies an FSM described in [3], which will be used as an example in the following. Roughly speaking there is one row in the cube table for each nontrivial entry in the flow table, so in the sequel we will use the flow table in the examples for compactness. As an example, the row for the top-left entry of the flow table of Fig. 1 would be:

$$x1 \quad a \quad a \quad 0.$$

Manuscript received June 5, 1991; revised May 18, 1992. This work was supported in part by NSF/DARPA grant MIP-8719546. This paper was recommended by Associate Editor R. Brayton.

J.-K. Rho, G. D. Hachtel, and F. Somenzi are with the University of Colorado, Boulder, CO 80309–0425.

R. M. Jacoby is with Cadence Design Systems, Lowell, MA 01852–4995.

IEEE Log Number 9212231.

[1] Another objective may be increased testability.

0278–0070/94$04.00

	$x1$	$x2$	$x3$	$x4$	$x5$	$x6$	$x7$
a	$a,0$	$-,-$	$d,0$	$e,1$	$b,0$	$a,-$	$-,-$
b	$b,0$	$d,1$	$a,-$	$-,-$	$a,-$	$a,1$	$-,-$
c	$b,0$	$d,1$	$a,1$	$-,-$	$-,-$	$-,-$	$g,0$
d	$-,-$	$e,-$	$-,-$	$b,-$	$b,0$	$-,-$	$a,-$
e	$b,-$	$e,-$	$a,-$	$-,-$	$b,-$	$e,-$	$a,1$
f	$b,0$	$c,-$	$-,1$	$h,1$	$f,1$	$g,0$	$-,-$
g	$-,-$	$c,1$	$-,-$	$e,1$	$-,-$	$g,0$	$f,0$
h	$a,1$	$e,0$	$d,1$	$b,0$	$b,-$	$e,-$	$a,1$

	Classes	Class Sets
1	$\{a,b,d,e\}$	$\emptyset$
2	$\{b,c,d\}$	$\{(a,b),(a,g),(d,e)\}$
3	$\{c,f,g\}$	$\{(c,d),(e,h)\}$
4	$\{d,e,h\}$	$\{(a,b),(a,d)\}$
5	$\{b,c\}$	$\emptyset$
6	$\{c,d\}$	$\{(a,g),(d,e)\}$
7	$\{c,f\}$	$\{(c,d)\}$
8	$\{c,g\}$	$\{(c,d),(f,g)\}$
9	$\{f,g\}$	$\{(e,h)\}$
10	$\{d,h\}$	$\emptyset$
11	$\{a,g\}$	$\emptyset$
12	$\{f\}$	$\emptyset$

Fig. 1. The Grasselli-Luccio example: The flow-table (top) and the list of prime compatibles (bottom).

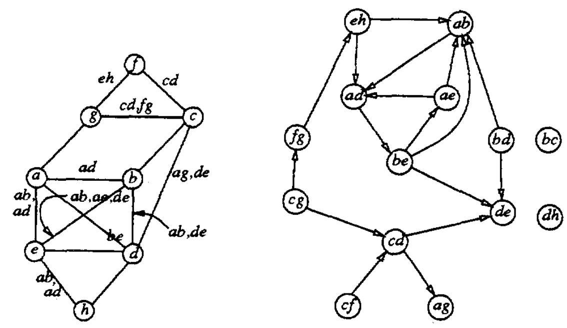


Fig. 2. Merge and compatibility graphs for the Grasselli-Luccio example.

An input sequence is *applicable* to state S_i of machine M if no unspecified next state transitions are encountered. Two states S_i and S_j are *compatible* if and only if for every input sequence that is applicable to both S_i and S_j, nonconflicting output sequences are produced. A set of states is compatible if and only if each pair of states in the set is compatible. A set of compatible states (a *compatible* for short) is *maximal* if it is not a proper subset of another set of compatible states. A state is *incompatible* if it is not compatible to any other state in S.

If there is an input such that k is the next state of i and l is the next state of j and $(k,l) \neq (i,j)$, then we say that (k,l) is *implied by* (i,j). Illustrated in Fig. 2 are two graphs, [9], called the *merge graph* G_M (left) and the *compatibility graph* G_C (right). The merge graph has one node for each state. There is no edge between nodes i and j (states i and j are incompatible) if there is an input such that states i and j produce conflicting output values, or if some other pair (k,l) (transitively) implied by (i,j) produces conflicting output values. There is an edge without label between nodes i and j (states i and j are fully compatible) if states i and j have no conflicting next output values and the state pairs they imply are fully compatible. Finally there is a labeled edge between nodes i and j (states i and j are conditionally compatible) if states i and j have no conflicting output values, but at least one pair of next states are neither incompatible nor fully compatible. The label consists of all such state pairs. The compatibility graph is a directed graph with one vertex for each pair of compatible states. $((i,j),(k,l))$ is an arc of G_C if (i,j) implies (k,l).

A set of compatibles is *closed* if for every compatible contained in the set, all the implied compatibles are contained in the set. A *closed covering* is a closed set of compatibles in which each state appears in at least one set. The *class set* of a compatible C is the set of compatibles $\mathcal{C}_C$, which are: a) Implied by C, that is, if C is to be part of a closed covering, all members of $\mathcal{C}_C$ must be included in at least one member of the closed covering, and b) not contained in C or any other member of $\mathcal{C}_C$. A compatible C is *prime*, if no other compatible C^+ exists for which:

$$C \subseteq C^+$$
$$\mathcal{C}_C \supseteq \mathcal{C}_{C^+}$$

In words, a compatible is not prime if it is included in another compatible and it has all the implications of the other compatible. The prime compatibles and their corresponding class sets are shown at the bottom of Fig. 1. Note that the first four primes and the eleventh are just the maximal compatibles (maximal cliques of the merge graph). From the definition, all maximal compatibles are prime and therefore there are more primes than maximals. It has been observed in the past [10, pg. 410] that the FSM's that do not have minimum solutions composed of maximal compatibles only are very rare. Though this appears to be the case for isolated, hand-designed machines, we have routinely observed FSM's that do require prime compatibles to get the minimum state solution in the optimization of interacting FSM's [11].

We shall have occasion to refer in the sequel to upper and lower bounds on the number of states in the minimized machine [9]. Of course, the number of states in the original FSM, and the total number of maximal compatibles are upper bounds; a tighter upper bound, b_U, is just the size of the minimum closed covering composed of maximal compatibles only. A lower bound, b_L, is the optimal solution of the same problem, except the closure requirement is dropped.

III. Exact State Minimization

We first discuss the methods for solving the problem of finding a machine with the smallest number of states which covers the specified machine. That is, the problem is to find the smallest closed set of compatibles that cover all the states in the original machine. We shall investigate two methods, and we start by recalling the essential facts about covering problems that will be useful in their discussion.

3.1. Covering Problems

The covering problem is the problem of selecting elements from a collection of subsets of a set S in such a way that the union of the selected subsets is S and that the cost of the selected subsets is minimum. A well-known example is the determination of a minimum cost cover, given the prime implicants of a Boolean function [12]. Solving the covering problem amounts to finding the minimum cost assignment satisfying a *unate* Boolean formula in conjunctive form [13].

The *binate* covering problem (BCP) is the generalization of the (unate) covering problem where the Boolean formula is not restricted to be unate, and therefore can express other constraints than just coverage.

In state minimization there are actually two sorts of constraints corresponding to the properties of coverage and closure of a solution. With reference to Fig. 1, the condition that state a must be covered by at least one of the prime compatibles is expressed as $(C_1 + C_{11})$, since C_1 and C_{11} are the only prime compatibles containing state a. In similar fashion the coverage of all the other states can be expressed as a set of unate clauses, or in other words, a set of unate rows of the coverage matrix.

Closure constraints, on the other hand, express implications. If prime $C_6 = \{c, d\}$ is to be part of the solution, then there must be other compatibles in the solution that contain all its implied classes, namely $\{a, g\}$ and $\{d, e\}$. Since $\{a, g\}$ is only contained in C_{11}, the selection of C_6 implies the selection of C_{11}. This can be written as $(\overline{C}_6 + C_{11})$. Compatible $\{d, e\}$, however, is found in both C_1 and C_4, hence the constraint $(\overline{C}_6 + C_1 + C_4)$.

There are several methods of solving the binate covering problem. We briefly outline one branch-and-bound method based on column splitting. The reader is referred to [3], [14], [15] for the details. The formula expressing the constraints can be put in matrix form by assigning a column to each variable and a row to each clause. Entry i, j is 2 if variable j does not appear in clause i, is 1 if it appears there uncomplemented, and is 0 otherwise.

The matrix is first simplified as much as possible by finding essential columns and applying row and column dominance, much in the same way as in the unate case. Then a column is tentatively selected and a solution is recursively sought for the residual problem under that assumption. The column is then rejected and another solution is determined. The optimum solution is the best of the two.

The algorithm prunes the search space by keeping up-to-date upper and lower bounds. If the lower bound (the sum of the costs of the partial solution and a lower bound on the cost of the residual problem) is greater than the upper bound, then the recursion is terminated. It is important to note that column dominance and bounding rely on the additive property of cost.

3.2. The Binate Covering Method

The binate covering method was developed by Grasselli and Luccio [3]. This method converts state minimization to a binate covering problem, hence the name. It can be summarized as follows.

a) Form the merge graph, G_M.
b) Find maximal compatibles.
c) Find lower bound b_L as discussed in Section II. If b_L is closed, stop. (b_L is an optimum solution [9, pg. 340]).
d) Find prime compatibles and class sets.
e) Form binate covering table of rows (state covering or closure constraints) and columns (prime compatibles).
f) Solve with a generic binate covering algorithm package.

The covering matrix for the binate covering algorithm has two parts. The first part is the "normal" part, which requires that the current solution (that is, set of compatibles being considered) covers all states of the given machine. This sub-matrix is unate, that is, each of its entries M_{ij} is 2 (meaning compatible j does not contain state i) or 1 (meaning that it does contain it). The second part imposes the constraint of closure on the considered solutions. There is one row in the second part of the covering matrix for each element of the class set of every prime compatible C. The formation of these rows has been illustrated in Section 3.1. The cost of each column is 1. Some results of state minimization using this method are presented in Section VII.

3.3. The Closed Compatible Pair Set Covering Method

This method is analogous to that described in [16], and can be derived by replacing prime compatibles with compatible pairs, and then proceeding similarly except for cost evaluation. It can be summarized as follows.

a–c) As in the binate covering method.
d) Form the compatibility graph G_C, as described in Section II. The nodes of G_C are compatible pairs, and the edges correspond to implied pairs.
e) Apply Tarjan's algorithm for the strongly connected components of a digraph [17] to find all closed subgraphs of G_C. Each closed subgraph identifies a set of edges (compatible pairs) of G_M that is called a *prime closed edge set*. Any closed set of compatible pairs is a member of the power set of the set of prime closed edge sets; hence the name.
f) Form a covering matrix, M, whose rows correspond to states and columns to closed subgraphs. The problem is to find a column covering set, which covers all rows, and has minimum cost. This is done with a branch and bound algorithm closely resembling that of the binate covering algorithm discussed above.
g) The cost of a solution is, unfortunately, also the solution of a covering problem, namely, the size of the minimum closed clique covering of all the edges in the subgraph of G_M induced by the current solution. Note that the current solution is just a set of closed subgraphs of G_C. Each closed subgraph has a specific set of nodes of G_C, and hence, corresponding edges of G_M. An ordinary unate covering routine may be called to find a tight lower bound on the size of the minimum clique covering of all edges of this subgraph of G_M. The edges corresponding to unconditionally compatible pairs are always kept in the subgraph.
h) If the solution found in Step g) is closed then the bound is sharp, and we may proceed with the next level of recursion. Else, the solution can be "shrunk"

as discussed in Section VI. If closure cannot be gained by shrinking, then the method resorts to binate covering on the current subproblem. Fortunately this is a very rare occurrence, based on the limited data we have at this point.

The prime closed edge sets for the example of Fig. 1 are:

$$\begin{array}{lll lll} P_1 & = & \{bc\} & P_2 & = & \{dh\} \\ P_3 & = & \{de\} & P_4 & = & \{ag\} \\ P_5 & = & \{cd\} \cup P_3 \cup P_4 & P_6 & = & \{cf\} \cup P_5 \\ P_7 & = & \{ab, ad, be, ae\} \cup P_3 & P_8 & = & \{bd\} \cup P_7 \\ P_9 & = & \{eh\} \cup P_7 & P_{10} & = & \{fg\} \cup P_9 \\ P_{11} & = & \{cg\} \cup P_5 \cup P_{10}. & & & \end{array}$$

A cover of the matrix of Step f) is given by P_1, P_8, and P_{10}. The solution of the clique covering problem for these sets yields the following closed cover,

$$(a, b, d, e), (d, e, h), (b, c), (f, g),$$

as in the binate covering method. Some experimental results for this method are presented in Section VII. These results show this method to be surprisingly fast (this is discussed in Section 3.4), although implementation is still incomplete in some respects.

3.4. Cost Functions Other than the Number of States

Here we shall consider briefly the more general problem of synthesizing the smallest cost (e.g., minimum gate count), or most encodable machine, which covers the original. Since the cost function for this more important problem is much more complicated, some of the efficiencies of the listed methods will disappear, due to the loss of important mechanisms which were formerly operative.

For example, the binate covering method has sophisticated and proven procedures for row and column dominance and lower bound computation. Further, instead of having to search the space of all closed coverings, this method needs only to search the space of all closed coverings of prime compatibles. At first glance, this seems far more efficient than the closed compatible pair set method, which presumably would have to search a much larger space. Experimentally, however, we have found that this is not always the case. But even when it is the case, there is still an intrinsic virtue to the compatible pair covering method: Because its cost function is not assumed to be additive, it can be completely arbitrary, and therefore this method should work about as well for finding an "optimally encodable" result as it would for finding a result FSM with a minimum number of states.

IV. Heuristic State Minimization

As shown in Section VII, most of the examples we have encountered so far are amenable to the exact solution of the state minimization problem. There are a few cases, however, where the covering problem cannot be solved, or even formulated, efficiently. In this section we introduce two heuristic techniques that have reduced substantially the time and memory requirements for the most difficult examples, while providing excellent results in terms of optimality for all examples for which an exact solution was known.

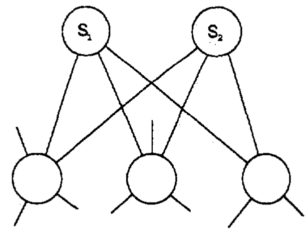


Fig. 3. Isomorphic states.

4.1. Isomorphic States

Definition 1: Two states s_1 and s_2 are isomorphic *if for every edge of the merge graph (s_1, s_i) there exists an edge (s_2, s_i) and vice versa.*

Fig. 3 illustrates the definition. Notice that since the merge graph does not contain self loops (edges of the form (s_i, s_i),) two isomorphic states are necessarily incompatible (or else, the presence of the edge (s_1, s_2) would imply the presence of (s_1, s_1) and (s_2, s_2)). It is readily seen that state isomorphism is transitive. Also, if s_1 and s_2 are two isomorphic states, then for every maximal compatible containing s_1, there is a corresponding maximal compatible containing s_2. The latter maximal compatible is obtained from the former by simply replacing s_2 for s_1. This results generalizes straightforwardly to sets of isomorphic states of arbitrary cardinality, thanks to transitivity.

One immediate application of this result allows one to reduce the computation required to find all maximal compatibles. From every set of isomorphic states of merge graph G_M, one *base* state is selected. Then all the other states in the set and their edges are removed from the merge graph. If G_M^R is the resulting graph, we can prove the following simple result, by induction on the number of isomorphic state sets.

Lemma 1: The maximal compatibles of G_M can be computed by finding the maximal compatibles of G_M^R and then adding to those all the compatibles obtained by replacing the other isomorphic states for each base state in every maximal compatible where it appears.

When the number of maximal compatible is very high, the detection of isomorphic states by itself is not sufficient (see example *jac4* in Section VII). However, it can form the basis for a heuristic algorithm that is now outlined.

1. Find isomorphic states and compute maximal compatibles of the residual merge graph only;
2. Add a minimal number of maximal compatibles to cover the (nonbase) isomorphic states;
3. Make the resulting set of maximal compatibles closed;
4. Generate the prime compatibles from the set of maximal compatibles obtained so far and solve the covering problem.

The addition of maximal compatibles to cover the isomorphic states is based on a simple greedy strategy: Among the computed maximal compatibles, the one containing the most base states is selected. Making the set of maximal compatibles closed, on the other hand, requires checking the class sets of the newly added maximal compatibles, possibly adding those maximal compatibles that contain those implied classes not included in any already selected class.

4.2. Tight Upper Bound

In cases where the number of prime compatibles is very large, the following heuristic method has been found useful.

1. Find the maximal compatibles;
2. Find the minimum closed cover composed of maximal compatibles only (b_U of Section II);
3. Compute the prime classes contained in the maximal compatibles of b_U;
4. Solve a covering problem with the set of compatibles obtained by adding the generated prime compatibles to the maximal compatibles b_U.

The rationale for this procedure is as follows. In many cases, Step 2 gives an optimum solution. For those cases when this is not true, the primes added are a small set of primes that requires no additional primes to guarantee closure, thus preventing excessive increase of computation time. Furthermore, they are selected from the maximal compatibles of Step 2, because these are (heuristically) good maximal compatibles. By replacing a maximal compatible with a prime derived from it, we hope to reduce the closure constraints, and thus be able to drop maximal compatibles that were included not to satisfy covering, but closure.

V. Mapping

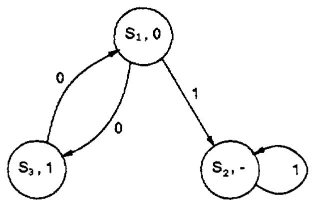
Here we consider the problem of optimally mapping the reduced machine into cube table format, suitable for input to an FSM state encoding algorithm. We introduce the problem by means of a simple example.

Example 1: Consider Fig. 4 representing a simple Moore machine. Fig. 4(a) gives the state transition graph, and Fig. 4(b) the equivalent flow table. The merge graph is given in Fig. 4(c), where the dotted lines visualize the compatibles used in the solution. It should be noted that state S_2 appears in two different compatibles included in the solution. More specifically, the implied class for both $\{S_1, S_2\}$ and $\{S_2, S_3\}$ under input 1 is $\{S_2\}$. So, in forming the reduced flow table (see Fig. 4(d)) we can satisfy the closure constraints in two different ways for two entries. Clearly, not all four combinations are equally effective.

The mapping problem can be stated as follows.

- **Given:** A closed set of compatibles which covers all the states of the original machine.
- **Find:** A mapping of the implied classes into the compatibles, so as to minimize the cost of the resulting machine.

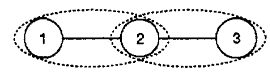
Though the problem has been known for quite some time [18], it has received relatively little attention. (One exception is [19].) Ideally, the cost should reflect attributes like size, speed, and testability of the implemented machines. Since the effects of a given choice percolate through encoding, logic minimization, and technology mapping, before they can be assessed precisely, any method working atthe flow-table level



(a)

	0	1	
1	3	2	0
2	-	2	-
3	1	-	1

(b)



(c)

	0	1	
{1,2}	{2,3}	{1,2} {2,3}	0
{2,3}	{1,2}	{1,2} {2,3}	1

(d)

Fig. 4. Example of mapping problem.

must be heuristic. Our approach consists of anticipating the effects of mapping on heuristic encodability measures such as those from MUSTANG [20] JEDI [21], or MUSE [22]. We solve the mapping problem in two phases.

1. Form a **symbolic** cube table where the next state field of each cube is a set of states, rather than a single state.
2. Solve the optimization problem of selecting a unique representative from each set of such options.

5.1. Forming the Symbolic Cube Table

Fig. 5 presents algorithm CUBE_MAP to form the symbolic cube table of the minimized machine, given the original cube table F, and the specified closed covering γ, obtained by one of the methods presented in Section III. Note that the disjoint input alphabet is computed at Line 1. This entails splitting the original cubes so that the input parts of two cubes are either identical or nonintersecting (Line 2). The unique input parts form the disjoint input alphabet. By this process we are effectively creating a cube table whose rows are in one-to-one correspondence with the entries of the full-blown flow table, with the exception that the next state entries will be sets of next states. This may be a problem for large machines. On the other hand, by working with a fine grain representation of the cube table, we can exploit possibilities for cube merging that would otherwise go unnoticed. The actual procedure that we have implemented differs from CUBE_MAP in two respects: a) the disjoint input alphabet is computed only for the states that are not incompatible; b) if the full flow table becomes too large, the procedure resorts to different alphabets for different rows.

5.2. Selecting a Unique Mapped Representation

Given a symbolic cube table such as that produced by CUBE_MAP, the next problem to be solved is to select a unique representative from each of the next-state sets. One could observe that this table is a symbolic relation (i.e., analogous to a Boolean relation with symbolic input and output fields,) and therefore, by the procedure presented in [23], one could determine the set of unique representatives which minimizes

Procedure CUBE_MAP(F, S, γ) {
  $F^* = \emptyset$; $n = 0$
  A^D = DISJOINT_INPUT_ALPHABET(F)
  $\{E_k\}$ = CUBE_SPLITTING$(F,\ A^D)$
  foreach ($input \in A^D$) {
    foreach ($C^j \in \gamma$) {
      $n = n + 1$
      $R = \{\ k \mid \text{IS}(E_k) \equiv input\ ,\ \text{PS}(E_k) \in C^j\ \}$
      $\mathcal{M}^n = \{\ i \mid C^i \supseteq \sum_{k \in R} \text{NS}(E_k)\ \}$
      $K_n = |\mathcal{M}^n|$
      $F^* = F^* \cup \{\ input,\ j,\ \mathcal{M}^n,\ \prod_{k \in R} \text{OS}(E_k)\ \}$
    }
  }
  return ($(F^*, \mathcal{M}, K)$)
}

INPUT: F — A machine M in cube table format. S — The set of states of M. γ — A closed covering $\{C^1,\ C^2,\ \ldots\}$ of M.

OUTPUT: F^* — The minimized machine in cube table format, where each state corresponds to a specific member $C^i \in \gamma$.

OUTPUT: $\mathcal{M}$ — A set of sets, one for each row in the mapped machine, representing the set of alternatives for selecting the next state for that cube of the reduced machine.

OUTPUT: K — A vector whose elements K_n are the number of mapping alternatives for each row of F^*. That is, K_n is the number of ways to select the next state entry of F^*_n, while maintaining the closure property.

Fig. 5. Procedure CUBE_MAP.

row count after symbolic minimization. The solution of the problem also provides the optimum encoding as a by-product.

However, we also observe that this problem may be too hard to solve practically, and, furthermore, cube count in the PLA FSM representation is not the first objective of multi-level targeted state-assignment algorithms. So, we consider here algorithms for heuristically selecting from among the next-state mapping alternatives, so as to optimize the encodability of the resulting cube table FSM specification, by anticipating the effects of mapping on encoding algorithms.

Let T be the symbolic flow table returned by CUBE_MAP, and let N_{ij} be the set of next state alternatives for entry T_{ij}, that is, for present state i, and input j. Note that if N_{ij} is a singleton, then the mapping for state i and input j is fixed.

Our heuristic method estimates the literal savings made possible by a particular choice. The savings taken into account are due to distance-1 merging. A cube $\mathbf{c}$ can merge with another cube $\mathbf{c}'$ if PS($\mathbf{c}$) = PS($\mathbf{c}'$) and if IS($\mathbf{c}$) and IS($\mathbf{c}'$) are distance-1 apart. Alternatively, $\mathbf{c}$ can merge with $\mathbf{c}'$ if IS($\mathbf{c}$) = IS($\mathbf{c}'$) and if PS($\mathbf{c}$) and PS($\mathbf{c}'$) receive adjacent encodings.

Even if the above conditions are not strictly met, the extraction of common cubes is still possible. The size of the cube is given by the number of literals common to the two cubes. Based on these considerations, the mapping procedure works as follows.

1. For each next-state set N_{ij} consider every element s_k in turn and determine its row and column values as explained in the following;
2. Let the row (column) candidate for entry i, j be the element of N_{ij} with the highest row (column) value;
3. Choose between the row and column candidates the one having the highest product of its two values.

The row value of s_k is computed as:

$$\sum_p L^i_{jp} \lambda^k_{ip}$$

where L^i_{jp} is the sum of the number of don't cares in the intersection of the input parts and of the number of ones in common in the output parts of the cubes corresponding to T_{ij} and T_{ip}, and λ^k_{ip} is 1 if $s_k \in N_{ip}$ and 0 otherwise. For the column value, we do not know the distance of the encodings of two states in advance. So, we assume that that distance is 1 for every pair. The column value is thus computed as:

$$\sum_p K^j_{ip} \kappa^k_{pj}$$

where K^j_{ip} is analogous to L^i_{jp}, only for entries T_{ij} and T_{pj}, and κ^k_{pj} is 1 if $s_k \in N_{pj}$ and 0 otherwise. The number of input literals in common is relevant because different "rows" of the table may have different input alphabets.

VI. Shrinking the Compatibles in the Solution

Though a solution with the minimum number of states can always be found by restricting attention to prime compatibles only, one may be interested in exploring solutions containing nonprime compatibles in order to minimize other indicators like the gate count after encoding, logic minimization, and technology mapping. As an example, we consider the FSM of Fig. 1. The only four-state solution composed of prime compatibles is

$$(a, b, d, e)(d, e, h)(b, c)(f, g)$$

TABLE I
EXPERIMENT SUMMARY

FSM	N_i/N_0	N_s	N_{com}	N_{max}	N_{prime}	N_{incom}	N_{iso}	literals original	literals reduced
bbara	4/2	10	6	1	1	6	0	54	45
bbsse	7/7	16	36	11	11	2	11	111	103
beecount	3/4	7	4	4	7	0	2	31	24
ex1	9/19	20	2	2	2	16	0	281	198
ex2	2/2	19	129	36	1366	0	0	118	27
ex3	2/2	10	37	10	91	0	0	57	23
ex5	2/2	9	26	6	38	0	0	45	14
ex7	2/2	10	32	6	57	0	0	60	20
lion9	2/1	9	9	5	5	0	0	15	13
mark1	5/16	15	20	12	18	0	11	73	69
opus	5/6	10	1	1	1	8	0	69	63
scf	27/56	121	70	12	90	85	0	820	754
sse	7/7	16	36	11	11	2	11	111	103
tbk	6/3	32	16	16	48	0	0	208	210
train11	2/1	11	25	5	16	1	0	22	12
grasselli	3/1	8	14	5	12	0	0	51	24
luccio	2/1	6	8	3	15	0	0	20	6
house	2/1	22	104	30	261	0	0	82	38
unger65	2/1	8	10	3	14	0	0	25	7
pager	3/5	22	65	32	71	2	5	89	42
palama	2/1	6	10	5	11	0	0	29	12
tma	7/6	20	15	15	15	5	13	121	126
lbk	6/3	32	16	16	48	0	0	261	204
green	21/17	53	305	524	524	0	46	4364	3690
jac1	13/9	32	109	109	109	9	20	499	485
jac2	11/10	30	66	31	64	2	7	447	410
jac3	9/19	48	410	618	1731	4	14	513	603
jac4	10/1	65	1514	32	803	1	52	370	278
TOTAL		667	3095	1561	5440	143	192	8946	7603

A four-state solution including nonprime compatibles is

$$(a, b, d, e)(e, h)(c)(f, g).$$

When subjected to the same logic synthesis procedure (see Section VII for the details), the former resulted in 26 literals and the latter in 24. The area after technology mapping was reduced by 6%.

The number of all compatibles, including those that are not prime, is generally too large to allow an exhaustive search of the optimum solution. In Section 3.3 a method has been described that avoids the computation of the prime compatibles. Here we present a post-processing technique that tries to shrink the compatibles in a closed cover, in the attempt to maximize the number of don't care entries in the minimized flow table. Shrinking a compatible C_α means finding a compatible $C'_\alpha \subset C_\alpha$ that can replace for C_α, without violating covering and closure constraints. The purpose is similar to the one pursued in [24], but does not require an increase in the number of prime compatibles. A different approach to finding solutions not restricted to prime compatibles is described in [25].

Let $\gamma = \{C_1, \ldots, C_n\}$ be a closed cover. A generic compatible C_α may contain states that are not found in any other compatible of γ. These states are said *essential* in this context and they cannot be removed from C_α. The nonessential states may be removed from C_α, if doing so does not violate the closure constraints.

TABLE II
RESULTS OF EXACT MINIMIZATION WITH BINATE COVERING

FSM	N_s	literals	time (s) cover	time (s) map	time (s) total
bbara	7	45	0.00	0.00	0.00
bbsse	13	103	0.00	0.15	0.17
beecount	4	24	0.00	0.00	0.01
ex1	18	198	0.00	0.02	0.06
ex2	5	27	4669.44	0.01	4702.22
ex3	4	23	0.66	0.01	0.80
ex5	3	14	0.05	0.00	0.07
ex7	3	20	0.10	0.00	0.16
lion9	4	13	0.00	0.00	0.00
mark1	12	70	0.00	0.19	0.21
opus	9	63	0.00	0.00	0.00
scf	97	786	0.00	0.05	0.75
sse	13	103	0.00	0.16	0.17
tbk	16	210	0.00	1.65	3.86
train11	4	12	0.00	0.00	0.01
grasselli	4	24	0.00	0.00	0.02
luccio	2	6	0.00	0.00	0.01
house	9	45	12.33	0.04	12.91
unger65	3	7	0.00	0.00	0.01
pager	10	42	0.00	0.02	0.04
palama	3	12	0.00	0.00	0.01
tma	18	126	0.00	0.05	0.05
lbk	16	204	0.00	1.64	4.09
green	37	3690	0.55	520.08	732.98
jac1	21	485	0.23	0.38	0.73
jac2	14	410	0.02	0.62	0.99
jac3	19	603	2304.65	0.90	2361.51
TOTAL	368	7365	6988.03	525.97	7821.84

There are two reasons why removal of nonessential states may destroy closure. The state is part of an implied class of some other compatible, or the reduced compatible has a larger class set. Based on this observation, the following simple procedure can be devised. Every nonessential state is considered in turn and tentatively removed. The resulting cover γ' is then checked for closure. If the answer is positive, then γ' replaces γ.

We have found this strategy quite effective in improving the quality of the solutions. However, the shrunk solution may be worse than the original one when the number of mapping choices is significantly reduced. This is taken into account by performing the shrinking process in two phases. The first phase takes place before mapping and is *constrained*: A move is accepted only if the number of mapping choices is not decreased. The second phase is performed after mapping and is unconstrained.

VII. EXPERIMENTAL RESULTS

Table I lists the main features of our experiments. The FSM's come from different sources. Many come from the MCNC benchmark set [26]. Actually all the MCNC FSM's were run. Those not reported, either have no compatible states, or have all states compatible, and hence degenerate to combinational logic, once minimized[2]. For all the cases not reported, the processing times were negligible. Our test suite also contains some contrived examples intended to put the

[2]These examples are *donfile*, *modulo12*, *s1a*, and *s8*.

TABLE III
RESULTS OF TECHNOLOGY MAPPING

	literals		area (μm^2)		delay (ns)	
FSM	original	minimized	original	minimized	original	minimized
bbara	54	45	48720	37584	15.21	8.46
bbsse	111	103	95584	90480	18.12	21.58
beecount	31	24	27840	22736	6.94	5.91
ex1	281	198	232928	164720	21.11	23.57
ex2	118	27	98368	23664	29.40	6.53
ex3	57	23	49648	19952	12.88	6.09
ex5	45	14	38048	12528	8.67	2.35
ex7	60	20	52896	15776	12.83	5.80
lion9	15	13	12992	12064	3.51	4.17
mark1	73	69	67280	66816	10.33	9.98
opus	69	63	61712	56608	11.22	13.74
scf	820	754	683008	624544	54.35	49.45
sse	111	103	95584	90480	18.12	21.58
tbk	208	210	174000	182352	28.16	35.51
train11	22	12	18560	12064	3.23	5.22
grasselli	51	24	45008	22272	10.54	7.80
luccio	20	6	16704	4640	6.93	2.76
house	82	38	66816	35264	19.71	7.91
unger65	25	7	22736	7888	9.63	2.17
pager	89	42	75632	40368	12.33	8.43
palama	29	12	26448	9744	7.89	5.58
tma	121	126	108112	103936	32.41	22.02
lbk	261	204	217152	169360	39.31	31.24
green	4364	3690	3130608	2670784	44.60	36.44
jac1	499	485	422240	393936	21.37	28.04
jac2	447	410	371200	343360	21.47	32.79
jac3	513	603	416672	494160	24.25	41.70
jac4	370	278	310416	238960	23.99	39.59
TOTAL	8946	7603	6986912	5667040	528.51	486.41

algorithms under strain. This is the case of *jac4*, that has 3 859 641 maximal compatibles (the number in Table I is the results obtained by the isomorphic heuristic) and, to a lesser extent, of other examples. In Table I, N_i, N_o, and N_s are the numbers of inputs, outputs, and states before minimization, respectively. The other columns show the numbers of compatible pairs (N_{com}), maximal compatibles ($N_{\max}$), and prime compatibles (N_{prime}); the number of incompatible states (N_{incom}), and the number of states that are isomorphic to some other states (N_{iso}). Finally, the two rightmost columns report the number of literals after encoding with MUSE and optimization with MIS2.2 [27] (standard Boolean script preceded by *cspf_simplify* if external don't cares exist [28]) for the original and the minimized machines. The results for the minimized machines are the best between those of Tables II and VI. In all tables, times are referred to DECstations 5000/200, except when otherwise stated.

Table II gives the results obtained with the exact algorithm based on the method of Grasselli and Luccio and binate covering. There, N_s represents the minimum number of states of a cover of the machine. A break-down of the execution times is given to show that, problems may actually arise in all phases of the algorithm, especially in the solution of the covering problem. The map time is the time required to build the reduced flow table, given the set of compatibles. It is apparent that most cases are amenable to exact solution. We didn't manage to find the exact solution for *jac4*, due to the aforementioned problem (over three million maximal compatibles).

TABLE IV
COMPARISON OF MAPPING HEURISTICS

	options				
FSM	edges	total	h1	h2	h3
train11	3	6	12	12	12
beecount	8	16	24	23	24
ex2	5	16	27	27	27
ex3	4	8	25	22	23
ex5	1	3	14	14	14
ex7	2	5	20	20	20
lion9	2	4	13	13	13
grasselli	8	16	24	24	24
house	7	18	41	45	45
pager	1	5	42	48	42
palama	1	2	12	12	12
tma	26	79	124	123	126
unger65	1	2	10	7	7
jac1	22	44	485	504	485
jac2	58	143	462	418	410
jac3	29	70	611	611	603
TOTAL	178	437	1946	1923	1887

Table III compares the results of applying encoding, logic optimization, and technology mapping to both the original and the minimized machines. As in Table I, the best result between Table II and Table VI was used for each machine. The data reported refers to the combinational logic of the FSM's. Routing area is not included. MIS2.2 was used for technology mapping [26] with the *lib2* library from MCNC [26]. Both the literal count and the total cell area decreased by about 15%. Delay decreased by only 8% and, unlike area, there are in this case big losses as well as big wins. This reflects the fact that no special consideration is paid to delay in the optimization process and indicates an area for future research. Not shown in the table is the reduction of the flip-flop count by 21%.

The comparison of three mapping heuristics is reported in Table IV. The first method (*h1*) is simply picking the first element in each set of next states[3]. Method *h2* consists of counting the number of occurrences of each possible choice in the other entries of the same row and column. The literal savings are not taken into account. Finally, Method *h3* is the one described in Section V. Only examples for which mapping was not trivial are reported. The number of edges of the transition graph for which there was a choice of different next states and the total number of options are reported. For all methods, the table gives the number of literals obtained starting from the exact solution found with the binate covering method.

The effectiveness of shrinking can be evaluated from the following. Synthesis of *green* could not be completed starting from the nonshrunk solution, because of the excessive memory requirements. For the remaining examples whose solutions could be shrunk, the total literal count was reduced from 2175 to 2095 (or 4%).

Preliminary results for the compatible pair method are given in Table V. The two numbers in the rightmost column are the times taken without/with closure check; $*$ means timeout. Results so far have been obtained with the number of states as

[3] Picking always the first choice gave consistently the best results among all the trivial algorithms, including random selection. This is because this method guarantees more uniform selections than, for instance, random choices.

TABLE V
RESULTS WITH THE COMPATIBLE PAIR METHOD

FSM	solution closed?	N_s	time (s)
lbk	YES	16	3.35/21.67
lcf	YES	91	4.47/5.72
scf	YES	97	5.04/6.77
tbk	YES	16	3.3/21.60
bbsse	YES	13	0.18/0.23
beecount	YES	4	0.05/0.07
ex1	YES	18	0.05/0.37
mark1	NO	15	.28/0.33
opus	YES	9	0.01/0.05
green	YES	37	14.35/164.38
jac1	YES	21	1.4/*
lion9	YES	4	0.00/0/00
jac2	NO	30	3.68/*
pager	YES	10	1.01/1.10
ex3	NO	10	3.06/14.97
ex5	YES	3	1.68/5.17
ex7	YES	3	2.48/12.03
train11	YES	4	0.42/0.38
kohavi6	YES	3	0.15/0.12
kohavi7	YES	3	0.13/0.08
kohavi8	NO	6	0.15/0.22
bbara	YES	7	0.15/0.15
bbtas	YES	6	0.03/0.05
sse	YES	13	0.15/0.20
palama	YES	1	0.08/0.05
ex2	YES	5	249.64/*
jac3	YES	19	38672.64/*
jac3	YES	19	(1020.50)/*

TABLE VI
RESULTS OF HEURISTICMINIMIZATION

FSM	N_s	time (s)	literals	heuristics
bbara	7	0.01	45	T
bbsse	13	0.17	103	I
beecount	4	0.01	24	I
ex1	18	0.06	198	T
ex2	5	79.92	27	T
ex3	4	0.39	25	T
ex5	3	0.07	14	I
ex7	3	0.12	20	T
lion9	4	0.00	13	T
mark1	12	0.22	69	I
opus	9	0.00	63	T
scf	97	0.76	754	T
sse	13	0.18	103	I
tbk	16	3.88	210	T
train11	4	0.01	12	T
grasselli	4	0.03	24	T
luccio	2	0.00	6	T
house	9	1.96	38	T
unger65	3	0.01	7	T
pager	10	0.04	48	I
palama	3	0.01	12	T
tma	18	0.06	126	T
lbk	16	4.10	204	T
green	37	644.28	4199	I
jac1	21	0.49	493	I
jac2	14	1.00	470	I
jac3	20	28.42	613	I
jac4	20	24618.85	278	I,T
TOTAL	389	25378.05	8198	

objective. Times refer to SUN Sparcstations 1. The second entry for *jac3* gives the time obtained when each unate covering problem is solved heuristically by stopping the search at the first leaf of the solution tree.

Finally, Table VI collects the results of applying the heuristic techniques described in Section IV. The rightmost column lists which of the techniques have been applied: I means isomorphic state identification and T means tight upper bound. A technique not listed for a particular example did not apply to it.

The only example where the heuristic approach did not deliver the true optimum is *jac3*, where the approximate solution had one more state. On the positive side we can note that:

- The times for the most difficult examples were substantially reduced;
- The isomorphic state heuristic makes it possible to solve, though in an approximate way, an example with over three million maximal compatibles. Such a solution is practically precluded to all methods based on the exhaustive enumeration of all maximal compatibles.

The only previous work reporting results on public domain benchmark FSM's is [25]. Table VII compares results from that work to ours. The literal counts for some of the examples were not reported in [25]. The CPU times of *FSMRED* were divided by 10 to account for the difference of computer speeds. *STAMINA* is the name of our program implementing the binate covering approach. *FSMRED* is faster for example *ex2*, but it obtains a substantially worse result. On most other examples, *STAMINA* is faster even when solving the problem exactly. *STAMINA* produces fewer states and considerably fewer literals. The latter comparison, however, must be taken with a grain of salt, since the encoding programs are different and the *misII* commands used in [25] were not reported.

In [7] the number of states is reported for examples taken from the literature. *STAMINA* obtains the minimum number of states on all those machines, when run in heuristic mode. Run times are negligible for all the examples.

VIII. CONCLUSIONS

This paper has described several efficient algorithms for the minimization of finite state machines. We have considered how to find solutions that result in better encoded machines, rather than just decrease the numbers of states. When an additive cost function such as the number of states is used, one has to select among the possibly many solutions with the same number of states. In that context, we have discussed the mapping and shrinking problems and shown their relevance. We have also considered the implications of cost functions that are not purely additive. One algorithm, based on the coverage of the closed subgraphs of the compatibility graph, has been shown to be remarkably efficient, in spite of its ability to deal with nonadditive cost functions.

Our experiments indicate that most hand-designed finite state machines are amenable to exact minimization. We have also shown that heuristic techniques can be used in other cases. In all but very few cases, state minimization followed by state mapping and solution shrinking provides a better starting point for the subsequent synthesis tasks of encoding,

TABLE VII
COMPARISON TO *FSMRED* [25]

FSMRED					*STAMINA*					
			literals		exact			heuristic		
FSM	N_s	time	Nova	Mustang	N_s	time	literals	N_s	time	literals
bbara	7	0.2	50	62	7	0.00	45	7	0.01	45
beecount	4	0.1	37	62	4	0.01	24	4	0.01	24
ex2	10	0.3	87	130	5	4702.22	27	5	72.92	27
ex5	5	0.1	19	30	3	0.07	14	3	0.07	14
ex7	4	0.1	25	35	3	0.16	20	3	0.12	20
mark1	12	0.1	–	–	12	0.21	–	12	0.22	–
opus	9	0.1	–	–	9	0.00	–	9	0.00	–
scf	97	99.8	–	–	97	0.75	–	97	0.76	–
sse	13	0.1	–	–	13	0.17	–	13	0.18	–
tbk	16	3.5	278	436	16	3.86	210	16	3.88	210
train11	4	0.1	18	32	4	0.01	12	4	0.01	12
TOTAL	181	104.5	514	787	173	4707.5	352	173	78.2	352

logic optimization, and technology mapping, when area is the goal. Further investigation is required to better direct the minimization process to improve the speed of the FSM's and to identify useful nonadditive cost functions. Another area of investigation is the development of algorithms that trade off some optimality for the ability to deal with machines even larger, and with many more compatibles, than those we have used for our experiments. One possible source of similar machines is the collapsing of two or more simpler machines in an attempt to resynthesize a network of FSM's.

REFERENCES

[1] M. C. Paull and S. H. Unger, "Minimizing the number of states in incompletely specified sequential switching functions," *IRE Trans. Elect. Comp.*, vol. EC-8, pp. 356–367, Sept. 1959.

[2] S. Ginsburg, "Synthesis of minimal state machines," *IRE Trans. Elect. Comp.*, vol. EC-8, pp. 441–449, Dec. 1959.

[3] A. Grasselli and F. Luccio, "A method for minimizing the number of internal states in incompletely specified sequential networks," *IEEE Tran. Elect. Comp.*, vol. EC-14, pp. 350–359, June 1965.

[4] Y. V. Pottosin, "Experimental evaluation of one method of minimizing the number of states of discrete automata," in *Synthesis of Digital Automata*, V. G. Lazarev and A. V. Zakrevskii, eds., transl. from Russian. New York: Consultants Bureau, 1969, pp. 92–98.

[5] E. B. Lee and M. Perkowski, "Concurrent minimization and state assignment of finite state machines," in *IEEE Conf. Systems, Man, and Cybernetics*, Halifax, Canada, Oct. 1984, pp. 248–260.

[6] M. J. Avedillo, J. M. Quintana, and J. L. Huertas, "A new method for the state reduction of incompletely specified sequential machines," in *Proc. European Design Automation Conf.*, Glasgow, U.K., Mar. 1990, pp. 552–556.

[7] M. J. Avedillo, J. M. Quintana, and J. L. Huertas, "New approach to the state reduction in incompletely specified sequential machines," in *IEEE Int. Symp. Circuits and Systems*, New Orleans, LA, May 1990, pp. 440–443.

[8] B. Reusch and W. Merzenich, "Minimal coverings for incompletely specified sequential machines," *Acta Informatica*, vol. 22, pp. 663–678, 1986.

[9] Z. Kohavi, *Switching and Finite Automata Theory*, 2nd ed. New York: McGraw-Hill, 1978.

[10] E. J. McCluskey, *Logic Design Principles*. Englewood Cliffs, NJ: Prentice-Hall, 1986.

[11] J.-K. Rho, G. D. Hachtel, and F. Somenzi, "Don't care sequences and the optimization of interacting finite state machines," in *IEEE Int. Conf. Computer-Aided Design*, Santa Clara, CA, Nov. 1991, pp. 418–421.

[12] E. J. McCluskey, Jr., "Minimization of Boolean functions," *Bell Syst. Technical J.*, vol. 35, pp. 1417–1444, Nov. 1956.

[13] S. R. Petrick, "A direct determination of the irredundant forms of a Boolean function from the set of prime implicants," Tech. Rep. AFCRC-TR-56-110, Air Force Cambridge Research Center, Cambridge, MA, Apr. 1956.

[14] A. Grasselli and F. Luccio, "Some covering problems in switching theory," in *Networks and Switching Theory*, G. Biorci, ed. New York: Academic Press, 1968.

[15] R. K. Brayton and F. Somenzi, "An exact minimizer for Boolean relations," in *IEEE Int. Conf. Computer-Aided Design*, Santa Clara, CA, Nov. 1989, pp. 316–319.

[16] 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. Comp.*, vol. C-18, pp. 953–956, Oct. 1969.

[17] R. Tarjan, "Depth first search and linear graph algorithms," *SIAM J. Computing*, vol. 1, pp. 146–160, 1972.

[18] S. H. Unger, *Asynchronous Sequential Switching Circuits*. New York: Wiley, 1969.

[19] M. Perkowski, A. Rydzewski, and P. Misiurewicz, *Teoria Uktadow Logicznych*. Warsaw, Poland: Wydawnictwa Politechniki Warszawskiej, 1978, in Polish.

[20] S. Devadas, H.-K. T. Ma, A. R. Newton, and A. Sangiovanni-Vincentelli, "MUSTANG: State assignment of finite state machines for optimal multi-level logic implementations," *IEEE Trans.Computer-Aided Design*, vol. CAD-7, pp. 1290–1300, Dec. 1988.

[21] B. Lin and A. R. Newton, "Synthesis of multiple level logic from symbolic high-level description languages," in *Proc. IFIP Int. Conf. VLSI*, Aug. 1989, pp. 187–196.

[22] X. Du, G. D. 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.

[23] B. Lin and F. Somenzi, "Minimization of symbolic relations," in *IEEE Int. Conf. Computer-Aided Design*, Santa Clara, CA, Nov. 1990, pp. 88–91.

[24] G. V. Russo and G. Palamà, "Minimization of incompletely specified sequential machines," *Digital Processes*, vol. 6, pp. 199–206, 1980.

[25] L. N. Kannan and D. Sarma, "Fast heuristic algorithms for finite state machine minimization," in *Proc. European Design Automation Conf.*, Amsterdam, The Netherlands, Feb. 1991, pp. 192–196.

[26] S. Yang, "Logic synthesis and optimization benchmarks user guide version 3.0," Tech. rep., Micro-electronics Center of North Carolina, Research Triangle Park, NC, Jan. 1991.

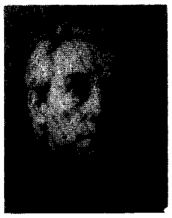
[27] R. K. Brayton, R. Rudell, A. Sangiovanni-Vincentelli, and A. R. Wang, "MIS: A multiple-level interactive logic optimization system," *IEEE Trans. Computer-Aided Design*, vol. CAD-6, pp. 1062–1081, Nov. 1987.

[28] H. Savoj and R. K. Brayton, "The use of observability and external don't cares for the simplification of multi-level networks," in *Proc. Design Automation Conf.*, Orlando, FL, pp. 297–301, June 1990.

[29] E. Detjens, G. Gannot, R. Rudell, A. Sangiovanni-Vincentelli, and A. Wang, "Technology mapping in MIS," in *Proc. IEEE Int. Conf. Computer-Aided Design*, Nov. 1987, pp. 116–119.

June-Kyung Rho received the B. S. degree in electrical engineering from Seoul National University in 1982 and the M. S. degree in electrical engineering from Korea Advanced Institute of Science and Technology (KAIST) in 1985 all in Seoul, Korea. He recived the Ph. D. degree in electrical engineering from the University of Colorado at Boulder in 1993.

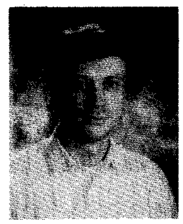
He is currently a Senior Engineer at Gold Star Information and Communication Inc., Seoul, Korea. He worked for Cadence Design Systems, Lowelll, MA, Mitsubishi Electric Reasearch Laboratories, Cambridge, MA, and Synopsys, Mountain View, CA as a summer intern. His research interests include logic synthesis, finite state machine optimization, and formal verification.



Gary D. Hachtel (S'62–M'65–SM'74–F'80) received the B. S. degree from the California Institute of Technology in 1959 and the Ph. D. degree from the University of California, Berkeley, in electrical engineering.

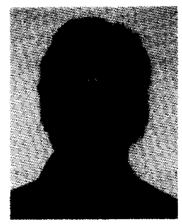
He has taught at U.C. Berkeley, at New York University, at U.C.L.A., where he was Regents Lecturer in 1974, and at the University of Denver (Department of Mathematics). From 1965 to 1981 he was with IBM at the Thomas J. Watson Research Center at Yorktown Heights, NY, where he was manager of Modeling and Systems Design in the Mathematical Sciences Department. Since 1981 he has been Professor of Electrical and Computer Engineering at the University of Colorado (Boulder). Since 1981 he has been a principal investigator on research and equipment grants, which have brought more than $3M to the University of Colorado. He is currently overall grant administrator and coprincipal investigator on a three-year joint $2.3M Boulder/Berkeley/Stanford research grant(Boulder share $.7M), jointly sponsored by NSF and DARPA, for which $2.7M renewal proposal has just been forwarded to NSF from the University of Colorado. His current research is on theory and algorithms for sequential and combinational logic synthesis, simulation, testing, layout, spare matrices, and optimization.

Dr. Hachtel was an Associate Editor for the *International Journal for Numerical Methods in Engineering* and for the IEEE TRANSACTIONS ON CIRCUITS AND SYSTEMS. He is now an Associate Editor for the *International Journal for Mathematics and Computation in Electronics and Electrical Engineering* and for the *Journal of Formal Methods in System Design*. He received an IBM Outstanding Contribution Award for integrated circuit device modeling in 1968 and an IBM Outstanding Invention Award for the Tableau Approach to Circuit Simulation and Design. In 1971 he was corecipient of the best paper award from the Circuits and Systems Society and in 1979 he was corecipient of the W.R.G. Baker award for the best IEEE Proceedings or Transactions article to appear in calendar year 1978. In 1981 he was a distinguished lecturer of the Circuits and Systems Society. In 1989 he was corecipient of an outstanding paper award from the 1989 Hawaii International Conference on System Science, and was awarded the annual Faculty Research award from the College of Engineering at the University of Colorado. In 1991 he was awarded a Fulbright Fellowship, as well as a Faculty Fellowship from the University of Colorado, to study at the Universidad Politecnica de Madrid for the 1991-1992 academic year. He is coholder of U.S. 3,750,409, the second software patent issued. He is a co-organizer of IBM/Boulder/Berkeley/Stanford Summer Logic Synthesis Seminar, held annually since 1980.



Fabio Somenzi (M'88) received the Dr. Eng. degree in Electronic Engineering from Politecnico di Torino, Italy, in 1980.

He was with SGS-Thomson Microelectronics from 1982 to 1989, responsible for computer-aided digital design. From 1984 to 1987 he taught digital logic design at the Computer Science Department of the University of Milano, Italy. In 1987 he visited the Electrical Engineering and Computer Science Department of the University of California, Berkeley. Since 1989, he has been with the Department of Electrical and Computer Engineering of the University of Colorado, Boulder, where he is currently an Associate Professor. His research interests include synthesis, simulation, verification, and testing of logic circuits.



Reily M. Jacoby received the B. S. degree in physics from the University of Hawaii in 1977 and the B. S. E. E. degree from the same institution in 1978. He was awarded the M. S. E. E. degree and the Ph. D. degree in electrical and computer engineering from the University of Colorado, Boulder, in 1986 and 1989, respectively.

He spent five years in industry working for Motorola and GenRad designing and implementing software systems. At GenRad he participated in the design and implementation of a functional level test systems. Since 1989 he has been working for Cadence Design Systems in Lowell, MA. His current research interests involve developing and implementing algorithms for logic verification of combinational logic circuits, automatic test pattern generation, multiple-level logic synthesis, and optimization. He has coauthored seven papers in these areas during the last four years.