

Exact Minimization of Binary Decision Diagrams Using Implicit Techniques

Arlindo L. Oliveira, *Member, IEEE*, Luca P. Carloni, *Member, IEEE*,
Tiziano Villa, and Alberto L. Sangiovanni-Vincentelli, *Fellow, IEEE*

Abstract—This paper addresses the problem of binary decision diagram (BDD) minimization in the presence of don't care sets. Specifically, given an incompletely specified function g and a fixed ordering of the variables, we propose an exact algorithm for selecting f such that f is a cover for g and the binary decision diagram for f is of minimum size. The approach described is the only known exact algorithm for this problem not based on the enumeration of the assignments to the points in the don't care set. We show also that our problem is NP-complete. We show that the BDD minimization problem can be formulated as a binate covering problem and solved using implicit enumeration techniques. In particular, we show that the minimum-sized binary decision diagram compatible with the specification can be found by solving a problem that is very similar to the problem of reducing incompletely specified finite state machines. We report experiments of an implicit implementation of our algorithm, by means of which a class of interesting examples was solved exactly. We compare it with existing heuristic algorithms to measure the quality of the latter.

Index Terms—Binary decision diagrams, incompletely specified functions, minimization of logic functions, incompletely specified finite state machines, state minimization, implicit logic computations.

1 INTRODUCTION

A completely specified Boolean function f is a cover for an incompletely specified function g if the value of f agrees with the value of g for all the points in the input space where g is specified. This paper describes an exact algorithm for selecting f such that f is a cover for g and the binary decision diagram (BDD) for f has a minimum number of nodes (complemented edges are not considered here). For a given ordering of the variables, the BDD for f is unique [4] and the problem has a well defined solution.

We show that this minimization problem can be solved by selecting a minimum sized cover for a graph that satisfies some additional closure conditions. In particular, we show that the minimum sized binary decision diagram compatible with the specification can be found by solving a covering problem that is very similar to the covering problem obtained using exact algorithms for the reduction of incompletely specified finite state machines (ISFSM) [8]. This similarity makes it possible to use implicit enumeration techniques developed for the purpose of ISFSM reduction [10] to efficiently solve the BDD minimization problem. The manipulation of the characteristic functions of the sets of compatibles and prime compatibles, represented with ROBDDs [2], allows the generation of very large sets that cannot be enumerated explicitly, as it is demonstrated in the experiments.

The transformation presented in this paper and the algorithms developed for the solution are important for practical and theoretical reasons.

From a practical point of view, there are applications in learning and logic synthesis where a high-quality solution is of paramount importance. This requires an exact algorithm to find those solutions or at least to validate the quality of heuristic algorithms.

For instance, in inductive learning applications, the accuracy of the inferred hypotheses is strongly dependent on the complexity of the result [1]. One possible and very effective representation scheme for inferred hypotheses is BDDs. However, it was observed [14] that, when BDDs are used as the representation scheme, existing heuristic algorithms for BDD minimization find solutions that are so far from the minimum that it makes them of little value for this particular application.

The selection of the minimum BDD consistent with an incompletely specified function is also important in logic synthesis applications that use BDDs not only as a tool for representing discrete functions but also to derive implementations that minimize some cost function. For instance, *timed Shannon circuits* [12] use the structure of the BDD to derive low power implementations and stand to gain from algorithms for the reduction of BDDs. The same holds for DCVS trees and multiplexer-based FPGAs.

An exact algorithm, even though unable to solve large instances, helps to measure the quality of heuristic algorithms by gauging them on instances where an exact solution can be found.

From a theoretical point of view, the transformation presented in this work is an elegant characterization of the problem. We show in Section 3 that the problem is NP-complete, answering a question raised by Shiple et al. [17].

- A.L. Oliveira is with Cadence European Labs/IST-INESC, R. Alves Redol 9, 1000 Lisbon, Portugal. E-mail: aml@inesc.pt.
- L.P. Carloni and A.L. Sangiovanni-Vincentelli are with the Department of Electrical Engineering and Computer Science, University of California at Berkeley, Berkeley, CA 94720.
E-mail: {lcarloni, alberto}@eecs.berkeley.edu.
- T. Villa is with PARADES, Via di S. Pantaleo, 66, 00186 Roma, Italy.
E-mail: villa@parades.rm.cnr.it.

Manuscript received 29 July 1996; revised 2 July 1998.
For information on obtaining reprints of this article, please send e-mail to: ic@computer.org, and reference IEEECS Log Number 102067.

Several heuristic algorithms for the problem addressed here have been proposed. These algorithms are important in applications where the available degrees of freedom in the functions represented can be used to reduce the memory requirements of BDD based algorithms. The restrict operator [7] and the constrain operator [6] (also known as generalized cofactor [19]) are two heuristics used to assign the don't cares of a BDD. A comprehensive study of heuristic BDD minimization has been presented in [17]. Another heuristic algorithm has been reported in [5].

We are also aware of work for an exact algorithm [15] based on the enumeration of the different covers that can be obtained by all possible assignments of the don't care points. A pruning technique reduces the enumeration process thanks to a result by Shiple that changing the value of a function f of n variables on a minterm m cannot change the size of the BDD for f by more than n nodes. The pruning is performed implicitly.

It is worth pointing out that reducing the BDD size by assigning the don't care points of an incompletely specified function for a fixed ordering is a priori an orthogonal step to the problem of finding the best variable ordering. They are both formidable problems and, at present, we know of no approaches, either heuristic or exact, that combine these two types of optimization or try to assess their relative impact and correlation. That objective, although important, is outside the scope of this work, where a fixed ordering of the variables is assumed. It is also outside the scope of this paper to compare the BDD size with other measures of complexity of the implementation of an incompletely specified function.

The remainder of this paper is organized as follows: Section 2 introduces basic definitions on BDDs and Section 3 has a proof that BDD minimization is NP-complete. Sections 4, 5, and 6 describe, respectively, the compatibility graph, closed clique covers, and the generation of a minimum BDD. Minimization of BDDs is formulated as a variant of FSM minimization in Section 7, while an implicit algorithm to compute a minimum closed cover is presented in Section 8. Results and conclusions are offered, respectively, in Sections 9 and 10.

2 PRELIMINARIES

A BDD is a rooted, directed, acyclic graph where each node is labeled with the name of one variable. and every non-terminal node n_i has one *else* and one *then* edge that point to the children nodes, n_i^{else} and n_i^{then} , respectively. The terminal nodes are n_z and n_o . By convention, we will draw the *else* (zero) edge as the edge pointing to left (west) and the *then* (one) edge as the edge pointing to right (east).

We define a minterm as any specific combination of input values. We will use m_i to denote the value that the i th variable takes in minterm m . Any minterm m in the input space induces a unique path in a BDD defined in the following way: Start at the root and take, at each node, the *else* or the *then* edge according to the value assigned by minterm m to the variable that is the label of the current node until a terminal node is reached. A BDD corresponds to the completely specified Boolean function f that has all the

minterms in f_{on} (and only these) inducing paths in the BDD that terminate in n_o . A BDD is called *reduced* if no two nodes exist that branch exactly in the same way, and it is never the case that all outgoing edges of a given node terminate in the same node. For a fixed ordering of the variables, the reduced ordered BDD for a given Boolean function is unique. This implies that reduced ordered BDDs are canonical representations of Boolean functions and we will, therefore, use the notation n_i to denote both the node in the BDD and the Boolean function to which it corresponds. Unless stated otherwise, we will use simply the term BDD when we refer to a reduced ordered BDD.

The level of a node n_i , $\mathcal{L}(n_i)$ is the index of the variable tested at that node under the specific ordering used. The level of the terminal nodes is defined as $N + 1$, where N is the number of input variables. The maximum level of a set s of nodes, $\mathcal{L}_{\text{max}}(s)$, is the maximum level of all the nodes in s . Since BDDs are canonical representations for Boolean functions, given a completely specified function h , the level of h , $\mathcal{L}(h)$ is the level of the node that represents h . This is true even if h is a function not represented in a specific BDD under consideration. A BDD is called *complete* if all edges starting at level i terminate in a node at level $i + 1$.¹ The level of a function h , $\mathcal{L}(h)$, is defined as the level of a BDD node that implements h . If n_i is a node in the BDD and m a minterm, $n_i(m)$ will be used to denote both the value of function n_i for minterm m and the terminal node that m reaches when starting at n_i . This notation is consistent because the two terminal nodes stand for the constant functions 0 and 1. The index 0 will be reserved for the root of the BDD. Therefore, if m is a minterm and F is the BDD for f , $n_0(m)$ represents the value of f for minterm m .

A three Terminal BDD (3TBDD) is defined in the same way as a BDD in all respects except that it has three terminal nodes: n_z , n_o , and n_x . A 3TBDD F corresponds to the incompletely specified function f that has all minterms in f_{off} , f_{dc} and f_{on} terminate in n_z , n_x , and n_o , respectively.

A finite state machine is incompletely specified if the destination or the output of some transition is not specified. State minimization of an ISFSM is the problem of finding an FSM with a minimum number of states that, for any input sequence, agrees with the original one on every completely specified output, i.e., for every input sequence produces output sequences compatible with those of the original one. Two states are compatible if the output sequences of the FSM initialized in the two states are compatible for every input sequence. A set of states is a compatible (or are compatible) if, for each input sequence, there is a corresponding output sequence which can be produced by each state in the compatible.

A compatible cover of an FSM is a set of compatibles such that every state of the FSM is in at least a compatible. The implied set of a compatible under an input is the set of states reachable in one transition from the states of the compatible under the input. A set of compatibles is a closed cover if it is a cover such that each implied set of every compatible is contained by a compatible in the cover. A compatible dominates another compatible if it contains it and its implied sets are the same as those of the dominated

1. A complete BDD will not, in general, be reduced.

compatible. Compatibles that are not dominated by any other compatible are called prime compatibles and are sufficient to minimize the states of an FSM. Compatibles that are not properly contained in any other compatible are called maximal compatibles and they are always prime compatibles.

The standard procedure for FSM state minimization is to compute all prime compatibles and, then, find a closed cover of them by solving a binate covering problem, i.e., a matrix where rows encode covering and closure conditions that must be satisfied by a minimum number of compatibles represented as columns intersecting the rows in which the compatibles occur. We refer to [11] for a complete treatment of FSM state minimization.

3 COMPLEXITY OF THE PROBLEM

Consider the problem of minimum BDD identification.

PROBLEM: MINIMUM BDD IDENTIFICATION (MBI)

INSTANCE: A set of minterms, labeled either positive or negative and an integer K .

QUESTION: For a given fixed ordering, is there a BDD with less than K nodes that satisfies all the examples, i.e., a BDD for a function whose on-set contains the minterms labeled positive and whose off-set contains the minterms labeled negative?

Takenaga and Yajima [18] proved that this problem is NP-complete, by reduction from graph K -colorability. The problem we address in this paper is the following:

PROBLEM: EXACT BDD MINIMIZATION (EBM)

INSTANCE: BDDs for functions f_{on} and f_{dc} and an integer K .

QUESTION: Is there a BDD with less than K nodes that implements a function that is a cover for f ?

PROOF THAT IT IS IN NP (due to Shiple [17]). Guess a BDD with fewer than K nodes. Check whether the guessed BDD implements a function that is a cover of f . This check can be done in time and space upper bounded by the product of the sizes of the BDDs for f_{on} and f_{dc} and of the guessed BDD. This product is polynomial in the input size. \square

PROOF THAT IT IS NP-HARD. Suppose we could solve this problem in polynomial time with a deterministic algorithm. Then, we can also solve the MBI problem. To prove the result, we need to prove two facts.

FACT 1. *The BDD for a given function f of v variables cannot have more than $n \times v$ internal nodes, where n is the number of minterms in f [15].*

PROOF OF FACT 1. To verify this, consider all the paths through the BDD defined by all the minterms in f . This set of paths has to go through each internal node in the BDD for f at least once. Otherwise, there are nodes other than the constant node 0 in the BDD that are only reached by minterms in the *off* set of f , thereby implying that the BDD is not reduced. Because a minterm can only traverse v internal nodes, we obtain immediately the above result. \square

FACT 2. *A BDD of a function $f: B^v \rightarrow B$ represented by n minterms can be constructed in $O(n^2 v^2 \log n)$ operations.*

PROOF OF FACT 2. Build the BDD of f from the minterms by doing the following: First, OR together each pair of minterms. Then, OR the results together and keep doing this until the final result is computed. The number of such iterations k is logarithmic in n . At iteration i , one needs to perform no more than $n/(2^i)$ operations on BDDs no larger than $v \times 2^{i-1}$ (the latter size is explained by Fact 1 that the BDD for f cannot have more than $n \times v$ internal nodes). Therefore, per iteration one needs no more than $n/(2^i)v2^{i-1}v2^{i-1} = nv^2 2^{i-2}$ elementary operation, that is upper bounded by $n^2 v^2$ because $i \leq \log n$. Since there are only $\log n$ iterations, the result can be built in time and space $n^2 v^2 \log n$. \square

It follows from Fact 2 that the BDDs for f_{on} and f_{off} can be constructed in time polynomial in the size of the input instance of the MBI problem we want to solve. This implies that the BDD for f_{dc} can also be constructed in polynomial time in the size of the input of the MBI problem because it can be obtained by polynomial time bounded BDD operations. To solve an MBI problem, simply transform it into an EBM problem and solve it. The resulting solution will represent directly the answer to the original problem. \square

After we reported this result [13], we were informed that an earlier proof had been published in a technical report by Sauerhoff and Wegener [16]. Our result has been obtained independently and provides a different proof. In [16], it is also proven that, under the hypothesis that $NP \neq P$, the problem has neither approximation schemes nor polynomial time approximation algorithms yielding solutions larger than the minimum by only a constant factor or a slowly increasing function. Finally, Hirata et al. proved in [9] the related result that MBI is NP-hard (differently from [18]) and that there is a constant $\epsilon > 0$ such that no polynomial algorithm can approximate MBI within the ratio n^ϵ unless $P = NP$.

Incidentally, the polynomial time procedure outlined in the proof of Fact 2, together with the results in [16] directly imply the result of Hirata et al., as the authors themselves point out in the concluding remarks of [9].

4 THE COMPATIBILITY GRAPH

Previous algorithms [15] for this problem used directly the BDD representation of f_{on} and f_{off} . The exact approach described in this paper uses the 3TBDD F that corresponds to the incompletely specified function f . F is assumed to be complete. If necessary, F is made complete by adding extra nodes that have the *then* and *else* edges pointing to the same node. In general, the resulting 3TBDD is no longer reduced. Moreover, we suppose that the 3TBDD does not use complemented edges.

DEFINITION 4.1. Two nodes n_i and n_j in F are compatible ($n_i \sim n_j$) iff no minterm m exists that satisfies $n_i(m) = n_z \wedge n_j(m) = n_o$ or $n_i(m) = n_o \wedge n_j(m) = n_z$.

This definition implies that n_o and n_z are not compatible between them and that n_x is compatible with any node in a 3TBDD.

DEFINITION 4.2. Two nodes n_i and n_j in F are common support compatible ($n_i \approx n_j$) iff there exists a **completely specified** function h such that $h \sim n_i$ and $h \sim n_j$ and $\mathcal{L}(h) \geq \max(\mathcal{L}(n_i), \mathcal{L}(n_j))$.

The definition implies that $n_z \neq n_o$ and $n_x \approx n_i$ for any node n_i .

It is important, at this point, to understand the relationship between these two concepts. First, note that the completely specified function h referred in Definition 4.2 does not necessarily correspond to any node in F . In fact, in most cases, h will not correspond to any node in F , since most nodes in F correspond to incompletely specified functions.

The relationship between compatibility and common support compatibility (CSC) is given by the following lemma:

LEMMA 4.1. If $n_i \approx n_j$, then $n_i \sim n_j$.

PROOF. If $n_i \not\sim n_j$, then there is a minterm m such that $n_i(m) \neq n_j(m)$. Any completely specified function h will assign a unique value to m and, therefore, cannot be compatible with both n_i and n_j . \square

The reverse implication of Lemma 4.1 is not true, in general. Fig. 1 illustrates a situation where two nodes are compatible but are not CSC. Nodes n_i and n_j are compatible because no minterm leads to n_o for one of these nodes and to n_z to the other. However, n_i and n_j are not common support compatible because no completely specified function h that only depends on the second variable is compatible with both of them.

However, when two nodes belong to the same level, common support compatibility and compatibility are equivalent:

LEMMA 4.2. If $\mathcal{L}(n_i) = \mathcal{L}(n_j)$, then $n_i \sim n_j \Rightarrow n_i \approx n_j$.

PROOF. The completely specified function h required to satisfy Definition 4.2 can be obtained by setting $h_{on} = n_{jon} \cup n_{ion}$ and $h_{off} = \overline{h_{on}}$. Since h depends only on the variables common to the supports of n_i and n_j , it can be used in Definition 4.2 to show that $n_i \approx n_j$. \square

The motivation for the definition of common support compatibility can now be made clear. Assume that two nodes belong to different levels and are compatible. In principle, they could be replaced by a new node that implements a function compatible with the functions of each node. In general, this function may depend on variables that are not on the support of the node at the higher level. Assume this node is n_j . Later, when we try to build the reduced BDD, edges that are incident into n_j will need to go upward, against the variable ordering of the BDD. On the other hand, if both nodes are common support compatible,

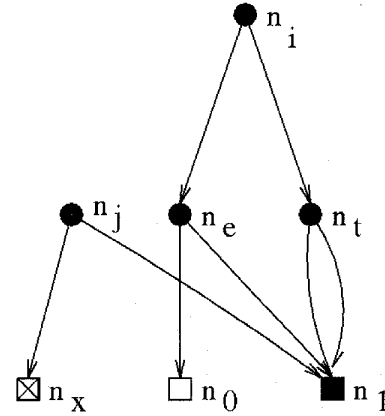


Fig. 1. Nodes n_i and n_j are compatible but not common support compatible.

then they can be replaced by a node that implements the completely specified function h referred to in Definition 4.2. Because this function only depends on the variables common to the supports of both nodes, this problem will not arise.

The concept of common-support compatibility can be extended to sets of nodes in the natural way:

DEFINITION 4.3. The nodes in the set $s_i = \{n_1, n_2, \dots, n_s\}$ are common support compatible iff there exists a completely specified function h such that $(h \sim n_j)_{j=1, \dots, s}$ and $\mathcal{L}(h) \geq \mathcal{L}_{\max}(s_i)$.

DEFINITION 4.4. A set of nodes that are common support compatible is called a **compatible set** or, simply, a **compatible**.

The definition of a compatible implies that any two nodes that belong to a compatible are pairwise common support compatible. The reverse implication is not true, but the next lemma holds.

LEMMA 4.3. Let s_i be a set of nodes belonging to the same level. Then, s_i is a compatible iff all nodes in s_i are pairwise common support compatible.

PROOF. To prove the *if* direction, note that, by the hypothesis, it can never happen that given a minterm m there are two nodes n_j and n_k in s_i that are CSC and satisfy $n_j(m) = n_z$ and $n_k(m) = n_o$. This would violate Lemma 4.1. The function h that is needed to prove that s_i is a compatible set is the function h that has the value 0 for m if some n_j exists that satisfies $n_j(m) = n_z$ and has the value 1 for m if some n_k exists that satisfies $n_k(m) = n_o$ (the value for minterms not defined in this way can be chosen arbitrarily). Moreover, $\mathcal{L}(h) = \mathcal{L}_{\max}(s_i)$, since all nodes of s_i are at the same level as h . If one does not assume that all nodes of s_i belong to the same level, it is not guaranteed that h satisfies $\mathcal{L}(h) \geq \mathcal{L}_{\max}(s_i)$, and one can build counterexamples to the lemma.

To prove the *only if* direction, note that if s_i is a compatible, then the function h referred to in Definition 4.3 is compatible with any pair of nodes in s_i , thereby showing that they are pairwise CSC. \square

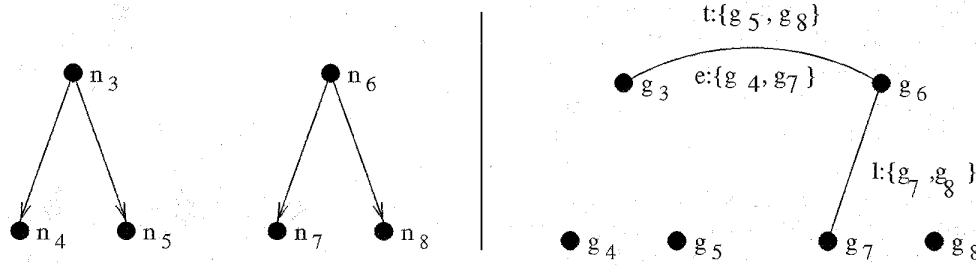


Fig. 2. Example of compatibility graph with labels. For clarity, only labels involving the edges (g_3, g_6) and (g_6, g_7) are shown.

DEFINITION 4.5. The **compatibility graph**, $G = (V, E)$, is an undirected graph that contains the information about which nodes in F can be merged. Except for the terminal node n_x , each node in F will correspond to one node in V with the same index. The level of a node in G is the same as the level of the corresponding node in F . Similarly, g_i^{else} and g_i^{then} are the nodes that correspond to n_i^{else} and n_i^{then} .

Graph G is built in such a way that if nodes n_i and n_j are common support compatible, then there exists an edge between g_i and g_j . An edge may have labels. A label is a set of nodes that expresses the following requirement: If nodes g_i and g_j are to be merged, then the nodes in the label also need to be merged. There are different types of labels that come from different requirements. A label of type t (e) represents the requirement that the *then* (*else*) nodes also have to be equivalent. For example, in Fig. 2, the t label in the edge between g_3 and g_6 means that nodes g_3 and g_6 can only be merged if nodes g_5 and g_8 are also merged. This labeling of the edges in the compatibility graph is inspired by the standard procedure for finite state machine reduction. A label of type l specifies a related requirement, but it involves nodes at different levels in the decision diagram. Fig. 2 also shows an example of an l label that, in this case, specifies that nodes g_6 and g_7 can only be merged if nodes g_7 and g_8 are also merged. The need for labeling of the nodes is given by the following two lemmas, that also justify the algorithm by which graph G is built:

LEMMA 4.4. If $\mathcal{L}(n_i) = \mathcal{L}(n_j)$, then

$$n_i \approx n_j \Rightarrow (n_i^{else} \approx n_j^{else} \wedge n_i^{then} \approx n_j^{then}).$$

PROOF. By contradiction. Since F is complete, the successors are at the same level. Therefore, by Lemma 4.2, $n_i^{else} \neq n_j^{else} \Rightarrow n_i^{else} \not\approx n_j^{else}$ and a minterm m can be selected in such a way that $n_i^{else}(m) \neq n_j^{else}(m)$ and $m_{\mathcal{L}(n_i)} = 0$. The existence of this minterm shows that $n_i \not\approx n_j$ and, therefore, that $n_i \neq n_j$. A similar argument is true for the *then* branch. Therefore,

$$n_i^{else} \neq n_j^{else} \vee n_i^{then} \neq n_j^{then} \Rightarrow n_i \neq n_j.$$

LEMMA 4.5. If $\mathcal{L}(n_i) < \mathcal{L}(n_j)$, then

$$n_i \approx n_j \Rightarrow (n_i^{else} \approx n_j \wedge n_i^{then} \approx n_j \wedge n_i^{else} \approx n_i^{then}).$$

PROOF. By contradiction. If $n_i^{else} \neq n_j$, then, for any completely specified functions h at level $\mathcal{L}(n_j)$ or higher, a minterm m can be selected in such a way that $n_i^{else}(m) \neq h(m)$ and $m_{\mathcal{L}(n_i)} = 0$. This minterm shows that $n_i \neq h$, thereby showing that $n_i \neq n_j$. Identically for the *then* branch. If $n_i^{else} \neq n_i^{then}$, then, by Lemma 4.2, there are minterms w and m such that

$$n_i^{else}(w) \neq n_i^{then}(w) \wedge w_{\mathcal{L}(n_i)} = 0$$

and

$$n_i^{else}(m) \neq n_i^{then}(m) \wedge m_{\mathcal{L}(n_i)} = 1.$$

These minterms can be chosen to differ only in the value of the variable $x_{\mathcal{L}(n_i)}$ and lead to incompatible terminal nodes. Therefore, n_i cannot be compatible with any function h such that $\mathcal{L}(h) \geq \mathcal{L}(n_j)$, thereby showing that $n_i \neq n_j$. \square

The previous two lemmas justify the following algorithm to build the compatibility graph.

Algorithm 4.1

- 1) Initialize G with a complete graph except for edge (g_z, g_o) that is removed.
- 2) If $\mathcal{L}(g_i) = \mathcal{L}(g_j)$, then the edge between g_i and g_j has two labels: an e label with $\{g_i^{else}, g_j^{else}\}$ and a t label with $\{g_i^{then}, g_j^{then}\}$ (by Lemma 4.4).
- 3) If $\mathcal{L}(g_i) < \mathcal{L}(g_j)$ edge (g_i, g_j) has an l label with $\{g_i^{else}, g_i^{then}, g_j\}$ (by Lemma 4.5).
- 4) For all pairs of nodes (g_i, g_j) , check if the edge between nodes g_i and g_j has a label that contains $\{g_a, g_b\}$ and there is no edge between g_a and g_b . If so, remove the edge between g_i and g_j . Repeat this step until no more changes take place.

Fig. 3 shows an example of the 3TBDD F obtained from f defined by the following sets: $f_{on} = \{011, 111\}$, $f_{off} = \{010, 110, 101\}$ and the corresponding compatibility graph. Fig. 4 shows the three steps required to compute the final compatibility graph. The initial graph (Fig. 4a) is complete, except for the edge between nodes g_z, g_o . Fig. 4b shows the result after

\square

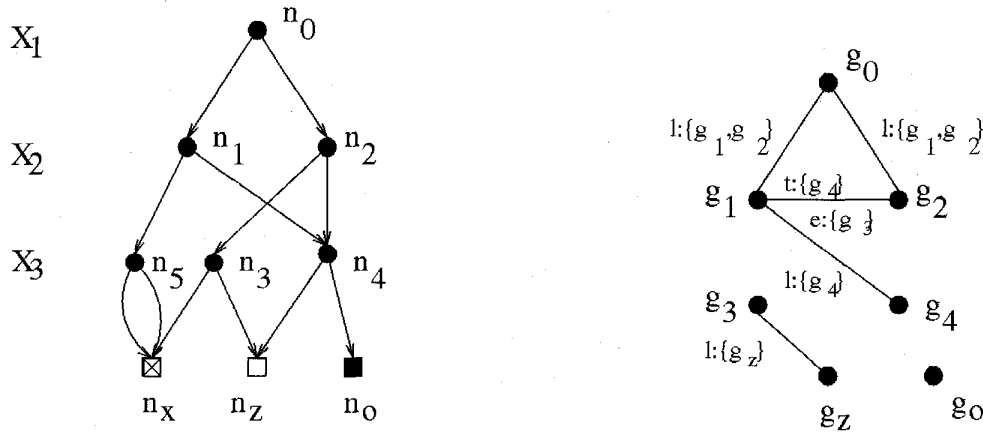


Fig. 3. The 3TBDD F and the compatibility graph G . Nodes g_5 and g_x are not shown on the compatibility graph since they are common support compatible with every node in the graph.

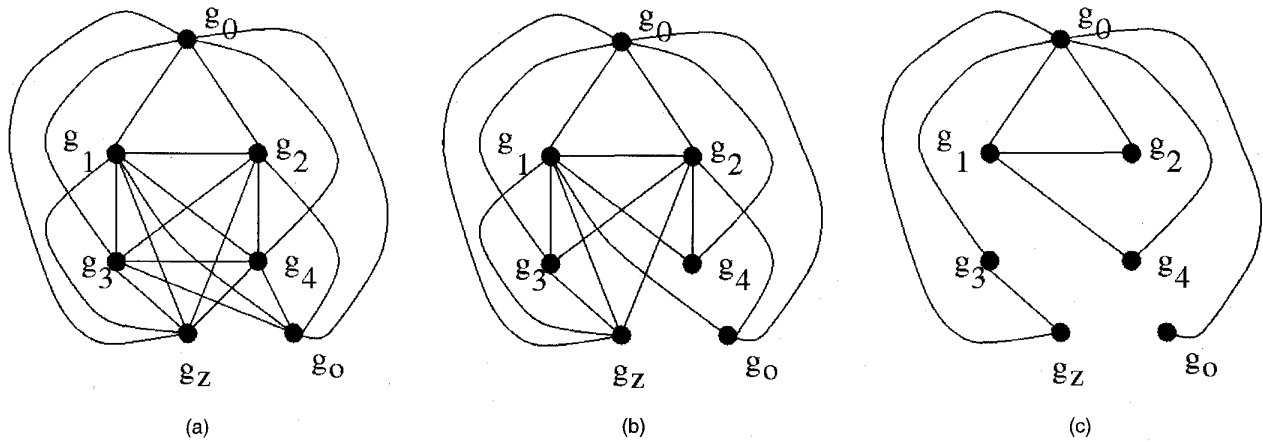


Fig. 4. The three steps required to compute the final compatibility graph shown in Fig. 3. To avoid excessive cluttering, edge labels and nodes g_5 and g_x are not shown.

the first step, when all edges with labels containing $\{g_z, g_o\}$ are removed. For example, the edge between g_3 and g_4 with t label $\{g_z, g_o\}$ is removed. The same happens with the edge between g_4 and g_o with l label $\{g_z, g_o, g_o\}$. This step removes all edges that should be removed involving nodes in the bottom two levels. The second step removes all edges that are to be removed involving nodes in the bottom three levels and the fourth step removes all remaining edges. In general, the procedure will require N steps, where N is the number of input variables, and each step requires a maximum of $|E|$ operations. Therefore, the procedure has total complexity given by $O(N|V|^2)$, where V is the number of nodes in the 3TBDD.

The existence of an edge in the compatibility graph is related with common support compatibility and with compatibility between pairs of nodes. In particular, if two nodes are common support compatible, then there will be an edge between these two nodes. The existence of an edge implies that the two nodes are compatible. However, given that compatibility is a weaker notion than common support

compatibility, the reverse is not true. This is formally stated in the following lemma:

LEMMA 4.6. $n_i \approx n_j \Rightarrow \exists e \in E \text{ s.t. } e = (g_i, g_j) \Rightarrow n_i \sim n_j$.

PROOF. The first implication follows from Lemmas 4.4 and 4.5 and the algorithm definition. Given these lemmas, the algorithm only removes edges that are between nodes that are not CSC. Let us prove the second implication by contradiction. Now, let $I = \mathcal{L}(n_i)$ and $J = \mathcal{L}(n_j)$ and assume that $n_i \not\sim n_j$ and that $I \leq J$. Then, there exists a minterm m such that $n_i(m) \neq n_j(m)$. Let $n_{a_1}, n_{a_{I+1}}, \dots, n_{a_{N+1}}$ be the sequence of nodes defined by minterm m in the BDD between the level I and $N + 1$, starting at n_i . Let $n_{b_1}, n_{b_{J+1}}, \dots, n_{b_{N+1}}$ be the sequence of nodes defined by minterm m between level J and $N + 1$ starting at node n_j . Then, because $n_{a_{N+1}}$ and $n_{b_{N+1}}$ are the terminal nodes and there is no edge between them, after k steps, the edge between node $g_{a_{N+1-k}}$ and node $g_{b_{N+1-k}}$ will have been removed because of

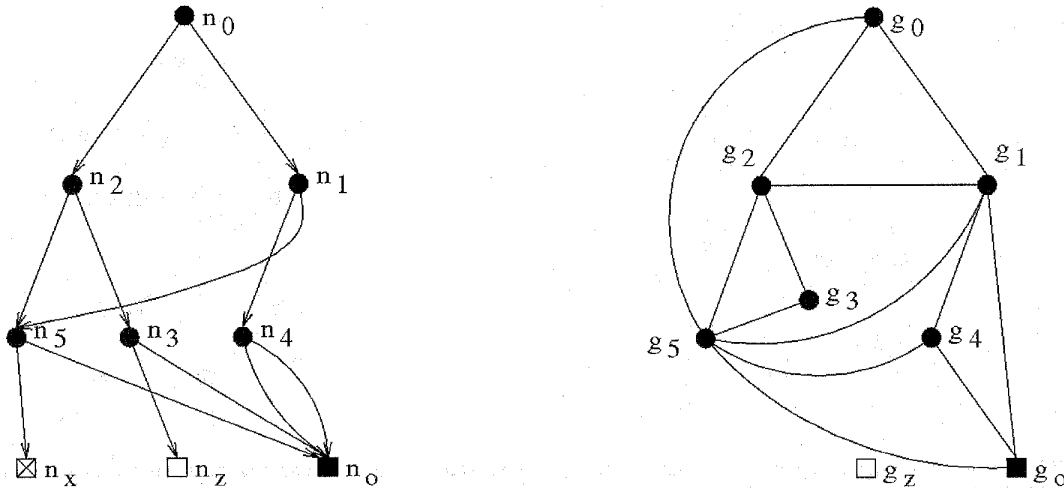


Fig. 5. Nodes n_0 and n_5 are not common support compatible but the compatibility graph does have an edge between the corresponding nodes.

condition 2. This will happen for k such that $N + 1 - k \geq J$. For k such that $N + 1 - k < J$, the edge between $g_{a_{N+1-k}}$ and g_{b_j} will have been removed because of condition 3. \square

It is important to note that the reverse implications are not true. In particular, the existence of an edge between two nodes in G does not imply that they are common support compatible. Consider the 3TBDD shown in Fig. 5.

For this 3TBDD, the algorithm described above does not remove the edge between nodes g_0 and g_5 because there are long range dependencies that cannot be found by the simple minded algorithm used to prune away edges. The edge between g_0 and g_5 has the following l label: $\{g_5, g_1, g_2\}$.

The edge between g_1 and g_5 has the l label $\{g_5, g_4\}$ and the edge between g_2 and g_5 has the l label $\{g_5, g_3\}$. Because n_5 is compatible with both n_3 and n_4 , the edge between g_1 and g_5 and the edge between g_2 and g_5 are never removed. Moreover, n_1 and n_2 are compatible. Therefore, the edge between g_0 and g_5 is never removed. However, no function depending only on the last variable can be compatible with n_0 and, therefore, n_0 and n_5 are not common support compatible.

Despite the fact that an edge between two nodes does not imply common support compatibility, this graph can be used to formulate a closed clique covering problem that yields the solution to the problem. This is done by defining an extra restriction on the type of cliques that can be used to cover the graph.

5 CLOSED CLIQUE COVERS

A clique of graph G is a completely connected subgraph of G . To any set s of nodes that is a clique of G there are associated class sets. If the nodes in s are to be merged into one, the nodes in its class sets are also required to be in the same set. Let $s_i = \{g_{i_1}, g_{i_2}, \dots, g_{i_w}\}$ be a set of nodes that form a clique in G . The following are the definitions of the e , t , and l classes of s_i . In some cases, we may blur the distinction between the nodes g s of G and the corresponding nodes n s

of F . Strictly speaking, cliques are defined on sets of g s and compatibles on sets of n s.

DEFINITION 5.1. The e class of s_i , $C_e(s_i)$ is the set of nodes that are in some e label of an edge between a node g_j and g_k in s_i with $\mathcal{L}(n_k) = \mathcal{L}(n_j) = \mathcal{L}_{\max}(s_i)$.

DEFINITION 5.2. The t class of s_i , $C_t(s_i)$ is the set of nodes that are in some t label of an edge between a node g_j and g_k in s_i with $\mathcal{L}(n_k) = \mathcal{L}(n_j) = \mathcal{L}_{\max}(s_i)$.

DEFINITION 5.3. The l class of s_i , $C_l(s_i)$ is the set of nodes that are in some l label of an edge between a node g_j and g_k in s_i with $\mathcal{L}(n_j) \neq \mathcal{L}(n_k)$.

LEMMA 5.1. If a set s_i of nodes are a clique of G and $C_l(s_i) \subseteq s_i$, then s_i is a compatible set.

PROOF. Let k be the maximum level of any node in s_i . The definition of $C_l(s_i)$ implies that $C_l(s_i)$ includes all the nodes of s_i at level k that are descendants of some node in s_i . Call these nodes the *foundation* of s_i . Because these nodes are a clique of G (or else they wouldn't be in s_i), they are all pairwise compatible, by Lemma 4.6. Because they are at the same level, Lemma 4.2 implies that they are pairwise CSC. Therefore, by the result of Lemma 4.3, these nodes are a compatible set. This implies that there exists a completely specified function h at the level $\mathcal{L}_{\max}(s_i)$ that is compatible with every node in the foundation of s_i . To finish the proof, we need to show that this function must be also compatible with every other node in s_i . To show this, assume that h is not compatible with some node n_j in s_i . Then, there must exist a minterm m such that $n_j(m) \neq h(m)$. This minterm defines a path in the BDD that goes through a node n_k in the foundation of s_i . Since $n_k(m) \neq h(m)$, n_k and h are not compatible, which violates the assumption that h is

compatible with every node in the foundation of s_i .

Therefore, h must be compatible with every node in s_i , thereby satisfying Definition 4.3. \square

Note that a clique of G that does not satisfy the condition in Lemma 5.1 is not necessarily a compatible set. For instance, in the example in Fig. 5, the nodes $\{g_0, g_1, g_2, g_5\}$ are a clique of G but are not a compatible set, because $g_3 \in C_l(\{g_0, g_1, g_2, g_5\})$ but $g_3 \notin \{g_0, g_1, g_2, g_5\}$.

The algorithm that selects the minimum BDD compatible with the original function works by selecting nodes of G that can be merged into one node in the final BDD. If a set s of nodes in G is to be merged into one, the set s has to be a compatible set. Therefore, it has to be a clique of G satisfying Definition 5.3. The objective is to find a set of cliques such that every node in G is covered by at least one clique. However, to obtain a valid solution, some extra conditions need to be imposed.

DEFINITION 5.4. A set $S = \{s_1, s_2, \dots, s_n\}$ of sets of nodes in G is called a closed clique cover for G if the following conditions are satisfied:

- 1) S covers G : $\forall g_i \in G \exists s_j \in S : g_i \in s_j$.
- 2) All s_k are cliques of G : $\forall g_i, g_j \in s_k : (g_i, g_j) \in \text{edges}(G)$.
- 3) S is closed with respect to the e and t labels:

$$\forall s_i \in S \exists s_j \in S : C_e(s_i) \subseteq s_j \wedge \forall s_i \in S \exists s_j \in S : C_t(s_i) \subseteq s_j.$$

- 4) All sets in S are closed with respect to the l labels:
 $\forall s_i \in S : C_l(s_i) \subseteq s_i$.

6 GENERATION OF A MINIMUM BDD

From a closed clique cover for G , a reduced BDD R is obtained by the following algorithm:

Algorithm 6.1

- 1) For each s_i in S , create a BDD node in R , r_i , at level $L_{\max}(s_i)$.
- 2) Let the nodes in R that correspond to sets s_i containing nodes that correspond to terminal nodes in F be the new corresponding terminal nodes of R .
- 3) Let the *else* edge of the node r_i go to the node r_j that corresponds to a set s_j such that $C_e(s_i) \subseteq s_j$.
- 4) Let the *then* edge of the node r_i go to the node r_j that corresponds to a set s_j such that $C_t(s_i) \subseteq s_j$.

LEMMA 6.1. R is an Ordered BDD compatible with F .

PROOF. Since the cover is closed, Steps 3 and 4 are always feasible. Any path in F that leads to a 1 or a 0 will lead to the corresponding terminal node in R . Finally, there will never be edges going upward in R because the node that results from a set s_i is at the lowest level of all the nodes in s_i . \square

Using an efficient data structure to keep the cover information, this algorithm can be executed in time linear on the number of nodes in the original BDD. In practice, this is of limited interest, since the procedure of actually finding the minimum cover may take exponential time. In fact, finding the minimum cover is the only step that is computationally inefficient, but it is unavoidable since we know that the

problem is NP-complete. The requirement that the minimum cover must be closed implies that the covering problem to be solved is binate. We refer to [11] for an extended survey on covering problems.

Now, the main result follows: Let \mathcal{B} be the set of all BDDs that represent functions compatible with the incompletely specified function f . Then, the following result holds:

THEOREM 6.1. For a given variable ordering, the BDD induced by a minimum closed cover for G is the BDD in \mathcal{B} with minimum number of nodes.

PROOF. Given the result in Lemma 6.1, it is sufficient to prove that there exists at least one closed cover of cardinality equal to the size of the minimum BDD in \mathcal{B} .

Let U be a BDD in \mathcal{B} with minimum number of nodes k . For each node in U , u_i , create a set s_i such that g_j is in s_i iff $n_j \sim u_i$ and $\mathcal{L}(n_j) \leq \mathcal{L}(u_i)$. Let $S = \{s_1, s_2, \dots, s_k\}$. We will show that S satisfies all the conditions in Definition 5.4:

- 1) (S covers G) We show that the assumption that some g_i at level l is not in some set of S leads to a contradiction: Let m be a minterm that defines a path in F that starts at the root and goes through n_i . Let M be the set of all minterms that have the same values as m for $x_1 \dots x_{l-1}$. Each one of these minterms will define a path in U that goes through some node u_j in U at a level equal or higher than l . Since $n_i \not\sim u_j$ (by the hypothesis), there exists a minterm $m' \in M$ such that $u_j(m') \not\sim n_i(m')$. For this minterm m' , $n_0(m') \not\sim u_0(m')$, thereby contradicting the assumption that U is compatible with F .
- 2) (All $s_i \in S$ are cliques of G) Since each node in s_i is compatible with a completely specified function (u_i), they satisfy Definition 4.3 and, therefore, by Lemma 4.1, they are a clique of G .
- 3) (S is closed with respect to the e and t labels) Let u_i be a node in U , $u_a = u_i^{\text{else}}$ and $u_b = u_i^{\text{then}}$. Let $b_i = \{g_j \in s_i : \mathcal{L}(g_j) = \mathcal{L}_{\max}(s_i)\}$. For each node $g_j \in b_i$, $n_j \sim u_i$ implies $u_a \sim n_j^{\text{else}}$ and $u_b \sim n_j^{\text{then}}$. Therefore, $C_e(s_i) \subseteq s_a$ and $C_t(s_i) \subseteq s_b$.
- 4) (S is closed with respect to the l labels) Suppose $C_l(s_i) \not\subseteq s_i$. Then, there must be a node n_w such that $g_w \in s_i$ at level $l < \mathcal{L}(u_i)$ and $g_w^{\text{else}} \notin s_i$ or $g_w^{\text{then}} \notin s_i$. Assume the first is true and let $n_w^{\text{else}} = n_a$; n_a is not compatible with u_i (or else it would be in s_i) and depends only on the variables $\{x_{l+1} \dots x_n\}$. Therefore, there exists a minterm m such that $u_i(m) \not\sim n_a(m)$ and $m_l = 0$. This minterm shows that $n_w \not\sim u_i$, which contradicts the hypothesis that g_w is in s_i .

Therefore, S is a closed clique cover for G and it has cardinality k . \square

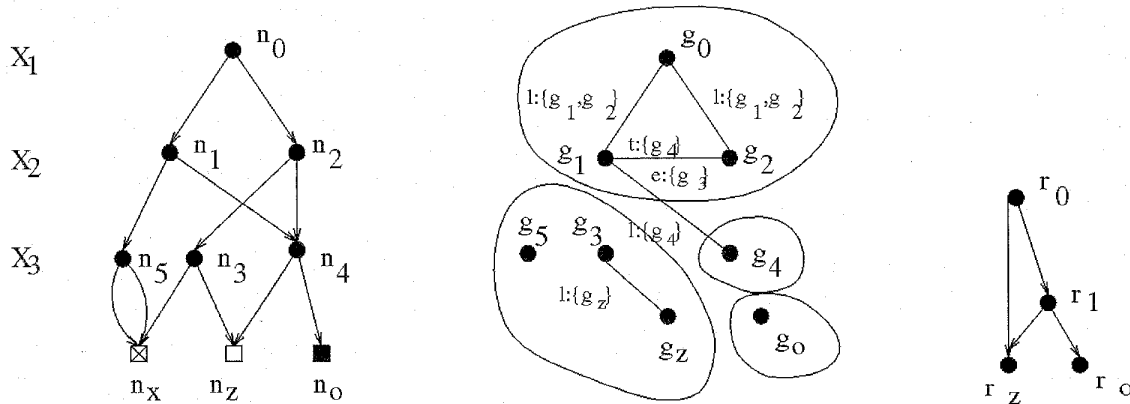


Fig. 6. The 3TBDD F , the compatibility graph G and a solution R . Node g_5 was arbitrarily included in compatible $\{g_3, g_2, g_5\}$.

As an example, $S = \{\{g_0, g_1, g_2\}, \{g_4\}, \{g_3, g_5, g_2\}, \{g_0\}\}$ is a closed cover for the example depicted in Fig. 3 and induces the BDD R shown on the right side of Fig. 6.

7 FORMULATION OF BDD MINIMIZATION AS FSM REDUCTION

The definition of a closed cover is very similar to the standard definition of a closed cover used in the minimization of finite state machines (FSMs). If the graph of a 3TBDD is viewed as the state transition graph of an FSM, the algorithms developed for the minimization of FSMs can be used with some modifications. The two important differences to consider are:

- 1) The definition of the e and t classes and the closure requirement in point 3 of Definition 5.4 are different from the definitions used in standard FSM minimization. In BDD minimization, only nodes at the highest level in some compatible define the e and t classes, while in standard FSM minimization all nodes in a compatible set are involved in the definition of these classes.
- 2) The requirement in point 4 of Definition 5.4 means that some sets of nodes that satisfy the definition of a compatible set in the FSM case do not satisfy the conditions for BDD minimization.

These two changes can be incorporated into existing algorithms for FSM minimization. In particular, the closure conditions with respect to the e and t labels are similar to the closure conditions imposed in standard FSM minimization. The restriction imposed by condition 4 in Definition 5.4 simply eliminates some cliques of the compatibility graph from consideration and can be implemented by a filtering step.

Let F be the 3TBDD that should be minimized, and consider the FSM with a state transition graph (STG) obtained from F in the following way:

- Initialize the STG with a graph isomorphic to the 3TBDD, with nodes $S_0, S_1, \dots, S_o, S_z, S_x$ each one corresponding to one node in F .
- Add a new node, S_f .

- Add transitions from S_z, S_o , and S_x to S_f labeled $-/0$, $-/1$, and $-/-$, respectively.
- Add a transition from S_f to S_f labeled $-/-$.

As an example, consider the FSM obtained from the 3TBDD in Fig. 3, shown in Fig. 7.

This transformation leads to our final important result. Let M be the incompletely specified FSM with the state transition graph obtained from F by the procedure outlined above and let G' be the compatibility graph for this finite state machine built in the following way:

- There is an edge between nodes g'_i and g'_j in G' if states n'_i and n'_j are compatible, in the FSM reduction sense.
- Edges of G' are labeled in accordance with Algorithm 4.1.

Note that the structure of graph G' reflects the compatibility between states of M defined as for FSM reduction. Therefore, G' can be computed using the standard procedures for FSM minimization. However, this means that G' is not isomorphic to G , defined by Algorithm 4.1. In particular, any two nodes that are at different levels in G' are compatible for FSM reduction, and so G' always has an edge between g'_i and g'_j if $L(g'_i) \neq L(g'_j)$ and, therefore, the second implication in Lemma 4.6 is not true, in general.

The following lemma establishes that the cliques of G and the cliques of G' that are closed with respect to the l class are exactly the same.

LEMMA 7.1. A set $s'_i = \{g'_{a_1} \dots g'_{a_k}\}$ is a clique of G' satisfying $C_l(s'_i) \subseteq s'_i$ iff $s_i = \{g_{a_1} \dots g_{a_k}\}$ is a clique of G satisfying $C_l(s_i) \subseteq s_i$.

PROOF. Since the edges of G' are a superset of the edges of G and the labels are the same, it is clear that, to any clique of G containing $C_l(s_i)$, there corresponds a clique of G' satisfying that condition. To prove the other direction, note that, for any two nodes in G' at the same level, the presence of an edge in G' implies that they are compatible, both in the sense of FSM reduction and according to Definition 4.1. Therefore, the rightmost implication of Lemma 4.6 is valid for nodes that are at the same level. The proof of Lemma 5.1

only uses this implication for nodes at the same level, namely the nodes in the foundation of s_i defined in that lemma. Therefore, Lemma 5.1 is still true if G is replaced by G' , implying that any set s'_i in G' that satisfies $C_i(s'_i) \subseteq s'_i$ corresponds to a compatible of G' , according to Definition 4.3. Therefore, the corresponding set s_i in G is a clique. Because the labels are the same for any edges common to both G and G' , $C_i(s_i) \subseteq s_i$. \square

COROLLARY 7.1. *A minimum closed cover for M satisfying Definition 5.4 when G is replaced by G' induces a minimum BDD compatible with F , in accordance with Theorem 6.1.*

PROOF. Since the compatibles that can be part of the cover are the same in both G' and G , this result follows directly from Theorem 6.1. \square

8 IMPLICIT COMPUTATION OF A MINIMUM CLOSED COVER

We will use the unified implicit framework proposed in [10].² Implicit techniques are based on the idea of operating on discrete sets by their characteristic functions represented by binary decision diagrams (BDDs) [4].

To perform state minimization, one needs to represent and efficiently manipulate sets of sets of states. With n states, each subset of states is represented in **positional-set** form, using a set of n Boolean variables, $x = x_1x_2\dots x_n$. The presence of a state s_k in the set is denoted by the fact that variable x_k takes the value 1 in the positional-set, whereas x_k takes the value 0 if state s_k is not a member of the set. For example, if $n = 6$, the set with a single state s_4 is represented by 000100 while the set of states $s_2s_3s_5$ is represented by 011010.

A set of sets of states S is represented in positional notation by a characteristic function $\chi_S : B^n \rightarrow B$ as: $\chi_S(x) = 1$ if and only if the set of states represented by the positional-set x is in the set S . A BDD representing $\chi_S(x)$ will contain minterms, each corresponding to a state set in S . As an example, $Tuple_{n,k}(x)$ denotes all positional-sets x with exactly k states in them (i.e., $|x| = k$). For instance, the set of singleton states is $Tuple_{n,1}(x)$, the set of state pairs is $Tuple_{n,2}(x)$, the set of full states is $Tuple_{n,n}(x)$, and the set of empty states is $Tuple_{n,0}(x)$. An alternative notation for $Tuple_{n,k}(x)$ is $Tuple_k(x)$.

Any **relation** R between pairs of sets S_1 and S_2 can be represented by its characteristic function $\mathcal{R} : B^n \times B^n \rightarrow B$ where $\mathcal{R}(x, y) = 1$ if and only if $\chi_{S_1}(x) = 1$, $\chi_{S_2}(y) = 1$, and the element of S_1 represented by x is in relation R with the element of S_2 represented by y . A similar definition holds for relations defined over more than two sets. For example, we represent the state transition graph (STG) of an FSM by the characteristic functions of two relations:

- 1) the output relation Λ , where input i , present state p and output o are in $\Lambda(i, p, o)$ if there is an edge from p with input/output label i/o , and

² $\exists x(\mathcal{F})$ ($\forall x(\mathcal{F})$) denotes the existential (universal) quantification of function \mathcal{F} over variables x ; \Rightarrow denotes Boolean implication; \Leftrightarrow denotes XNOR; \neg denotes NOT.

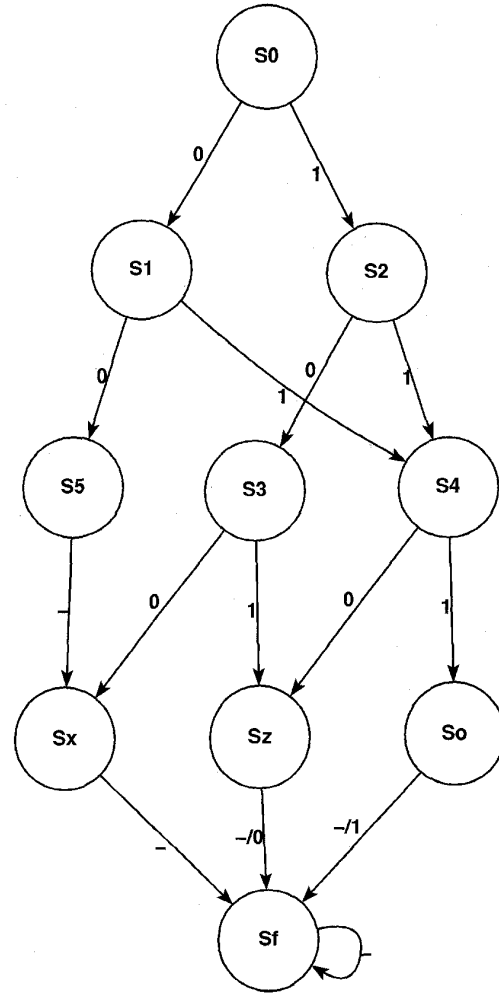


Fig. 7. The STG that corresponds to the 3TBDD F defined in Fig. 3.

- 2) the next state relation \mathcal{T} , where input i , present state p , and next state n are in relation $\mathcal{T}(i, p, n)$ if there is an edge from p to n with input label i .

PROPOSITION 8.1. *Set equality, containment, and strict-containment between two positional-sets x and y are expressed by:*

$$(x = y) = \prod_{k=1}^n (x_k \Leftrightarrow y_k); (x \supseteq y) = \prod_{k=1}^n (y_k \Rightarrow x_k);$$

$$\text{and } (x \supset y) = (x \supseteq y) \cdot (x \neq y).$$

PROPOSITION 8.2. *Given two sets of positional-sets, complementation, union, intersection can be performed on them as logical operations (\neg , $+$, \cdot) on their characteristic functions.*

DEFINITION 8.1. *The substitution in the function \mathcal{F} of variable x_i with variable y_i is denoted by:*

$$[x_i \rightarrow y_i]\mathcal{F} = \mathcal{F}(x_1, \dots, x_{i-1}, y_i, x_{i+1}, \dots, x_n)$$

and the substitution in the function \mathcal{F} of a set of variables $x = x_1x_2\dots x_n$ with another set of variables $y = y_1y_2\dots y_n$ is obtained simply by:

```


$$\begin{aligned} & \mathcal{L}_0(p) = r(p) \\ & k = 0 \\ & \text{do } \{ \\ & \quad \mathcal{L}_{k+1}(p) = [n \rightarrow p] \exists p, i [\mathcal{L}_k(p) \cdot T(i, p, n)] \\ & \quad k = k + 1 \\ & \} \text{ until } \mathcal{L}_{k+1}(p) = \mathcal{L}_k(p) \end{aligned}$$


```

Fig. 8. Computation of array \mathcal{L} .

```


$$\begin{aligned} & \text{Level}(p, u) = \emptyset \\ & \text{for } (i = 0; i < k; i++) \{ \\ & \quad \text{for each state } Tuple_1(p) \in \mathcal{L}_i(p) \{ \\ & \quad \quad \text{for } (j = i; j < k; j++) \{ \\ & \quad \quad \quad \text{for each state } Tuple_1(u) \in ([u \rightarrow p] \mathcal{L}_j(p)) \{ \\ & \quad \quad \quad \quad \text{Level}(p, u) = \text{Level}(p, u) + (Tuple_1(p) \cdot Tuple_1(u)) \\ & \quad \quad \quad \} \\ & \quad \quad \} \\ & \quad \} \\ & \} \end{aligned}$$


```

Fig. 9. Computation of the relation $\text{Level}(p, u)$.

$$[x \rightarrow y]F = [x_1 \rightarrow y_1][x_2 \rightarrow y_2] \dots [x_n \rightarrow y_n]F.$$

PROPOSITION 8.3. *The maximal of a set χ of subsets is the set containing subsets in χ not strictly contained by any other subset in χ , and can be computed as:*

$$\text{Maximal}_\chi(\chi) = \chi(x) \cdot \exists [(y \supset x) \cdot \chi(y)].$$

8.1 Implicit Generation of Compatibles

It has been shown in Section 7 that, given a BDD minimization problem, it is possible to generate a companion FSM whose closed covers of compatibles correspond to closed clique covers of the BDD, if:

- FSM compatibles that do not satisfy the L-closure are discarded, and
- FSM compatible closure is replaced by E-closure and T-closure.

Our starting point is the fully implicit algorithm for exact state minimization reported in [10], to which we refer for a complete description of the implicit computations. In the sequel, we discuss the modifications needed to generate closed clique covers of the BDD.

8.2 Implicit Computation of L-Closure

We compute as in [10] the set of compatibles $C(c)$, where $C(c) = 1$ iff c is the positional set representing a compatible of the companion FSM. When minimizing an FSM obtained from an instance of BDD minimization, one must delete from $C(c)$ the compatibles c that are not closed with respect to their l -class. The l -class, $C_l(c)$, of a compatible c is the set of nodes that are in some l -label of an edge between nodes g_j and g_k in c with $L(g_j) < L(g_k)$. If $L(g_j) < L(g_k)$, then edge (g_j, g_k) has the l -label $\{g_j^{\text{else}}, g_j^{\text{then}}, g_k\}$.

It is shown next how to capture the information on the level of the nodes. By construction, an FSM obtained by BDD minimization is represented by a direct acyclic STG rooted at the unique reset state r ; each node has two successors, except the terminal node that has a self-loop. Fig. 8 illustrates a procedure to build an array $\mathcal{L}(p)$, that partitions the FSM states based on their distance from the root: $\mathcal{L}_k(p)$ is the set of states associated to the nodes having a distance k from r . Starting from r and visiting in breadth-first order the STG, one computes iteratively the array elements $\mathcal{L}_k(p)$, using the transition relation $T(i, p, n)$. In fact, state n is a successor of state p iff $\exists i T(i, p, n)$.

Using the informations stored in $\mathcal{L}(p)$, one defines the order relation $\text{Level}(p, u)$, for each couple of states (p, u) in the FSM. States p and u are in relation $\text{Level}(p, u)$ iff the distance of p from r is less than the distance of u from r , i.e. formally

$$\text{Level}(p, u) = 1 \Leftrightarrow \exists i \exists j \{(i < j) \wedge \mathcal{L}_i(p) \cdot \mathcal{L}_j(u)\}. \quad (1)$$

Fig. 9 illustrates the procedure to compute the global relation $\text{Level}(p, u)$.

A compatible c is pruned from the set of compatibles $C(c)$ if:

- 1) c contains states p and u that are in the order relation $\text{Level}(p, u)$,
- 2) c does not contain all the successors of p .

Hence, the filtered set of compatibles is given by:

$$C(c) = C(c) -$$

$$\begin{aligned} & \exists p \{ \exists u [Tuple_1(p) \cdot Tuple_1(u) \cdot (c \supseteq p) \cdot (c \supseteq u) \cdot \text{Level}(p, u)] \\ & \quad \exists n [\exists i (i, p, n) \cdot (c \not\supseteq n)] \}. \end{aligned} \quad (2)$$

$$\begin{array}{l}
Max_Level(c, c') = \emptyset \\
\mathcal{C}_j(c) = \mathcal{C}(c) \\
\text{for } (j = k - 1; j = 0; j - -) \{ \\
\quad \mathcal{ML}_j(c, c') = \mathcal{C}_j(c) \cdot \exists l[\mathcal{L}_j(l) \cdot \sum_{n=1}^N (l_n \cdot c_n)] \cdot \prod_{n=1}^N \{c'_n - [c_n \cdot \exists l(l_n \cdot L_j(l))]\} \\
\quad Max_Level(c, c') = Max_Level(c, c') + \mathcal{ML}_j(c, c') \\
\quad \mathcal{C}_j(c) = \mathcal{C}_j(c) - \exists c'[\mathcal{ML}_j(c, c')] \\
\}
\end{array}$$

Fig. 10. Computation of the relation $\mathcal{C}(c).Max_Level(c, c')$.

8.3 Implicit Computation of E-Closure and T-Closure

In standard FSM minimization, one requires closure with respect to implied sets. Given a compatible c , an implied set under input i is the set of next states from the states in c under i . Instead, in the case of BDD minimization, one must compute the implied sets only from the states in c of highest level. This requires a change in the computation of the relation of the implied classes $\mathcal{F}(c, i, n)$, which is used by the following procedures:

- 1) the computation of primes,
- 2) the set up of the binate clauses in the covering table,
- 3) the construction of a reduced FSM.

The new computation for $\mathcal{F}(c, i, n)$ is described by the following equation:

$$\mathcal{F}(c, i, n) = \exists p \{ \exists c' [\mathcal{C}(c) \cdot Max_Level(c, c') \cdot (c' \supseteq p)] \cdot \mathcal{T}(i, p, n) \}. \quad (3)$$

Subsets of states c and c' are in relation $Max_Level(c, c')$, iff c' is the subset of c that contains the states of c of maximum level, i.e., the states having the largest distance from r in the STG of the FSM.

The computation of the relation $\mathcal{C}(c).Max_Level(c, c')$ is based on the availability of $\mathcal{L}(p)$ and is summarized in Fig. 10. For each level j starting from the maximum to the minimum, a relation $\mathcal{ML}_j(c, c')$ is determined performing N bitwise conjunctions, where N is the number of states. The n th element of c' is 1 iff the n th element of c is 1 and it has level j . $\mathcal{ML}_j(c, c')$ represents the pairs (c, c') such that c is a compatible that contains at least one state at level j and no state at level greater than j , and c' has exactly the states of c of level j . Before examining level $j - 1$, $Max_Level(c, c')$ is updated adding the elements in $\mathcal{ML}_j(c, c')$, and the sets c already in $\mathcal{ML}_j(c, c')$ are removed from $\mathcal{C}(c)$. Notice that the time complexity of the computation depends only linearly from the explicit parameters N , number of states, and k , number of levels in the STG representation.

8.4 Example

Fig. 7 illustrates the companion FSM obtained from the 3TBDD F shown in Fig. 3. The FSM has two pairs of incompatible states $(\{S_3, S_4\}, \{S_{3'}, S_{4'}\})$, while its set $\mathcal{C}(c)$ contains 575 compatibles. After filtering away by means of (2) the compatibles c that are not closed with respect to their l -class, 32 sets of compatibles are left:

$$\{S_j\}, \{S_0\}, \{S_{0'}, S_j\}, \{S_2\}, \{S_{2'}, S_j\}, \{S_3\}, \{S_{3'}, S_2\}, \{S_{3'}, S_{2'}\}, \{S_{3'}, S_{2'}\}, \{S_{3'}\},$$

$$\{S_{3'}, S_{2'}\}, \{S_{3'}, S_{0'}\}, \{S_{3'}\}, \{S_5\}, \{S_5, S_x\}, \{S_5, S_x, S_z\}, \{S_5, S_x, S_z, S_j\}, \\ \{S_5, S_x, S_0\}, \{S_5, S_x, S_j\}, \{S_5, S_x, S_{0'}\}, \{S_4\}, \{S_4, S_5\}, \{S_3\}, \\ \{S_3, S_x, S_2\}, \{S_3, S_x, S_z, S_j\}, \{S_3, S_5\}, \{S_3, S_5, S_x, S_z\}, \\ \{S_3, S_5, S_x, S_z, S_j\}, \{S_2\}, \{S_1\}, \{S_1, S_2\}, \{S_1, S_4, S_5\}, \{S_0\}, \{S_0, S_1, S_2\}.$$

Using the computation of the implied classes $\mathcal{F}(c, i, n)$ of (3), the following eight primes are identified from the previous 32 compatibles:

$$\{S_0\}, \{S_0, S_1, S_2\}, \{S_1\}, \{S_1, S_4, S_5\}, \{S_2\}, \{S_3, S_5, S_x, S_z, S_j\}, \{S_4\}, \\ \{S_5, S_x, S_{0'}\}.$$

Among the eight primes, there are two essential primes:

$$\{S_3, S_5, S_x, S_z, S_j\}, \{S_5, S_x, S_{0'}\}$$

and six nonessential primes:

$$\{S_0\}, \{S_0, S_1, S_2\}, \{S_1\}, \{S_1, S_4, S_5\}, \{S_2\}, \{S_4\}.$$

After solving the binate covering problem, two nonessential primes are chosen:

$$\{S_0, S_1, S_2\}, \{S_1, S_4, S_5\}.$$

Hence, the final reduced FSM has the following four states:

$$R_0 \leftarrow \{S_0, S_1, S_2\} \\ R_1 \leftarrow \{S_1, S_4, S_5\} \\ R_z \leftarrow \{S_3, S_5, S_x, S_z, S_j\} \\ R_0 \leftarrow \{S_5, S_x, S_{0'}\}$$

and is described by the following state transition table:

0	$R_0 R_z$	-
1	$R_0 R_1$	-
0	$R_1 R_z$	-
1	$R_0 R_0$	-
-	$R_z R_z$	0
-	$R_0 R_0$	1

This state transition table induces the BDD R shown on the right side of Fig. 6. R is an exact solution of the BDD minimization problem.

9 RESULTS

Starting from the program ISM for implicit state minimization [10], we developed IMAGEM, a new program based on the theory described in this paper for exact BDD minimization. In particular, we transformed the implicit algorithm for exact state minimization in a new algorithm for the

TABLE 1
RESULTS ON MACHINE LEARNING PROBLEMS

example	# orig. states	# compat.	# filtered compat.	# prime compat.	# red. states	heuristic	IMAGEM CPU time (sec)
dnfa	64	2.435821e+12	1332186	89	14	16	517.29
dnfb	36	4.853883e+08	2987	94	6	12	11.85
dnfc	40	2.291224e+08	2613	102	10	15	12.94
dnfd	93	1.137739e+20	9.517899e+08	-	-	23	timeout
dnfe	63	2.102303e+13	141179	509	6	12	217.29
dnff	62	2.184367e+11	92027	357	15	22	151.8
xor3	9	179	14	7	6	6	0.1
xor4	17	14975	118	13	6	6	0.43
xor5	24	608255	267	36	9	10	1.37
xor6	40	3.355914e+08	1329	170	13	20	13.98
xor7	57	2.791115e+11	3076	640	15	31	88.16
xor8	94	1.539147e+17	164929	21830	17	45	9041.11
<i>ex. paper</i>	10	575	32	8	4	4	0.17

TABLE 2
RESULTS ON PROBLEMS FROM *ESPRESSO* BENCHMARK SUITE

example	# orig. states	# compat.	# filtered compat.	# prime compat.	# red. states	heuristic	IMAGEM CPU time (sec)
alu1	95	1.025649e+21	841993	1204	6	6	7409.97
br1	74	2.99581e+18	799173	329	6	11	1313.91
br2	51	5.937363e+14	53687	78	3	8	14.59
clpl	50	1.467671e+13	7559	39	3	13	12.39
dc2	46	8.277148e+10	8831	98	8	12	57.66
exp	54	2.695432e+11	10638	25	3	3	31.34
exps	71	1.8345e+10	3810	125	43	44	44.79
in0	151	2.622416e+25	1680740	1323	42	44	18201.76
in3	173	5.060229e+39	587880	12	9	14	1755.21
inc	35	1.119744e+07	364	26	12	13	3.84
intb	189	4.884137e+46	3.891123e+14	-	-	69	spaceout
mark1	71	7.487812e+18	8049	35	4	5	41
newapla	52	1.24299e+12	3252	33	10	11	41.5
newapla1	57	8.766887e+14	8733	63	6	6	141.66
newapla2	19	93311	137	6	5	5	0.49
newbyte	16	20735	127	9	5	5	0.41
newcond	165	3.825623e+31	7.484552e+12	-	-	54	spaceout
newcpla2	39	3.396557e+08	477	68	10	21	5.72
newcwp	16	10367	106	10	6	11	0.39
newtpla	94	1.265561e+23	411525	148	7	23	469.14
newtpla1	39	6.912e+09	1441	31	4	5	4.45
newtpla2	26	3149279	158	9	9	9	0.9
newxcpla1	39	4.470682e+09	1473	35	5	10	5.13
p82	16	15551	102	10	7	7	0.4
prom1	65	5.189184e+09	382	77	50	50	30.04
prom2	33	2.17728e+08	446	38	12	12	3.33
sex	28	1.679616e+07	419	16	5	5	1.62
spla	155	1.647427e+39	1.401835e+12	-	-	8	spaceout
sqn	41	1.05336e+07	173	43	19	19	9.13
t4	68	5.108787e+14	31775	157	9	11	89.98
vg2	150	3.655064e+36	4.038678e+07	-	-	14	timeout
wim	14	4319	82	8	6	6	0.26

implicit computation of a minimum closed cover, as described in Section 8.

To experimentally evaluate the algorithms presented in this paper, we assembled two sets of problems: The first set

derives directly from a machine learning application and the second set was obtained from a logic synthesis benchmark. In all the problems, the original ordering specified for the variables was the ordering used.

For the first set of problems, 12 completely specified Boolean functions f_i were used as the starting point. For each of these functions, a randomly selected set of min-terms was designated as the *care set*, resulting in a set of incompletely specified Boolean functions g_i . The original objective was to identify the set of problems for which it is possible to recover exactly the original functions f_i from the incompletely specified functions g_i , thereby characterizing the conditions under which it is possible to infer the original function from a training set [14]. For the purposes of this work, the functions g_i are used solely as a set of incompletely specified functions. An advantage that exists for this set is that upper bounds on the size of the solution are well defined because the BDD sizes for the f_i are known. Under certain conditions, these upper bounds tend to become tight, with high probability, as the size of the problem increases, providing a welcome check for the results obtained.

The second set of problems was obtained by selecting a subset of the problems that are distributed with *Espresso* [3], a well-known two-level minimizer. More specifically, we included in this set of problems the functions that are the first output from each of the PLAs that are included in the *industry* subset of the *Espresso* benchmark suite. From this set, we eliminated all the functions that have a null don't care set, since, for these functions, the problem is trivial.

Table 1 summarizes the results obtained from running the set of machine learning problems and Table 2 the ones from the problems derived from the *Espresso* benchmark suite. The last entry in Table 1 is denoted *ex.paper* and simply refers to the case that has been presented in the paper to illustrate the theory.

For each example of a 3TBDD, the number of states of the companion FSM is reported in the column denoted "# *orig. states*." This number is always equal to the number of nodes of the 3TBDD plus one because a new node is added to the STG, as explained in Section 7. The following two columns report the number of compatibles of the FSM (i.e., the cardinality of the set $C(c)$) and the number of compatibles after filtering as per Section 8.2 (i.e., the ones which are closed with respect to their l -class). This step reduces the number of compatibles of many orders of magnitude.

Then, after the number of primes, in column "# *red. states*," we report the number of states of the reduced FSM. This number coincides with the number of nodes of the final BDD and represents the exact solution of the BDD minimization problem. Instead, the column denoted with the label "*heuristic*" presents the solutions obtained using the restrict operator [7], a well-known heuristic algorithm for BDD minimization; equal solutions are obtained using the constrain operator [6] (also known as generalized cofactor [19]).³ Other proposed heuristic algorithms [17] were uninteresting for comparison purposes, since, for these problems, they always obtained results equal or worse than results obtained by the restrict operator. Therefore, IMAGEM is the first exact algorithm that helps to evaluate the quality of the heuristics for BDD minimization on an interesting set of examples.

Moreover, as we discussed in Section 1, there are specific applications, as for instance inductive learning, where, from

3. Notice that the sizes of the BDD obtained by the heuristic algorithms have been measured without considering complemented edges.

one side, BDDs are used as very effective representation scheme, but, on the other side, heuristic algorithms return unsatisfactory solutions. The results reported in Table 1 show that IMAGEM returns the exact solution for a class of nontrivial problems.

The last column contains the time spent by IMAGEM to find the solution: All run times are reported in CPU seconds on a DEC Alpha (300 Mhz) with 2Gb of memory. For all experiments, "timeout" has been set at 21,600 seconds of CPU time and "spaceout" at 2Gb of memory.

10 CONCLUSIONS

This paper addresses the problem of binary decision diagram (BDD) minimization in the presence of don't care sets. Specifically, given an incompletely specified function g and a fixed ordering of the variables, we propose an exact algorithm for selecting f such that f is a cover for g and the binary decision diagram for f is of minimum size. We show that the minimum-sized binary decision diagram compatible with the specification can be found by solving a problem that is very similar to the problem of reducing an ISFSM. The approach described is the only known exact algorithm for this problem not based on the enumeration of the assignments to the points in the don't care set.

We show that this minimization problem can be formulated as a binate covering problem and solved using implicit enumeration techniques. We have implemented this algorithm and performed experiments, by means of which, exact solutions for an interesting benchmark set were computed. In particular, we could solve exactly some nontrivial examples from the learning literature, where quality of the solution is of paramount importance.

The current bottleneck of our implicit computation is the step from filtered compatibles to prime compatibles. It would be interesting to study new techniques for the implicit computation of prime compatibles or of a superset of them in order to enlarge the set of examples that can be solved exactly.

ACKNOWLEDGMENTS

The authors thank Timothy Kam for discussions on the reduction of BDD minimization to FSM state minimization and Thomas Shiple for discussions on BDD minimization and pointers to the literature. Partial funding for this research was provided by DARPA, NSF, SRC, and industrial grants from Bell Northern, Cadence, Digital, Fujitsu, Intel, MICRO, Motorola, and we gratefully acknowledge it.

REFERENCES

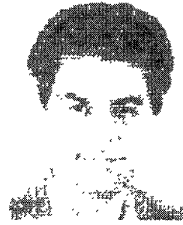
- [1] A. Blumer, A. Ehrenfeucht, D. Haussler, and M.K. Warmuth, "Occam's Razor." *Information Processing Letters*, vol. 24, pp. 377-380, Apr. 1987.
- [2] K. Brace, R. Rudell, and R. Bryant, "Efficient Implementation of a BDD Package," *Proc. Design Automation Conf.*, pp. 40-45, June 1990.
- [3] R. Brayton, G. Hachtel, C. McMullen, and A. Sangiovanni-Vincentelli, *Logic Minimization Algorithms for VLSI Synthesis*. Kluwer Academic, 1984.
- [4] R. Bryant, "Graph Based Algorithm for Boolean Function Manipulation," *IEEE Trans. Computers*, vol. 35, no. 8, pp. 667-691, Aug. 1986.

- [5] S.-C. Chang, D.I. Cheng, and M. Marek-Sadowska, "Minimizing ROBDD Size of Incompletely Specified Multiple Output Functions," *Proc. European Design and Test Conf.*, pp. 620-624, Mar. 1994.
- [6] O. Coudert, C. Berthet, and J.C. Madre, "Verification of Sequential Machines Using Functional Boolean Vectors," *Proc. IFIP Int'l Workshop, Applied Formal Methods for Correct VLSI Design*, Nov. 1989.
- [7] O. Coudert, C. Berthet, and J.C. Madre, "Verification of Synchronous Sequential Machines Based on Symbolic Execution," *Proc. Workshop Automatic Verification Methods for Finite State Systems*, pp. 365-373, June 1989.
- [8] A. Grasselli and F. Luccio, "A Method for Minimizing the Number of Internal States in Incompletely Specified Sequential Networks," *IRE Trans. Electronic Computers*, vol. 14, no. 3, pp. 350-359, June 1965.
- [9] K. Hirata, S. Shimozone, and A. Shinoara, "On the Hardness of Approximating the Minimum Consistent OBDD Problem," *Proc. Fifth Scandinavian Workshop Algorithm Theory*, July 1996.
- [10] T. Kam, T. Villa, R. Brayton, and A. Sangiovanni-Vincentelli, "A Fully Implicit Algorithm for Exact State Minimization," *Proc. Design Automation Conf.*, pp. 684-690, June 1994.
- [11] T. Kam, T. Villa, R. Brayton, and A. Sangiovanni-Vincentelli, *Synthesis of FSMs: Functional Optimization*. Kluwer Academic, 1997.
- [12] L. Lavagno, P. McGeer, A. Saldanha, and A.L. Sangiovanni-Vincentelli, "Timed Shannon Circuits: A Power-Efficient Design Style and Synthesis Tool," *Proc. 32nd Design Automation Conf.*, pp. 254-260, June 1995.
- [13] A. Oliveira, L. Carloni, T. Villa, and A. Sangiovanni-Vincentelli, "Exact Minimization of Binary Decision Diagrams Using Implicit Techniques," Technical Report No. UCB/ERL M96/16, Apr. 1996.
- [14] A.L. Oliveira, "Inductive Learning by Selection of Minimal Complexity Representations," PhD thesis, Univ. of California, Berkeley, Electronics Research Laboratory, College of Eng., Univ. of California, Berkeley, Dec. 1994. Memorandum No. UCB/ERL M94/97.
- [15] R. Ranjan, T. Shiple, and R. Hojati, "Exact Minimization of BDDs Using Don't Cares," EE290ls Project Report, May 1993.
- [16] M. Sauerhoff and I. Wegener, "On the Complexity of Minimizing the OBDD Size for Incompletely Specified Functions," Forschungsbericht Nr. 560, Universität Dortmund, 1994.
- [17] T. Shiple, R. Hojati, A. Sangiovanni-Vincentelli, and R. Brayton, "Heuristic Minimization of BDDs Using Don't Cares," *Proc. Design Automation Conf.*, pp. 225-231, June 1994.
- [18] Y. Takenaga and S. Yajima, "NP-Completeness of Minimum Binary Decision Diagram Identification," Technical Report COMP 92-99, Inst. of Electronics, Information, and Comm. Engineers (of Japan), Mar. 1993.
- [19] H. Touati, H. Savoj, B. Lin, R.K. Brayton, and A. Sangiovanni-Vincentelli, "Implicit State Enumeration of Finite State Machines Using BDD's," *Proc. Int'l Conf. Computer-Aided Design*, pp. 130-133, Nov. 1990.

Arlindo L. Oliveira received the BS (Eng) and the MS degrees in electrical and computer engineering from Lisbon Technical University, and the PhD degree in electrical engineering and computer science from the University of California, Berkeley, in 1994. He is currently an assistant professor at Lisbon Technical University and is also affiliated with INESC and the Lisbon Center of the Cadence European Laboratories. His research interests include logic synthesis, machine learning, combinatorial optimization, low power design, sequential systems optimization and automata theory.



Luca P. Carloni received the Laurea degree in electrical engineering, summa cum laude, from the University of Bologna, Italy, in July 1995, and the MS degree in electrical engineering and computer sciences from the University of California, Berkeley, in December 1997. He is currently a PhD candidate in the Department of Electrical Engineering and Computer Sciences of the University of California, Berkeley. His research interests are in the area of computer-aided design of electronic systems and include logic synthesis, high level synthesis, and combinatorial optimization.



Tiziano Villa studied mathematics at the Universities of Milano, Italy, Pisa, Italy, and Cambridge, U.K., and electrical engineering and computer science at the University of California, Berkeley, where he completed a PhD in electrical engineering and computer sciences in 1995. He worked in the integrated circuits division of the CSELT Labs, Torino Italy, as a computer-aided design specialist, and then, for many years, he was a research assistant at the Electronics Research Laboratory, University of California, Berkeley. In 1997, he joined the Parades Labs, Rome, Italy. His research interests include logic synthesis, formal verification, combinatorial optimization, automata theory, and hybrid systems. His contributions are mainly in the area of combinatorial and sequential logic synthesis. In May 1991, he was awarded the Tong Leong Lim Predoctoral Prize from the Electrical Engineering and Computer Sciences Department of the University of California, Berkeley. He is a coauthor of the books *Synthesis of FSMs: Functional Optimization* (Kluwer, 1997) and *Synthesis of FSMs: Logic Optimization* (Kluwer, 1997).



Alberto Sangiovanni-Vincentelli is a professor of electrical engineering and computer sciences at the University of California at Berkeley, where he has been a member of the faculty since 1976. He obtained an electrical engineering and computer science degree ("Dottore in Ingegneria") summa cum laude from the Politecnico di Milano, Milano, Italy, in 1971. He was an assistant professor from 1971 to 1974 and a "Professore Incaricato" (associate professor) from 1974 to 1976 at the Politecnico di Milano. In 1980-1981, he spent a year as a visiting scientist in the Mathematical Sciences Department of the IBM T.J. Watson Research Center. In 1987, he spent six months at MIT as a visiting professor. He has held a number of visiting professor positions at the University of Torino, University of Bologna, University of Pavia, University of Pisa, and University of Rome. He worked on several issues concerning the application of mathematical ideas and approaches to the design of electronic systems. Recently, he has been focusing on design methodologies and tools for mixed signal integrated circuits, including high-frequency and low power circuits, and for embedded controllers. In this latest area, he has focused his activity on methods for software synthesis and formal verification.

He is a co-founder of Cadence and Synopsys, the two leading companies in the area of Electronic Design Automation. He was a director of ViewLogic and Pie Design System and chair of the Technical Advisory Board of Synopsys. He is on the Board of Directors of Cadence, the founder of the Cadence Berkeley Laboratories and of the Cadence European laboratories, and a member of their Management Boards. He has consulted for a number of U.S., Japanese, and European companies. He is the scientific director of the Project on Advanced Research on Architectures and Design of Electronic Systems (PARADES), a European Group of Economic Interest funded by Cadence, Magneti-Marelli, SGS-Thomson, and the Italian National Research Council. He is on the Advisory Board of the Lester Center of the Haas School of Business and of the Center for Western European Studies and a member of the Berkeley Roundtable of the International Economy (BRIE).

He is an author of more than 450 papers and eight books in the area of design methodologies and tools. Dr. Sangiovanni-Vincentelli is a fellow of the IEEE and a member of the National Academy of Engineering.