An implicit formulation for exact BDD minimization of incompletely specified functions

Arlindo L. Oliveira
Cadence European Laboratories / IST-INESC
R. Alves Redol 9, 1000 Lisboa, Portugal, aml@inesc.pt

Luca P. Carloni, Tiziano Villa and Alberto Sangiovanni-Vincentelli
Department of EECS, University of California at Berkeley
Berkeley, CA 94720, USA, {lcarloni,villa,alberto}@eecs.berkeley.edu

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. We proved that this problem is NP-complete. Here we show that the BDD minimization problem can be formulated as a binate covering problem and solved using implicit enumeration techniques similar to the ones used in the reduction of incompletely specified finite state machines.

Keywords: Logic Synthesis, Binary Decision Diagrams, Finite State Machines.

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 (Bryant 1986) and the problem has a well defined solution.

This problem was proved NP-complete (Oliveira, Carloni, Villa & Vincentelli 1996) using Takenaga & Yajima's (1993) result that the problem of identification of the minimum BDD consistent with a set of minterms is NP-complete.

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) (Grasselli & Luccio 1965).

This similarity makes it possible to use implicit enumeration techniques developed for the purpose of ISFSM reduction (Kam, Villa, Brayton & Vincentelli 1994) to solve efficiently the BDD minimization problem. The manipulation of the characteristic functions of the sets of compatibles and prime compatibles, represented with ROBDDs (Brace, Rudell & Bryant 1990), allows the generation of very large sets that cannot be enumerated explicitly, as is demonstrated in the experiments.

The transformation presented in this paper and the algorithms developed for the solution are important for a variety of reasons. In inductive learning applications that use BDDs as the representation scheme (Oliveira & Vincentelli 1996), the accuracy of the inferred hypotheses is strongly dependent on the complexity of the result. The selection of the minimum BDD consistent with an incompletely specified function is important also in logic synthesis applications that use BDDs to derive gate-level implementations from a BDD, like *timed Shannon circuits*, DCVS trees and multiplexer based FPGAs.

Several heuristic algorithms have been proposed for this problem. The restrict (Coudert, Berthet & Madre 1989) and the constrain operators are two heuristics commonly used to assign the don't cares of a BDD. A comprehensive study of heuristic BDD minimization has been presented by Shiple, Hojati, Vincentelli & Brayton (1994).

An exact algorithm (Ranjan, Shiple & Hojati 1993) based on the enumeration of the different covers that can be obtained by all possible assignments of the don't care points has also been proposed. A pruning technique reduces the enumeration process thanks to a result 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. Although this pruning is performed implicitly, this method is exponential on the number of don't care points, and therefore is not applicable in problems of non-trivial size.

2 DEFINITIONS

We use the standard notation for BDDs. A BDD is a rooted, directed, acyclic graph where each node is labeled with the name of one variable. A BDD is called *reduced* if no two nodes exist that branch exactly in the same way and no redundant nodes exist. A BDD is ordered if there is an ordering of the variables such that, for all possible paths in the graph, the variables are always tested in that order.

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 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.

A 3 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, that correspond to the `zero`, `one` and `undefined` terminal nodes. A 3TBDD F corresponds to the incompletely specified function f that has all minterms in f_{off}, f_{on} and f_{dc} terminate in n_z, n_o and n_x, respectively.

3 THE COMPATIBILITY GRAPH

Previous algorithms (Ranjan et al. 1993) 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. The definition of the algorithm is based on the lemmas and definitions that follow. Due to space limitations, proofs for all the lemmas are omitted here and can be found in Oliveira et al. (1996).

Definition 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 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 $n_z \not\approx n_o$ 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 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 1 *If* $n_i \approx n_j$ *then* $n_i \sim n_j$.

The reverse implication of lemma 1 is not true, in general. Two nodes may be compatible but not CSC, as shown by the example presented by Oliveira et al. (1996). However, when two nodes belong to the same level, common support compatibility and compatibility are equivalent:

Lemma 2 *If* $\mathcal{L}(n_i) = \mathcal{L}(n_j)$ *then* $n_i \sim n_j \Rightarrow n_i \approx n_j$.

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 upwards, against the variable ordering of the BDD. On the other hand, if both nodes are common support compatible, then they can be replaced by a node that implements the completely specified function h referred to in Definition 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 3 *The nodes in the set* $s_i = \{n_1, n_2, \ldots, n_s\}$ *are common support compatible iff there exists a completely specified function* h *such that* $(h \sim n_j)_{j=1,\ldots,s}$ *and* $\mathcal{L}(h) \geq \mathcal{L}_{\max}(s_i)$.

Definition 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 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.*

Definition 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 three types of labels: e, t and l labels. The following two lemmas justify the algorithm by which graph G is built:

Lemma 4 *If* $\mathcal{L}(n_i) = \mathcal{L}(n_j)$ *then* $n_i \approx n_j \Rightarrow \left(n_i^{else} \approx n_j^{else} \wedge n_i^{then} \approx n_j^{then}\right)$.

Lemma 5 *If* $\mathcal{L}(n_i) < \mathcal{L}(n_j)$ *then* $n_i \approx n_j \Rightarrow \left(n_i^{else} \approx n_j \wedge n_i^{then} \approx n_j \wedge n_i^{else} \approx n_i^{then}\right)$.

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

Algorithm 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.)*
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 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.*

Figure 1 shows an example of the 3TBDD F obtained from f defined by the following sets: $f_{\mathrm{on}} = \{011, 111\}$, $f_{\mathrm{off}} = \{010, 110, 101\}$ and the corresponding compatibility graph.

The existence of an edge in the incompatibility graph is related with common support compatibility and with compatibility between pairs of nodes in the following way:

Lemma 6 $n_i \approx n_j \;\;\Rightarrow\;\; \exists e \in E \,\text{s.t.}\, e = (g_i, g_j) \;\;\Rightarrow\;\; n_i \sim n_j$.

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, as it is possible to have an edge between two nodes in G that are not CSC.

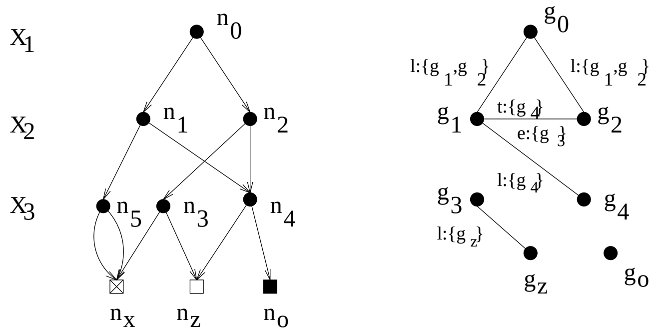

Figure 1 The 3TBDD F and the compatibility graph G. Nodes g_5 and g_x are not shown on the compatibility graph, since they are common support compatible with every node in the graph.

4 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 $s_i = \{g_{i_1}, g_{i_2}...g_{i_w}\}$ 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. Notice that for concision 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 6 *The e (t) class of s_i, $C_e(s_i)$ is the set of nodes that are in some e (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 7 *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 $\mathcal{L}(g_j) \neq \mathcal{L}(g_k)$*

Lemma 7 *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.*

Note that a clique of G that does not satisfy the condition in Lemma 7 is not necessarily a compatible set. 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 7. 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 8 *A set $S = \{s_1, s_2 ... s_n\}$ 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 : $\forall g_i, g_j \in s_k \; : (g_i, g_j) \in \mathrm{edges}(G)$*
3. *S is closed with respect to the e and t labels :*
   $\forall s_i \in S \, \exists s_j \in S \; : C_e(s_i) \subseteq s_j \; \bigwedge \; \forall s_i \in S \, \exists s_j \in S \; : C_t(s_i) \subseteq s_j$
4. *All sets in S are closed with respect to the l labels : $\forall s_i \in S \; : C_l(s_i) \subseteq s_i$*

5 GENERATION OF A MINIMUM BDD

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

Algorithm 2

1. *For each s_i in S, create a BDD node in R, r_i, at level $\mathcal{L}_{\max}(s_i)$.*
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 8 *R is an Ordered BDD compatible with F.*

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:

Theorem 1 *The BDD induced by a minimum closed cover for G is the BDD in $\mathcal{B}$ with minimum number of nodes.*

As an example, $S = \{\{g_0, g_1, g_2\}, \{g_4\}, \{g_3, g_5, g_z\}, \{g_o\}\}$ is a closed cover for the example depicted in Figure 1 and induces the BDD R shown on the right side of Figure 2.

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:

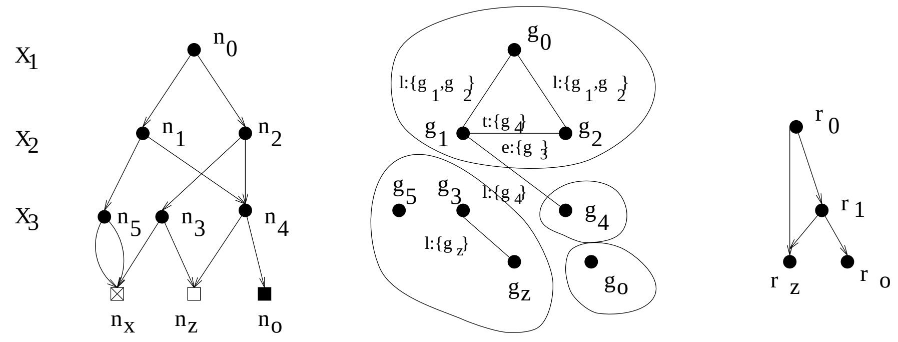

Figure 2 The 3TBDD F, the compatibility graph G and a solution R. Node g_5 was arbitrarily included in compatible $\{g_3, g_z, g_5\}$.

1. The definition of the e and t classes and the closure requirement in point 3 of Definition 8 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 8 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 8 simply eliminates some cliques of the compatibility graph from consideration and can be implemented by a filtering step. The transformation from BDD minimization to FSM reduction and its correctness are shown in Oliveira et al. (1996).

6 IMPLICIT COMPUTATION OF A MINIMUM CLOSED COVER

We will use the unified implicit framework proposed in (Kam et al. 1994). Implicit techniques are based on the idea of operating on discrete sets by their characteristic functions represented by (BDDs) (Bryant 1986).

To perform state minimization, one needs to represent and manipulate efficiently 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, $x = x_1 x_2 \ldots x_n$. 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 $s_2 s_3 s_5$ 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 $Tuple_{n,1}(x)$. 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 $\mathcal{R} : B^n \times B^m \rightarrow B$ 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) the next state relation $\mathcal{T}$, where 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.

It has been shown in Section 5 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: a) FSM compatibles that do not satisfy the L-closure are discarded, and b) FSM compatible closure is replaced by E-closure and T-closure. Our starting point is the fully implicit algorithm for exact state minimization reported by Kam et al. (1994), 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.

We follow Kam et al.'s (1994) approach to compute the set of compatibles $\mathcal{C}(c)$, where $\mathcal{C}(c) = 1$ 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 $\mathcal{C}(c)$ the compatibles c that are not closed with respect to their l-class. The l-class, $\mathcal{C}_l(c)$, 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 $\mathcal{L}(g_j) < \mathcal{L}(g_k)$. If $\mathcal{L}(g_j) < \mathcal{L}(g_k)$ then edge (g_j, g_k) has the l-label $\{g_j^{else}, g_j^{then}, g_k\}$.

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 $\mathcal{F}(c, i, n)$.

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

$$\mathcal{F}(c, i, n) \quad = \quad \exists p \; \{\exists c' \; [\mathcal{C}(c) \cdot Max\_Level(c, c') \cdot (c' \supseteq p)] \cdot \mathcal{T}(i, p, n)\} \qquad (1)$$

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.

7 RESULTS

Starting from the program ISM for implicit state minimization (Kam et al. 1994) we developed IMAGEM, a new program based on the theory described in this paper for exact BDD minimization.

To evaluate experimentally 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 minterms was designated as the *care set*, resulting in a set of incompletely specified Boolean functions g_i. The objective was to verify if the algorithm was able to identify a BDD no larger than the BDD for f_i, which represents a known upper bound on the solution. The second set of problems was obtained by selecting a subset of the problems that are distributed with *Espresso* (Brayton, Hachtel, McMullen & Vincentelli 1984), a well known two-level minimizer. 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, after eliminating all the functions that have a null don't care set.

Table 1 summarizes the results obtained from running these sets of examples. The last entry in the table is the example presented in this paper to illustrate the theory.

The first four columns report the original number of states, the number of compatibles, the number of compatibles after filtering (i.e. the ones which are closed with respect to their l-class) and the number of primes. The next two columns report the exact result obtained and the result obtained by the restrict operator (Coudert et al. 1989). 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 21600 seconds of CPU time and "spaceout" at 2Gb of memory.

8 CONCLUSIONS

This paper addresses the problem of binary decision diagram (BDD) minimization in the presence of don't care sets. We show that the minimum-sized

example	orig. states	compat.	filtered comp.	prime comp.	red. states	restrict	Imagem Cpu time
dnfa	64	2.4e+12	1332186	89	14	*16*	517.29
dnfb	36	4.8e+08	2987	94	6	*12*	11.85
dnfc	40	2.2e+08	2613	102	10	*15*	12.94
dnfd	93	1.1e+20	9.5e+08	-	-	*23*	timeout
dnfe	63	2.1e+13	141179	509	6	*12*	217.29
dnff	62	2.1e+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	3.3e+08	1329	170	13	*20*	13.98
xor7	57	2.7e+11	3076	640	15	*31*	88.16
xor8	94	1.5e+17	164929	21830	17	*45*	9041.11
alu1	95	1.0e+21	841993	1204	6	*6*	7409.97
br1	74	2.9e+18	799173	329	6	*11*	1313.91
br2	51	5.9e+14	53687	78	3	*8*	14.59
clpl	50	1.4e+13	7559	39	3	*13*	12.39
dc2	46	8.2e+10	8831	98	8	*12*	57.66
exp	54	2.6e+11	10638	25	3	*3*	31.34
exps	71	1.8e+10	3810	125	43	*44*	44.79
in0	151	2.6e+25	1.6e+06	1323	42	*44*	18201.76
in3	173	5.0e+39	587880	12	9	*14*	1755.21
inc	35	1.1e+07	364	26	12	*13*	3.84
intb	189	4.8e+46	3.8e+14	-	-	*69*	spaceout
mark1	71	7.4e+18	8049	35	4	*5*	41
newapla	52	1.2e+12	3252	33	10	*11*	41.5
newapla1	57	8.7e+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	3.8e+31	7.4e+12	-	-	*54*	spaceout
newcpla2	39	3.3e+08	477	68	10	*21*	5.72
newcwp	16	10367	106	10	6	*11*	0.39
newtpla	94	1.2e+23	411525	148	7	*23*	469.14
newtpla1	39	6.9e+09	1441	31	4	*5*	4.45
newtpla2	26	3.1e+06	158	9	9	*9*	0.9
newxcpla1	39	4.4e+09	1473	35	5	*10*	5.13
p82	16	15551	102	10	7	*7*	0.4
prom1	65	5.1e+09	382	77	50	*50*	30.04
prom2	33	2.1e+08	446	38	12	*12*	3.33
sex	28	1.6e+07	419	16	5	*5*	1.62
spla	155	1.6e+39	1.4e+12	-	-	*8*	spaceout
sqn	41	1.0e+07	173	43	19	*19*	9.13
t4	68	5.1e+14	31775	157	9	*11*	89.98
vg2	150	3.6e+36	4.0e+07	-	-	*14*	timeout
wim	14	4319	82	8	6	*6*	0.26
ex.paper	10	575	32	8	4	4	0.17

Table 1 Results obtained in the sets of problems studied.

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.

REFERENCES

Brace, K., Rudell, R. & Bryant, R. (1990). Efficient implementation of a BDD package, *The Proceedings of the Design Automation Conference*, pp. 40–45.

Brayton, R., Hachtel, G., McMullen, C. & Vincentelli, A. S. (1984). *Logic Minimization Algorithms for VLSI Synthesis*, Kluwer Academic Publishers.

Bryant, R. (1986). Graph based algorithm for Boolean function manipulation, *IEEE Transactions on Computers*, pp. C–35(8):667–691.

Coudert, O., Berthet, C. & Madre, J. C. (1989). Verification of synchronous sequential machines based on symbolic execution, *Proceedings of the Workshop on Automatic Verification Methods for Finite State Systems, vol. 407 of Lecture Notes in Computer Science* pp. 365–373.

Grasselli, A. & Luccio, F. (1965). A method for minimizing the number of internal states in incompletely specified sequential networks, *IRE Transactions on Electronic Computers* **EC-14**(3): 350–359.

Kam, T., Villa, T., Brayton, R. & Vincentelli, A. S. (1994). A fully implicit algorithm for exact state minimization, *The Proceedings of the Design Automation Conference*, pp. 684–690.

Oliveira, A. L., Carloni, L., Villa, T. & Vincentelli, A. S. (1996). Exact minimization of boolean decision diagrams using implicit techniques, *Technical Report M96/16*, UCB/ERL.

Oliveira, A. L. & Vincentelli, A. S. (1996). Using the minimum description length principle to infer reduced ordered decision graphs, *Machine Learning Journal* **25**: 23–50.

Ranjan, R., Shiple, T. & Hojati, R. (1993). Exact minimization of BDDs using don't cares. EE290ls Project Report.

Shiple, T., Hojati, R., Vincentelli, A. S. & Brayton, R. (1994). Heuristic minimization of BDDs using don't cares, *The Proceedings of the Design Automation Conference*, pp. 225–231.

Takenaga, Y. & Yajima, S. (1993). NP-completeness of minimum binary decision diagram identification, *Technical Report COMP 92-99*, Institute of Electronics, Information and Communication Engineers (of Japan).