Efficient state reduction methods for PLA-based sequential circuits

M.J. Avedillo
J.M. Quintana
J.L. Huertas

Indexing terms: Finite sequential machines, Programmable logic arrays, State minimisation

Abstract: Experiences with heuristics for the state reduction of finite-state machines are presented and two new heuristic algorithms described in detail. Results on machines from the literature and from the MCNC benchmark set are shown. The area of the PLA implemention of the combinational component and the design time are used as figures of merit. The comparison of such parameters, when the state reduction step is included in the design process and when it is not, suggests that fast state-reduction heuristics should be implemented within FSM automatic synthesis systems.

1 Introduction

Nowadays, technological advances lead to more and more sophisticated digital system. It is impracticable to realise them without the help of CAD tools [1–3]. Systems for the design of finite state machines (FSM) have been implemented since the early 1960s but, few of them included state minimisation because the inherent complexity of this process. In particular, it was shown that the reduction of completely specified finite automata can be achieved in $O(n \log n)$ steps [4]. The minimisation of incompletely specified finite automata is a NP-complete problem [5]. Nevertheless, FSM designers have recently pointed out the need for efficient algorithms for the minimisation of large machines [15, 16].

The minimisation of the number of states is an important task in the optimal design of sequential circuits. Reducing the number of states corresponds to decreasing the number of transitions of the sequencing functions and eventually to reducing the number of implicants in a two-level logic realisation or reducing the number of literals in a multilevel one. Moreover, a reduction of the number of states may correspond to a reduction of the number of bits that is needed for the state encoding [6].

Another area where state minimisation applies is the test generation for sequential machines. Many of the reported approximations are ineffective when the number of states in the circuit is large and the test demands long input sequences.

In this paper we deal with the state minimisation problem as a part of an automatic synthesis system. After giving some basic concepts to make the paper self contained, we review existing methods for state minimisation of a FSM. Emphasis is placed on heuristic approaches, establishing a distinction between constructive heuristics and iterative improvement techniques. Two new methods for solving this problem are described, each from one of the two categories above. Experimental results are given and a detailed comparison is carried out mainly in terms of silicon area and computer time.

2 Background

A finite state machine (FSM) can be formally defined as a quintuple $M = (I, S, O, \delta, \lambda)$ where I is a finite set of inputs, S is a finite, nonempty set, of states, O is a finite set of outputs, $\delta: I \times S \rightarrow S$ is the next state function, and $\lambda: I \times S \rightarrow O$ ($\lambda: S \rightarrow O$) is the output function for a Mealy (Moore) machine. The FSM model used is shown in Fig. 1. The combinational component is implemented with programmable logic arrays (PLAs). Fig. 2 depicts a flow graph of the design process of FSMs.

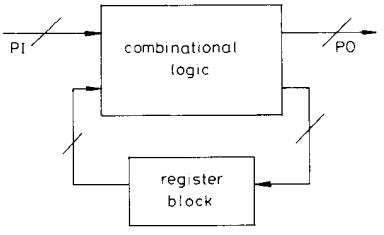

Fig. 1 *FSM model*

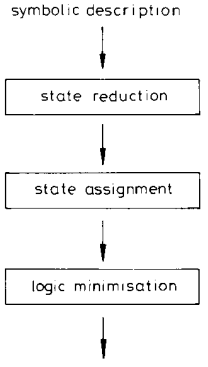

Fig. 2 *Design flow*

Paper 9098E (C2, E10), first received 16th July 1991 and in revised form 18th June 1992

The authors are with the Departamento de Diseño Analógico, Centro Nacional de Microelectrónica, Edif. CICA, Avda. Reina Mercedes s/n, 41012-Sevilla, Spain

Symbolic descriptions consist of a set of symbolic implicants [7]. Each symbolic implicant has four components:

$$i \quad s \quad \delta(i, s) \quad \lambda(i, s)$$

and represents the state transition and outputs produced when input i is applied to the FSM being in state s. The present state s and the next state $\delta(i, s)$ are symbolic representations (labels) which will be coded in the state assignment phase. Input and output functions can be both symbolic, in which case they will also be optimally encoded, or binary-valued due to the constraints imposed by other components of the system being designed. If the next state or outputs are not specified for all (input, present state)-pairs the FSM is said to be incompletely specified.

State tables (tabular symbolic descriptions) are the starting point of classical state reduction theory [8]. The problem can be stated as: 'given a state table, find another one which specifies the same external behaviour with a minimum number of states'.

We briefly review for the sake of completeness some basic concepts in state reduction [8–11]. Internal states s_i and s_j are compatible if:

(*a*) for any input $i_k \in I$ such that both $\lambda(i_k, s_i)$ and $\lambda(i_k, s_j)$ are specified

$$\lambda(i_k, s_i) = \lambda(i_k, s_j)$$

(*b*) for any input $i_k \in I$ such that both $\delta(i_k, s_i)$ and $\delta(i_k, s_j)$ are specified

$$\delta(i_k, s_i) \text{ is compatible to } \delta(i_k, s_j)$$

A compatibility class or compatible C_i is a set of states such that members of the set are pairwise compatible. A compatible which is not a proper subset of any other compatibility class is called a maximal compatible (MC). If there is an input such that s_k is the next state of s_i and s_l is the next state of s_j and $(s_k, s_l) \neq (s_i, s_j)$ then we say that (s_k, s_l) is implied by (s_i, s_j).

The class set P_i implied by the compatible C_i is the set of all compatibles C_{ij} implied by C_i for all inputs i_j, such that

(i) C_{ij} has more than one element
(ii) $C_{ij} \not\subset C_i$
(iii) $C_{ij} \not\subset C_{ik}$ if $C_{ik} \in P_i$

The closure class set E_i of a compatible C_i is a set of all compatibles implied by C_i obtained by repeated use of transitivity of implication, such that the compatibles which are subsets of either C_i or any other member of E_i are removed from the set.

A compatible C_i dominates a compatible C_j if $C_i \supset C_j$ and $P_i \subseteq P_j$. A compatible that is not dominated by any other compatible is called a prime compatibility class (PC).

A collection of compatibles covers all states of a state table if every state is contained in at least one compatible in the collection.

A collection of compatibles is closed if for each compatibility class in it, $C_i = \{s_{i1}, s_{i2}, \ldots, s_{im}\}$ and any input i_k, the set of states $\{\delta(i_k, s_{i1}), \delta(i_k, s_{i2}), \ldots, \delta(i_k, s_{im})\}$ is contained in at least one compatible of the collection.

3 Exact minimisation approaches

A fundamental theorem [10] states that, given an incomplete state table, another state table specifying the same external behaviour corresponds to each closed set of compatibility classes which covers all internal states of the given table. Classical approaches to the state reduction problem are structured in two well differentiated steps:

(*a*) generation of the complete set of some kind of compatibility classes (prime compatibles [9], prime closed sets [12], etc.)

(*b*) extraction of a minimum closed cover.

The allowed compatibility classes are those for which there is assurance a minimum cardinality closed cover exits which is composed uniquely of them. There are drawbacks related to both steps. Concerning the first one, the number of those compatibility entities might be too large precluding an efficient generation. The second step implies the solution of a minimum closed covering problem, which is known to belong to the class of NP-complete problems for incompletely specified finite state machines. Then, the very nature of this problem has precluded the inclusion of a state minimisation step in many FSM design systems during the 1970s and early 1980s.

However, advances both in computing power and heuristic algorithms have convinced researchers that revising state minimisation concepts is worthwhile [13–16]. But nowadays FSM designers are no more interested in symbolic descriptions with a minimum number of states; instead they focus the FSM synthesis process as a global optimisation task aimed at obtaining minimal cost implementations both in terms of silicon area and design time. It is well known that a symbolic description with a minimal number of states is not necessarily the appropriate starting point when minimal area implementations are looked for; however, usually a reduction in the number of states leads to a reduction in the complexity of the resulting FSM. Then, there is a need for developing efficient algorithms which are able to produce FSMs optimised in terms of silicon area, speed, testability, etc.

4 Heuristic minimisation approaches: previous work

There are several styles of heuristic strategies for solving combinatorial optimisation problems. We classify these strategies into two categories: constructive approaches, and iterative improvement methods. In constructive heuristics, a good solution to the problem being solved is built up piece by piece. This kind of approach to the state-reduction problem follows the structure of classical methods but, instead of attempting to determine minimum solutions, near-minimum (minimal) ones are heuristically selected through

(*a*) generation of a set (usually, a complete set) of some kind of compatibles

(*b*) heuristic selection of minimal closed cover.

Heuristic algorithms were developed as early as 1972 by Bennets [18]. In this pioneering work it was pointed out that the type of initial compatibles on which they are based is not so critical. Bennets proposed a reduction algorithm based on the maximal compatible sets, MCs. Using this subset of the prime compatible set reduces (in some cases drastically) the number of candidate compatibles. The procedure consists of selecting one of the essential (or quasi-essential) MCs and attempting to satisfy its closure requirements (generating one of the smallest set of MCs that satisfies the violated closure requirements for the MC selected). The result will be a closed set of MCs that may or may not provide full covering on the

initial set of states. The procedure is repeated until a full cover is achieved.

Very recently, several heuristic algorithms for FSM minimisation have been proposed. Kannan and Sarma [15] begin by generating the set of all the maximal compatibles. Then a minimal set of maximal compatibles which covers all the states is built up. Finally, if the FSM is incompletely specified and the minimal cover is not the complete set of maximal compatibles, this minimal cover is expanded to obtain an optimal closed cover. Two different heuristic algorithms are proposed for the last step. In the 'large set' approach, each compatible in the minimal cover is checked for closure and appropriate compatibles are added to fulfill closure requirements. At every step, extra states are tried to be added to compatibles in the cover. New sets are created only if absolutely necessary. In the 'lean set' approach, all the states that occur more than once in the set of maximal compatibles in the minimal cover are removed. Then closure requirements are satisfied adding states to the existing compatibles or adding new compatibles.

Hachtel *et al.* [16] stated that exact state minimisation is feasible for a large class of practical examples but have also pointed out the interest of heuristic techniques for the state minimisation of FSM generated by sequential synthesis systems. Two heuristic approaches are presented to solve the problem. In both of them they attempt to reduce the time and memory requirements by generating only a subset of the prime class set for the subsequent closed covering problem. First, a closed cover composed of maximal compatibles uniquely is found. Then, only the prime classes contained in the maximal compatibles of the previous solution are computed and used to formulate a binate covering problem [9, 10] which is then solved exactly. There are two different heuristics reported to find a closed cover of maximal compatibles. One of them is suitable for machines with a high number of maximal compatibles and is based on the concept of isomorphic states so that not all the maximal compatibles are taken into account in the process of building up a closed cover. The other one is for machines with a large number of prime classes. In this case, the complete set of maximal compatibles is generated and a minimum closed cover is found.

Although these recent references appear as very interesting, the field is worth exploring further.

In iterative improvement strategies an existing solution is perturbed in the direction of a lower cost one. These strategies have been only applied to the final steps of some methods and thus they may be considered as refinements [13, 16] of solutions obtained by constructive approaches. But in our opinion, iterative improvement strategies have not been explored in depth. There seems to be a parallelism between the history of logic minimisation and that of state reduction. Approaches to logic minimisation exist which simultaneously identify and select implicants for a cover. This is the case of the well known logic minimiser Espresso [19]. No similar approach exists for state reduction. So an interesting objective is developing state-reduction algorithms which simultaneously identify and select compatibles for the closed cover, and comparing them with existing approaches.

5 A new constructive heuristic

We have developed a state-reduction algorithm, Reduces [20], using a constructive heuristic based on maximal compatibles. The state reduction is achieved by processing the state table T, from which the set of maximal compatibles $CCSS$ is obtained. The global process is described using Pidgin-C in Fig. 3. Then, a table Q with the covering and closure constraints is derived by **Get-constraints( )** in Fig. 3. Covering and closure constraints

```
REDUCES(T)
/* state reduction algorithm based on maximal compatibles
(MCs) */
{
CCSS = Get_MC(T);
Q = Get_constraints(CCSS, T);
CC = {∅};
while(CC ! = closed cover)
{  if(CC ! = closed)
   CC = +Select_CC(violated closure constraints);
   else
   {CC = +Select_CC(violated covering constraints);
   }
}
T' = Get_Table(CC);
}
Select_CC(violated type constraints)
{  Cand = Most_restrictive(violated type constraints);
   for (each CC in Cand)
   Compute_C();
   return (M_i maximises C);
}
```

Fig. 3 *Pidgin-C description of algorithm Reduces*

can be formulated as an integer linear programming problem [10]. The closed cover (CC) is initialised to the empty set. Set CC is built up by adding MCs one by one until no closure or covering constraint is violated. To do this, if CC is not closed, procedure **Select_CC( )** is called with the set of violated closure constraints as an argument, else it is called with the set of violated covering constraints. This procedure returns a maximal compatible M_i satisfying at least one of the most restrictive constraints (constraints satisfied by the smallest number of MCs) in the passed set. Procedure **Most-restrictive( )** evaluates the most restrictive constraints and those MCs satisfying them are stored in the set *Cand*. For each MC in this set, **Compute_C( )** calculates the number of violated constraints satisfied by selecting M_i (PC, positive contribution), and the number of them that are violated as a consequence of adding M_i to CC (NC, negative contribution). Finally, the MC M_i which maximises C, a weighted sum of NC and PC, is returned so ending the call to procedure **Select_CC( )**. If both closure and covering constraints are violated, we attempt to satisfy closure requirements first. These actions are repeated until CC is a closed cover for table T. Then, the reduced state table T' is derived by **Get_Table( )**. An important point in the proposed heuristic is the selection of weights in the definition of C. From our experience we conclude that an effective guidelines is $C = PC - 2NC$.

Note that we do not need to generate a minimum set of MCs satisfying a given subset of the constraints, this preventing us to handle a NP-complete problem.

We use the FSM in Table 1 from Reference 21 as an example to clarify how the algorithm works. The set of maximal compatibles ($CCSS$) for this machine is shown in Fig. 4.

5.1 Example

We associate a 0–1 integer variable c_i with each maximal compatible M_i so that the corresponding integer program is formulated, $c_i = 1$ if M_i is selected as a

member of a closed cover set and $c_i = 0$ if M_i is not selected. These covering and closure constraints are shown in Fig. 5.

Initially, $CC = \varnothing$ and the procedure **Select_CC()** is called with the set of covering constraints (1–9 in Fig. 5) as argument. Expr. 4 in Fig. 5 is the most restrictive violated one, as it is satisfied only by the selection of M_1 or M_4 (these MCs are the only ones that cover state s_4).

MCs for Table 1

$M_1 = (s_4, s_5, s_7, s_8, s_9)$ $M_6 = (s_2, s_3, s_6, s_9)$
$M_2 = (s_2, s_5, s_7, s_8, s_9)$ $M_7 = (s_1, s_2, s_5, s_8)$
$M_3 = (s_2, s_3, s_7, s_8, s_9)$ $M_8 = (s_1, s_2, s_3, s_8)$
$M_4 = (s_4, s_5, s_6, s_9)$ $M_9 = (s_1, s_2, s_5, s_6)$
$M_5 = (s_2, s_5, s_6, s_9)$ $M_{10} = (s_1, s_2, s_3, s_6)$

Fig. 4 *Maximal compatibles for FSM in Table 1*

Table 1: FSM example [21] to clarify how new algorithm works

A	B	C	D
s_2, —	s_4, —	—	s_3, —
s_6, —	s_9, —	—	—
—	—	s_7, —	s_8, —
s_2, —	s_1, —	s_6, —	s_5, —
—	—	—	s_6, —
s_1, 0	—	s_2, —	—, 1
s_5, 1	s_2, —	—	—
s_5, —	—	—	s_1, 0
s_5, —	s_3, —	—	—

$$-1 + c_7 + c_8 + c_9 + c_{10} \geq 0 \quad (1)$$
$$-1 + c_2 + c_3 + c_5 + c_6 + c_7 + c_8 + c_9 + c_{10} \geq 0 \quad (2)$$
$$-1 + c_3 + c_6 + c_8 + c_{10} \geq 0 \quad (3)$$
$$-1 + c_1 + c_4 \geq 0 \quad (4)$$
$$-1 + c_1 + c_2 + c_4 + c_5 + c_7 + c_9 \geq 0 \quad (5)$$
$$-1 + c_4 + c_5 + c_6 + c_9 + c_{10} \geq 0 \quad (6)$$
$$-1 + c_1 + c_2 + c_3 \geq 0 \quad (7)$$
$$-1 + c_1 + c_2 + c_3 + c_7 + c_8 \geq 0 \quad (8)$$
$$-1 + c_1 + c_2 + c_3 + c_4 + c_5 + c_6 \geq 0 \quad (9)$$
$$-c_1 + c_2 + c_5 + c_7 + c_9 \geq 0 \quad (10)$$
$$-c_1 + c_8 + c_{10} \geq 0 \quad (11)$$
$$-c_1 + c_9 \geq 0 \quad (12)$$
$$-c_2 + c_4 + c_5 + c_9 \geq 0 \quad (13)$$
$$-c_2 + c_3 + c_6 \geq 0 \quad (14)$$
$$-c_2 + c_9 + c_{10} \geq 0 \quad (15)$$
$$-c_3 + c_4 + c_5 + c_9 \geq 0 \quad (16)$$
$$-c_3 + c_7 + c_8 \geq 0 \quad (17)$$
$$-c_4 + c_7 + c_9 \geq 0 \quad (18)$$
$$-c_4 + c_8 + c_{10} \geq 0 \quad (19)$$
$$-c_4 + c_5 + c_6 + c_9 + c_{10} \geq 0 \quad (20)$$
$$-c_5 + c_9 \geq 0 \quad (21)$$
$$-c_5 + c_3 + c_6 \geq 0 \quad (22)$$
$$-c_6 + c_9 \geq 0 \quad (23)$$
$$-c_6 + c_2 + c_3 \geq 0 \quad (24)$$
$$-c_7 + c_5 + c_9 \geq 0 \quad (25)$$
$$-c_7 + c_1 + c_4 \geq 0 \quad (26)$$
$$-c_7 + c_{10} \geq 0 \quad (27)$$
$$-c_8 + c_5 + c_9 \geq 0 \quad (28)$$
$$-c_8 + c_1 + c_4 \geq 0 \quad (29)$$
$$-c_9 + c_1 + c_4 \geq 0 \quad (30)$$
$$-c_9 + c_6 + c_{10} \geq 0 \quad (31)$$
$$-c_{10} + c_1 + c_4 \geq 0 \quad (32)$$
$$-c_{10} + c_2 + c_3 \geq 0 \quad (33)$$
$$-c_{10} + c_3 + c_8 \geq 0 \quad (34)$$

Fig. 5 *Covering and closure constraints for FSM in Table 1*

It is clear that M_1 and/or M_4 will be members of any closed cover set, including that of minimum cardinality. Then, $Cand = \{M_1, M_4\}$, and C is computed for both MCs:

▶ for M_1

$PC(M_1) = 5$ because $M_1 = (s_4, s_5, s_7, s_8, s_9)$ covers five uncovered states of the original table. That is, the selection of this MC satisfies five violated inequalities: expr. 4, 5, 7, 8 and 9. $NC(M_1) = 3$ because the consequence of selecting M_1 is the violation of inequalities 10, 11, 12 which express the closure requirements of M_1. That is, if M_1 is selected as a member of a closed cover, MCs containing compatibles (s_1, s_2, s_3), (s_2, s_5) and (s_1, s_5, s_6) will have to be included too.

$$C(M_1) = PC(M_1) - 2NC(M_1) = -1$$

▶ for M_4

$$PC(M_4) = 4 \quad NC(M_4) = 3$$
$$C(M_4) = PC(M_4) - 2NC(M_4) = -2$$

M_1 is selected ($CC = \{M_1\}$) because it maximises the parameter C. Now there are some closure constraints violated (expr. 10, 11, 12) and the algorithm tries to satisfy them firstly, so procedure **Select_CC()** is called for these constraints. Among this set, expr. 12 is the most restrictive one and $Cand = \{M_9\}$. In this case, as there is only one candidate MC, we do not need to compute C. M_9 becomes a member of CC.

Now, the set of violated closure constraints is {(11), (31)} and

$$Cand = \{M_6, M_8, M_{10}\}$$
$$C(M_6) = 0 \quad C(M_8) = 2 \quad C(M_{10}) = -1$$

M_8 is selected. The only inequality not satisfied is expr. 31 and the algorithm evaluates C for each of the MCs whose selection satisfies that constraint, i.e.:

$$Cand = \{M_6, M_{10}\}$$
$$C(M_6) = -1 \quad C(M_{10}) = -1$$

Arbitrarily, M_6 is selected. As a consequence of adding this MC to CC, expr. 24 is violated:

$$Cand = \{M_2, M_3\}$$
$$C(M_2) = 1 \quad C(M_3) = 1$$

M_2 is added to CC ($CC = \{M_1, M_9, M_8, M_6, M_2\}$) which now is a closed cover of the original table describing the FSM.

6 Iterative improvement heuristic

6.1 Motivations

The strategy we propose herein is similar to that in Espresso [19] for the heuristic minimisation of combinational functions. That is, a set of basic operations is defined which transforms a symbolic description of the FSM in another one with a smaller number of states. Our algorithm Arnes [22, 23] may be described, from a high level point of view, as a sequence of transformations (functions) which, starting with the initial description of the FSM, results in a sequence of intermediate descriptions of such a machine with a decreasing number of states. The process finishes when the application of the functions implemented by the algorithm does no longer reduce the number of states. The main advantage of this

approach is the avoidance of generating any complete set of compatibles, as in classical state reduction algorithms.

6.2 The new algorithm

Primary objective in Arnes is reducing the number of states in the symbolic description of a FSM being used as an input to other phases of the design process of sequential circuits. Moreover, once it obtains a solution with a reduced number of states, Arnes operates on it to maximise the number of 'don't cares' in excitation functions. This aims at simplifying the task of state assignment programs and logic minimisers [13, 16], so better solutions are obtained.

In Fig. 6 the algorithm control block is described using Pidgin-C, where T is a symbolic description of a FSM using state tables, C is a closed cover for table T, that is, a closed set of compatibles which covers all internal states in T, and Φ is the number of compatibles in C (number of internal states in the description of the FSM). Procedure **init(*T*)** initialises C to the set of internal states in the initial description T which is a closed cover of the FSM. Function **cost(C)** evaluates the cost of a solution, C. There are two main functions in Arnes: **expand** and **reduction** which we describe in detail. Once C has been transformed by the application of basic previous functions, procedure **table(C, T)** builds a symbolic description T' for the FSM defining an internal state for each compatible in C. If the cost of this new description is smaller than that of the initial one, the whole process is repeated for T'.

```
ARNES(T)
{  C = init(T);
   Φ* = φ = cost(C);
   (Φ, C) = expand(C, T);
   (Φ, C) = reduction(C, T);
   T' = table(C, T);
   if(Φ < Φ*)
   {  ARNES(T');
   }
}
```

Fig. 6 *Pidgin-C description of control block of the algorithm*

```
expand(C, T)
/*given a closed cover of T, returns a closed cover composed by
  larger compatibles (eventually prime compatibles) without
  having generated them previously*/
{  Φ* = Φ = cost(C);
   for (i = 1; i ≤ |C|; i++)
   {  expand1(C_i);
      Φ = cost(C);
   }
   if (Φ < Φ*)
   {expand(C, T);
   }
}
```

Fig. 7 *Pidgin-C description of procedure* ***expand***

Now, we explain the main procedures of Arnes. Procedure **expand** adds states to each compatible C_i in C, includes those compatibles needed to fulfill closure requirements of C_i^+ (expanded compatible) and eliminates those in C that are now covered. Given C_i to be expanded, an optimum expanded compatible C_i^{opt} is defined such that

$$\alpha = \text{card}\,(W(C_i^{opt})) - \text{card}\,(DIMP(C_i^{opt})) \text{ is maximum} \quad (1)$$

$$W(C_i^{opt}) = W_1(C_i^{opt}) \cup W_2(C_i^{opt}) \quad (2)$$

where $DIMP(C_i^{opt})$ is the closure set E_i^{opt} of C_i^{opt} eliminating those compatibles which are members of C, $W_1\{C_i^{opt}\}$ is the set of compatibles which can be eliminated from C, when substituting C_i by C_i^{opt} because they are included in C_i^{opt}, and $W_2(C_i^{opt})$ is the set of compatibles which can be eliminated from C, when substituting C_i by C_i^{opt} because they are included in one compatible in $DIMP(C_i^{opt})$. If all compatibles $C \supset C_i$ could be enumerated, we would select one satisfying eqn. 1. Instead, since this operation is computationally too expensive, the basic operation **expand1** obtains C_i^+ that is a good approximation to eqn. 1. In Fig. 7 and 8 procedures **expand** and **expand1** are described using Pidgin-C.

Procedure **expand1** begins by computing the set of states that are compatible to all states in C_i and storing them in set *comp*. Then, for each potential expanded compatible $C_i^{+j} = C_i \cup s_j$, where s_j is a state in set *comp*, α is evaluated. One of the expanded compatibles which maximises α is selected to be C_i and procedure **expand1** is repeated for this new C_i. The expansion of one compatible finishes when there are not compatible states to the current expanded compatible or α is negative for all potential expanded compatibles. After expanding a compatible, C is still a closed cover for T. Moreover, expand-

```
expand1(C_i)
/*given C_i, compatible in C, returns C_i^+, which is a good approximation to (1)*/
{  flag = 1;
   while(flag==1)
   {  comp = compatibles(C_i);   /*stores in 'comp' those states which are compatible to all states in C_i*/
      if(comp != {∅})
      {  for(each s_j ∈ comp)
            C_i^{+j} = C_i ∪ s_j;
            select C_i^{+j} which maximises α, and store in C_i^+;
            C = (C ∪ C_i^+ ∪ DIMP(C_i^+)) − C_i − W(C_i^+);
         /*C_i^+ is a compatible which contains C_i*/
         /*DIMP(C_i^+) is the closure set E_i^+ of C_i^+ minus those compatibles in C*/
         /*W is the set of compatibles in C contained in C_i^+ or in any compatible in DIMP*/
         if(α < 0 for all C_i^{+j})
            flag = 0;   /*C, expansion is not continued because expanding means increasing the number of compatibles in C*/
      }
      else
         flag = 0; /*C, expansion is not continued because there are not compatible states to C_i*/
   }
}
```

Fig. 8 *Pidgin-C description of procedure* ***expand1***

ing is only allowed if it does not mean an increment in the cardinality of $\boldsymbol{C}$.

Procedure **reduction** transforms $\boldsymbol{C}$ in a new $\boldsymbol{C}^*$ where each C_i is sequentially substituted by a reduced compatible $C_i^- \subseteq C_i$, such that

$$\{\boldsymbol{C} - C_i\} \cup C_i^- \text{ is a closed cover}$$

For reducing each C_i, first the set of states in C_i covered by unless another compatible in $\boldsymbol{C}$, H, is derived. All states in H could be removed from C_i and $\boldsymbol{C}$ still would be a cover of T but may be not a closed set of compatibles. Procedure **MCEE** selects the largest subset of H that can be eliminated from C_i, such that $\boldsymbol{C}$ is still a closed cover of T.

Procedure **reduction** allows to move away from one solution to another of less cost, as the application of the whole procedure to state table T', might lead to less costly solutions. Moreover, **reduction** eliminates states that are covered by more than one compatible. The minimisation of such number of states leads to the maximisation of 'don't cares' in excitation functions [13]. In Fig. 9 we describe the procedure. Fig. 10 contains a description of the basic function in **reduction** (procedure **MCEE**). **MCEE** complexity increases exponentially with the number of states in the set H. From our experience we conclude this is not a problem, even for large machines.

We use again the machine described in Table 1 to show how the algorithm works. As seen from the foregoing description there are three main blocks in the algorithm: initialisation, expansion and reduction.

6.3 Example

Initialisation: We initialise $\boldsymbol{C}$ to the set of internal states in the original state table and evaluate the cost of such solution:

$$\boldsymbol{C} = \{(s_1), (s_2), (s_3), (s_4), (s_5), (s_6), (s_7), (s_8), (s_9)\}$$

$$\Phi = \Phi^* = 9$$

Expansion: Procedure **expand1** is called for the first compatible in $\boldsymbol{C}$. That is, $C_1 = (s_1)$. States which are compatible to all states in C_1 are stored in *comp*:

$$comp = \{s_2, s_3, s_5, s_6, s_8\}$$

α is evaluated for each $C_1^{+j} = C_1 \cup s_j$, where s_j is an element from *comp*.

▶ for s_2:

$$C_1^{+2} = (s_1, s_2)$$

$$DIMP(C_1^{+2}) = \{(s_2 s_6), (s_4 s_9), (s_1 s_6), (s_1 s_3), (s_2 s_5), (s_3 s_8), (s_1 s_8)\}$$

$$W_1(C_1^{+2}) = \{(s_2)\}$$

It can be eliminated from $\boldsymbol{C}$ because it is covered by C_1^{+2};

$$W_2(C_1^{+2}) = \{(s_6), (s_4), (s_9), (s_3), (s_5), (s_8)\}$$

They can be eliminated from $\boldsymbol{C}$ because they are covered by compatibles in *DIMP*.

$$\text{card}(W(C_1^{+2})) = 7$$

$$\text{card}(DIMP(C_1^{+2})) = 7$$

$$\alpha = 7 - 7 = 0$$

This means, expanding s_1 with s_2 leads to a closed cover with the same number of compatibles.

```
reduction(C, T)
/*returns C* where each compatible C_i is substituted sequentially by C_i^-, minimum compatible which when substituting C_i, verifies that
  C* is still a closed cover of T*/
{
   for(i = 1; i < |C|; i++)
   {  H = {∅};
      for(j = 1; j < |C|; j++)
      {  if(i != j)
         {H = H ∪ (C_j ∩ C_i);
         }
      }
      /*H is the set of states in C_i covered by at least another compatible in C*/
      if(H != ∅)
      {  C_i^- = C_i - MCEE(H); /*largest set of states that may be eliminated*/
         C = (C - C_i) ∪ C_i^-;
      }
   }
}
```

Fig. 9 *Pidgin-C description of procedure* ***reduction***

```
MCEE(H)
/*given H, a set of states in C_i covered by at least another compatible in C, returns the largest subset of H that can be eliminated from C_i
  such that (C - (C_i)) ∪ (C_i - MCEE(H)) is still a closed cover of T*/
{  if(all states in H can be eliminated)
   {  return (H);
   }
   else
   {  s_i = select-state(H);   /*selects the first state in H still not selected*/
      H^- = MCEE(H_i^-);       /*tries a solution without eliminating s_i from C_i*/
      H^+ = MCEE(H_i^+);       /*tries a solution eliminating s_i from C_i*/
      H' = merge(H^-, H^+);    /*selects the set with the largest cardinality*/
      return (H');
   }
}
```

Fig. 10 *Pidgin-C description of procedure* ***MCEE***

Summarising the results of the evaluation of α for each state s_j from *comp:*

$C_1^{+j}\alpha$

$(s_2 s_2)$ 0	$(s_1 s_3)$ 1	$(s_1 s_5)$ 1
$(s_1 s_6)$ 0	$(s_1 s_8)$ 1	

s_3 is selected to expand C_1. $\boldsymbol{C}$ is updated. New closed cover and cost are

$\boldsymbol{C} = \{(s_1 s_3), (s_3 s_8), (s_1 s_8), (s_2 s_5), (s_4), (s_6), (s_7), (s_9)\}$

$\Phi = 8$

New compatible C_1 in $\boldsymbol{C}$ is expanded

$C_1 = \{(s_1 s_3)\}$

$comp = \{s_2, s_6, s_8\}$

Again, α is evaluated for each C_1^{+j}.

$C_1^{+j}\alpha$

$(s_1 s_3 s_2)$ 0	$(s_1 s_3 s_6) < 0$	$(s_1 s_3 s_8)$ 2

s_8 is selected to expand $(s_1 s_3)$. New closed cover and cost are

$\boldsymbol{C} = \{(s_1 s_3 s_8), (s_2 s_5), (s_4), (s_6), (s_7), (s_9)\}$

$\Phi = 6$

We expand compatible C_1:

$C_1 = \{(s_1 s_3 s_8)\}$ $comp = \{s_2\}$

$C_1^{+j}\alpha$

$(s_1 s_3 s_8 s_2)$ 1

As α is positive, state s_2 is added to C_1. New closed cover and cost are

$C = \{(s_1 s_2 s_3 s_8), (s_2 s_5 s_6), (s_1 s_6), (s_4 s_9), (s_7)\}$

$\Phi = 5$

There are not compatible states to $C_1 = \{(s_1 s_2 s_3 s_8)\}$, and so the expansion of this compatible is not continued. Now the algorithm tries to expand compatible C_2 in $\boldsymbol{C}$.

$C_2 = \{(s_2 s_5 s_6)\}$ $comp = \{s_1, s_9\}$

$C_2^{+j}\alpha$

$(s_2 s_5 s_6 s_1) < 0$	$(s_2 s_5 s_6 s_9) < 0$

The expansion of C_2 is not continued because α is negative for all states in *comp*. The number of states in closed cover $\boldsymbol{C}$ would increase if we used s_1 or s_9 to expand $C_2 = \{(s_2 s_5 s_6)\}$.

The algorithm goes on with C_3

$C_3 = \{(s_1, s_6)\}$ $comp = \{s_2, s_3, s_5\}$

$C_3^{+j}\alpha$

$(s_1 s_6 s_2) < 0$	$(s_1 s_6 s_3) < 0$	$(s_1 s_6 s_5) < 0$

α is negative for each C_3^+ and C_3 is not expanded.

For $C_4 = \{(s_4 s_9)\}$ $comp = \{s_5, s_6, s_7, s_8\}$

$C_4^{+j}\alpha$

$(s_4 s_9 s_5) < 0$	$(s_4 s_9 s_6) < 0$
$(s_4 s_9 s_7)$ 1	$(s_4 s_9 s_8) < 0$

$\boldsymbol{C}$ is updated by expanding C_4 with s_7.

$\boldsymbol{C} = \{(s_1 s_2 s_3 s_8), (s_2 s_5 s_6), (s_1 s_6), (s_4 s_7 s_9)\}$

$\Phi = 4$

Now $C_4 = \{(s_4 s_7 s_9)\}$ $comp = \{s_5, s_8\}$

$C_4^{+j}\alpha$

$(s_4 s_7 s_9 s_5) < 0$	$(s_4 s_7 s_9 s_8) < 0$

C_4 is not expanded because α is negative for all potential expanded compatible C_4^{+j}.

Reduction: Procedure **reduction** sequentially removes from the compatibles in $\boldsymbol{C}$, those states which are not necessary to fulfill covering constraints nor closure ones. In this example no state is eliminated.

As $\Phi = 4$ is less than $\Phi^* = 9$ the whole procedure is repeated for the new state table derived from the final closed cover. No improvement is achieved in this case.

7 Experimental results

Experiments have been carried out on a large set of FSMs to test the algorithms previously described. Both of them have been coded in *C*. The program implementing the constructive one is called Reduces and the iterative improvement one, Arnes. Results both for machines from the literature and for machines in the MCNC benchmark set [24] are shown.

7.1 Figures of merit

We focus on three figures of merit: the final number of states in the FSM representation, the area occupied by the combinational component in a PLA-based implementation of the FSM, and the time which is required for design. We introduce them in more detail:

(*a*) the number of states in the symbolic description after state reduction. This is an important parameter to evaluate how well the state reduction algorithms work. It is the main concern of all the algorithms described in classic papers on the subject. If state assignment programs which use minimum length codes are employed, the number of states determines the number of memory elements.

(*b*) the area of the combinational component. We use a PLA to implement the combinational component of the FSM. The area in this case is

$$\text{area} = K_p(2n_i + 3n_v + n_o) \tag{3}$$

where n_i is the number of input variables, n_v is the number of state variables, and n_o is the number of output variables.

(*c*) design time. This parameter reports the time needed for passing from the initial symbolic description to the minimised Boolean representation of next state and output functions. It is composed of three partial times:

t_1 time for state reduction phase (if implemented)

t_2 time for state assignment phase

t_3 time for final logic minimisation

time $= t_1 + t_2 + t_3$

7.2 Results on machines from the literature

We have tried many examples of machines from switching circuit textbooks and journal papers about minimisation [26]. In many of them the gains were significant. Table 2 shows the number of states (*ns*), the number of product terms (*tp*) and the size of the PLAs implementing the combinational part of the machines after state assignment and logic minimisation for four different initial descriptions: state tables as they appear in the literature, state tables reduced by the application of an implementation of a classic state reduction algorithm (Bennets algorithm [18]), state tables reduced by the application of Reduces, and state tables obtained by Arnes. This parameter *size* has been obtained with a $K_p = 1$ in eqn. 3. Reported times in the last three include the time invested in the state reduction step. In all cases, the optimal state assignment algorithm we used was the 'i-hybrid complement' algorithm in Nova [25].

From Table 2 Arnes obtains minimum cardinality solutions in 14 of 15 cases, including the well known (and difficult) 22-state machine (FSM6) in Reference 17. Also, from this table, it should be clear that the size of the final circuit when Arnes is applied is lower than the size when starting with original FSMs except for FSM10 (equal size). In ten machines Arnes gave less area realisations than solutions supplied by the well-known Bennets algorithm. Only for FSM1 and FSM10 did the implementation of Bennets algorithm achieve better size results than Arnes. Summarising the comparison between Reduces and Arnes, in nine machines Arnes gave area savings while Reduces obtained better results for three machines.

For each of the machines in Table 2 two ratios have been calculated

$$A = \frac{\text{size}_A}{\text{size}_O} \quad \text{and} \quad T = \frac{\text{time}_A}{\text{time}_O} \tag{4}$$

This is the area occupied by the PLA, obtained using Arnes, divided by the area resulting when the original description is used for state assignment. In the same way, the design time with state reduction using Arnes is divided by the design time without state reduction step.

We represent the pair (A, T) for each machine in a design space whose Y-axis represents the time ratio and the X-axis the area ratio. There are two significant lines in this space, defined as $A = 1$ and $T = 1$, respectively. For any machine FSM_i with coordinates (A_i, T_i) below the line defined by $T = 1$, the design time decreases when state reduction is included in the synthesis process. For any machine FSM_i with coordinates (A_i, T_i) on the left of the line defined by $A = 1$, the state-reduction step is advantageous in terms of silicon area.

Advantages in both area and time correspond with machines inside the region limited by those lines. Also its borders represent favourable situations. That is, they represent cases where state reduction achieves a decreasing in one of the design coordinates without changing the other one. In Fig. 11, pairs (A, T) for the machines in Table 2 have been drawn. The point labelled [FSM] (0.45, 0.22) represents the average for the machines in Table 2.

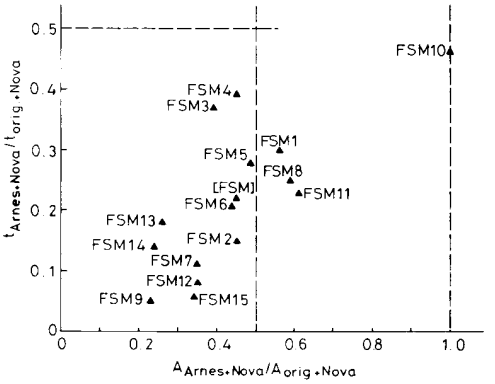

Fig. 11 *Experimental results when Arnes is applied to literature benchmark*

7.3 Results on machines in MCNC benchmark

Both Reduces and Arnes programs have been applied to the machines in the MCNC benchmark. Table 3 summarises the results after state assignment for those machines for which we found descriptions with a reduced number of states. We do not include the machines which have identical primary outputs for any present state and input conditions. Time for heuristic state reduction is

Table 2: Minimisation results for machines from literature

			Original FSM				Bennets FSM				Reduces FSM				Arnes FSM				ns_{min}
	n_i	n_o	*ns*	*tp*	$size_o$	$time_o$	ns_B	*tp*	*size*	*time*	ns_R	*tp*	*size*	*time*	ns_A	*tp*	$size_A$	$time_A$	
FSM1	2	1	5	7	98	2.7	4	4	44	1.4	4	4	44	1.2	3	5	55	0.8	3
FSM2	2	1	9	10	170	12.1	6	10	140	3.7	5	10	140	3.1	4	7	77	1.8	4
FSM3	2	1	7	9	126	4.8	4	7	77	1.5	4	7	77	1.2	4	5	55	1.8	4
FSM4	2	1	6	7	98	3.1	4	4	44	1.0	3	4	44	0.8	3	4	44	1.2	3
FSM5	2	1	6	8	112	4.3	4	6	66	1.2	4	6	66	1.2	3	5	55	1.2	3
FSM6	2	1	22	25	500	261.2	12	16	272	180.3	11	12	204	23.9	9	13	231	55.9	9
FSM7	2	1	9	14	238	16.5	5	10	140	2.6	6	10	140	2.7	4	8	88	1.8	4
FSM8	2	1	6	8	112	5.0	4	6	66	1.4	4	6	66	1.3	3	6	66	1.2	3
FSM9	3	1	9	15	285	35.2	4	9	117	3.0	4	11	143	2.7	3	5	65	1.7	3
FSM10	3	1	8	14	224	16.4	5	13	208	4.8	5	13	208	4.7	5	14	224	7.5	4
FSM11	2	1	6	9	126	6.1	5	9	126	3.5	5	9	126	3.2	4	7	77	1.5	4
FSM12	2	1	9	11	187	15.3	4	7	77	1.7	4	7	77	1.4	3	6	66	1.3	3
FSM13	3	1	6	11	176	6.6	3	5	65	1.3	3	5	65	1.4	3	4	52	1.2	3
FSM14	2	1	6	7	98	2.8	2	3	24	0.6	2	3	24	0.6	2	3	24	0.4	2
FSM15	3	1	10	10	190	23.6	4	7	91	1.9	4	6	78	1.9	4	5	65	1.5	4

n_i, number of input variables; n_o, number of output functions; *ns*, ns_B, ns_R, ns_A, states in descriptions obtained from papers, Bennets algorithm, Reduces and Arnes, respectively; ns_{min}, minimum number of states; *tp*, number of product terms; *size*, PLA size evaluated by Nova; *time*, time in seconds on a Sun 3/260 workstation

Table 3: Minimisation results for machines in MCNC benchmark

	Original FSM						Reduces FSM					Arnes FSM				
	n_i	n_o	ns	tp	$size_o$	$time_0$	ns'_R	tp	$size$	$time$		ns'_A	tp	$size_A$	$time_A$	
										t_1	t_2+t_3				t_1	t_2+t_3
bbara	4	2	10	25	550	17.1	7	20	380	0.8	8.1	7	20	380	0.2	8.8
bbsse	7	7	16	29	957	47.0	13	29	957	4.1	435.2	13	29	957	3.5	39.8
beecount	3	4	7	10	190	7.8	4	9	144	0.3	3.4	4	9	144	0.2	3.7
ex1	9	19	20	50	2600	350.3	18	43	2236	32.3	281.5	18	43	2236	24.3	281.5
ex2	2	2	19	36	756	95.0	10	19	342	3.6	13.8	7	13	195	4.3	6.9
ex3	2	2	10	18	324	14.5	6	10	150	0.9	3.5	5	9	135	0.9	5.1
ex5	2	2	9	18	324	11.6	4	8	96	0.8	2.3	3	6	72	0.3	2.3
ex7	2	2	10	17	306	17.3	4	7	84	0.9	2.3	4	9	108	0.4	3.1
lion9	2	1	9	9	153	8.7	4	7	77	0.5	2.4	4	7	77	0.1	2.6
mark1	5	16	15	17	646	51.5	12	17	646	4.1	225.6	12	16	608	4.2	44.1
opus	5	6	10	16	448	11.6	9	16	448	1.1	10.0	9	16	448	0.4	9.8
scf	27	56	121	146	19126	9676.6	97	128	16768	137	7234.0	97	128	16768	112	7234.0
sse, bbsse	7	7	16	29	957	49.4	13	29	957	5.0	459.2	13	29	957	3.5	40.0
tbk	6	3	32	170	5100	7840.0	16	52	1404	39.5	1426.9	16	52	1404	30.8	1425.3
train11	2	1	11	12	204	14.5	4	6	66	0.5	2.6	4	6	66	0.1	3.4

shown in the column labelled t_1 for each of our algorithms. The columns labelled $t_2 + t_3$ give the time invested in state assignment and logic minimisation with the '*i*-greedy complement' option in Nova.

From the Table for almost half of the machines area savings of more than 50% are achieved when the reduced machine obtained with Arnes or Reduces is used as input for the state assignment phase. The time to obtain such solutions with Arnes ($t_1 + t_2 + t_3$ in $time_A$ column) is less than that resulting from applying Nova to the original machine ($time_o$ column) for all of them. Concerning the state assignment and minimisation steps, there are three machines (bbsse, sse, mark1) for which the Nova time when using Reduces ($t_2 + t_3$ in Reduces column) is higher than the Nova time when the original description is used. It is due to the symbolic descriptions generated by the state minimisers which are then used as input to the state assignment phase. In some cases, they are descriptions with a large number of symbolic implicants and the logic minimisation step slows down. However, this is a feature of the algorithm's implementation which can be corrected. In Fig. 12, pairs (A, T) for the machines in Table 3 have been drawn.

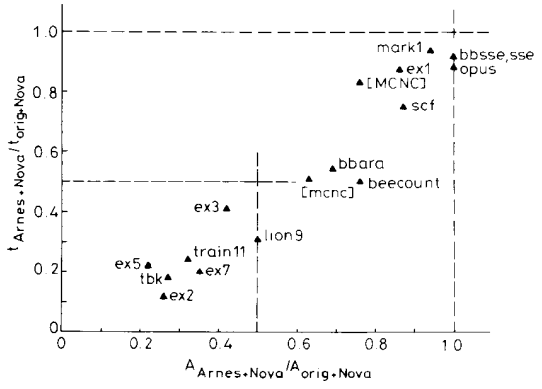

Fig. 12 *Experimental results when Arnes is applied to MCNC benchmark*

Nevertheless, the discussion of results would be incomplete if we did not consider what happens with machines for which state reduction does not achieve any reduction. When provided with a description of a machine to be implemented, we do not know beforehand whether there are compatible states or not. This means the state reducer will be invoked and a certain amount of time invested in checking if there are compatible states. If so, a description with less states will be tried, but the state reducer can fail in finding such one. Thus, the result of this processing being the same initial description and coordinates for this machine ($A = 1$, $T > 1$). It is difficult to carry out an estimation of how this increasing in the computer time may affect. To give a rough idea, we have evaluated coordinates (A, T) for the whole MCNC set (about the half of the machines do not have compatible states), the average result leading to the point (0.76, 0.83), point [MCNC] in the Figure, which is inside the striped region in Fig. 12. Of course, the average value for those MCNC machines admitting minimisation is much better (point [mcnc] with coordinates (0.63, 0.53) in the Figure).

From the set of machines from the literature, the superiority of Arnes against Reduces in area is evident. They are similar in time. The superior performance of Arnes is not so clear for the machines in the MCNC benchmark. The key point to understand the differences is whether the selection of maximal compatibles as variables to formulate the LP integer problem (Reduces) implies a limitation to the quality of the solutions it can obtain. This is the case for the literature machines but, obviously, it is not for the MCNC benchmark.

Since there have been two other approaches recently [15, 16], it is convenient to compare our results with them. From the published results, we have elaborated Table 4 where the minimisation achieved on MCNC

Table 4: Comparison of minimisation results achieved on MCNC machines

	Kannan *et al.* [15]	Hachtel *et al.* [16]	Reduces	Arnes
	*ns**	*ns**	*ns**	*ns**
bbara	7	7	7	7
bbsse	—	13	13	13
beecount	4	4	4	4
ex1	—	18	18	18
ex2	10	5	10	7
ex3	—	4	6	5
ex5	4	3	4	3
ex7	4	3	4	4
lion9	—	4	4	4
mark1	12	12	12	12
opus	9	9	9	9
scf	97	97	97	97
sse	13	13	13	13
tbk	16	16	16	16
train11	4	4	4	4

*ns**, number of states in reduced descriptions

machines is depicted. Concerning the state reduction itself, that is comparing the number of states in the reduced descriptions (ns^*) obtained with each approach, Arnes gave results similar to those reported in References 15 and 16. However, we have not been able to compare them in terms of silicon area because we do not have enough detail from the papers. Comparing computer time is not significant because the machines used have very different power.

8 Conclusions

The experiments carried out suggest that fast state reduction heuristics should be included within FSM automatic synthesis systems to achieve both area and time savings.

Probably new improvements of all of these methods will come about in the future, but they are promising in terms of augmenting the present capabilities of design automation for FSM. In that sense, Arnes opens the door to a new way of solution compared with traditional approaches like Reduces or other, more recently reported, contributions.

9 References

1 BROWN, D.W.: 'A state-machine synthesizer'. Proceedings of the 18th conference on *Design automation*, 1981, pp. 301–305

2 SAUCIER, G., CRASTES DE PAULET, M., and SICARD, P.: 'ASYL: A rule-based system for controller synthesis', *IEEE Trans.*, 1987, **CAD-6**, (6), pp. 1088–1097

3 AMANN, R., NEHER, M., RIETSCHE, G., and ROSENSTIEL, W.: 'CASTOR: FSM synthesis in a digital synthesis system'. Proceedings of European conference on *Solid-state circuits*, 1989, pp. 105–108

4 HOPCROFT, J.: 'An $n \log n$ algorithm for minimizing states in a finite automaton', *in* KOHAVI, Z. (Ed.): 'Theory of machines and computation' (Academic Press 1971), pp. 189–196

5 PFLEEGER, C.: 'State reduction of incompletely specified finite state machines', *IEEE Trans.*, 1973, **C-26**, pp. 1099–1102

6 DE MICHELI, G.: 'Synthesis of control systems', *in* DE MICHELI, G., SANGIOVANNI-VINCENTELLI, A., and ANTOGNETTI, P. (Eds.): 'Design systems for VLSI circuits' (Martinus Nijhoff, 1987), pp. 327–364

7 DE MICHELI, G.: 'Computer-aided synthesis of PLA-based systems'. PhD dissertation, Berkeley University, USA, 1984

8 PAUL, M.C., and UNGER, S.H.: 'Minimizing the number of states in incompletely specified sequential switching functions', *IRE Trans.* 1959, **EC-8**, pp. 356–367

9 GRASSELLI, A., and LUCCIO, F.: 'A method for minimizing the number of internal states in incompletely specified sequential networks', *IEEE Trans.*, 1965, **EC-14**, pp. 350–359

10 GRASSELLI, A., and LUCCIO, F.: 'Some covering problem in switching theory', *in* 'Network and switching theory' (Academic Press, 1968)

11 FRIEDMAN, A.D., and MENON, R.: 'Theory and design of switching circuits' (Computer Science Press, 1975)

12 DE SARKAR, S.C., BASU, A.K., and CHOUDHURY, A.K.: 'Simplification of incompletely specified flow tables with the help of prime closed sets', *IEEE Trans.*, 1969, **C-18**, pp. 953–956

13 PERKOWSKI, M.A., and NGUYEN, N.: 'Minimization of finite state machines in SuperPeg'. Proceedings of the Midwest conference on *Circuits and systems*, Louisville, KY, USA, 1985, pp. 139–147

14 PERKOWSKI, M.A., and LIU, J.: 'Generation and optimization of finite state machines from parallel program graphs'. DIADES Research Group Report 10/89, 1989

15 KANNAN, L.N., and SARMA, D.: 'Fast heuristic algorithms for finite state machine minimization'. Proceedings of EDAC'91, Amsterdam, 1991, pp. 192–196

16 HACHTEL, G.D., RHO, J.K., SOMENZI, F., and JACOBY, R.: 'Exact and heuristic algorithms for the minimization of incompletely specified state machines'. Proceedings of EDAC'91, Amsterdam, 1991, pp. 184–191

17 HOUSE, S.: 'A new rule for reducing CC tables', *IEEE Trans.*, 1970, **C-19**, pp. 1108–1110

18 BENNETS, R.G., WASHINGTON, J.L., and LEWIN, D.W.: 'A computer algorithm for state table reductions', *Radio & Electron. Eng.*, 1972, **42**, pp. 513–520

19 BRAYTON, R.K., HATCHEL, G.D., McMULLEN, C., and SANGIOVANNI, A.L.: 'Logic minimization algorithms for VLSI synthesis' (Kluwer, Hingham, MA, USA, 1984)

20 AVEDILLO, M.J., QUINTANA, J.M., and HUERTAS, J.L.: 'A new method for the state reduction of incompletely specified finite sequential machines'. Proceedings of EDAC'90, Glasgow, 1990, pp. 552–556

21 UNGER, S.H.: 'Asynchronous sequential switching circuits' (Wiley-Interscience, New York, 1969)

22 AVEDILLO, M.J., QUINTANA, J.M., and HUERTAS, J.L.: 'New approach to the state reduction in incompletely specified sequential machines'. Proceedings of IEEE international symposium on *Circuits and systems*, New Orleans, LA, USA, 1990, pp. 440–443

23 AVEDILLO, M.J., QUINTANA, J.M., and HUERTAS, J.L.: 'State reduction of incompletely specified finite sequential machines' *in* MICHEL, P., and SAUCIER, G. (Eds.): 'Proceedings of the IFIP Working Conference on Logic and Architecture Synthesis' (Elsevier Science Publishers, 1991)

24 LISANKE, R.: 'Introduction of synthesis benchmark'. Presented at international workshop on *Logic synthesis*, North Carolina, USA, 1989

25 VILLA, T., and SANGIOVANNI-VINCENTELLI, A.: 'NOVA: state assignment of finite state machines for optimal two-level logic implementation', *IEEE Trans.*, 1990, **CAD-9**, (9), pp. 905–924

26 REUSCH, B., and MERZENICH, W.: 'Minimal coverings for incompletely specified sequential machines', *Acta Inform.*, 1986, (22), pp. 663–678