Asia and South Pacific Design Automation Conference (ASP-DAC'99)
pp. 323-326, Hongkong, January 1999

Minimization of Free BDDs

Wolfgang Günther Rolf Drechsler

Institute of Computer Science, Albert-Ludwigs-University
79110 Freiburg im Breisgau, Germany
{guenther,drechsle}@informatik.uni-freiburg.de

Abstract— Free BDDs (FBDDs) are an extension of ordered BDDs (OBDDs). FBDDs may have different orderings along each path. They allow a more efficient representation, while keeping (nearly) all of the properties of OBDDs. In some cases even an exponential reduction can be observed. In this paper we present for the first time an exact algorithm for finding a minimal FBDD representation for a given Boolean function. To reduce the huge search space, it makes use of a pruning technique. The algorithm also considers symmetries of the function. Since the algorithm is only applicable to small functions we also present a heuristic for FBDD minimization starting from an OBDD. Our experiments show that in many cases significant improvements can be obtained.

I. Introduction

Binary Decision Diagrams (BDDs) were proposed for the representation and manipulation of Boolean functions. However, many problems like satisfiability are NP-complete for general BDDs, which makes function manipulation impractical. For this, BDDs without any structural restriction are of small use as a data structure in VLSI CAD. Therefore, in [4] *Ordered BDDs* (OBDDs) were introduced. OBDDs are canonical and thus allow to efficiently solve the satisfiability problem and to compare two functions. Furthermore, function manipulation becomes easy. In the meantime OBDDs have been successfully applied in many areas [6] and also have been integrated in industrial tools. OBDDs can be used for synthesis, and the resulting circuits have nice properties, like testability and low power.

However, for many functions the size of the OBDD representation is much larger than for general BDDs, and for many important functions, the OBDD sizes are exponential [5]. It is therefore desirable to have less restrictive representations which allow to handle a larger set of functions.

As one alternative *Free BDDs* (FBDDs) have been proposed. In FBDDs the ordering restriction is relaxed in the way that each variable is only allowed to appear once on each path (analogously to OBDDs), but the variables may appear in different orderings along each path. For some functions FBDDs allow an exponential reduction with respect to the number of nodes compared to OBDDs [5]. Furthermore, FBDDs (with some slight modifications) are still a canonical representation for Boolean function and allow efficient manipulation, like synthesis operations and satisfiability [9, 10].

Especially if BDDs are used in synthesis, the resulting size is very important. Furthermore, it has been observed by many authors that their methods proposed for OBDDs can be directly transferred to FBDDs (see e.g. [1]). Therefore, in the past first heuristics have been proposed [12, 2], but none could improve the results in comparison to OBDDs, i.e. the best sizes known for OBDDs could only be improved for some "hand-designed" examples.

In this paper, we present the first exact algorithm to determine a minimal representation as FBDD for a given Boolean function. It will be shown that it is possible to prune large parts of the huge search space by using techniques from exact OBDD minimization, including an effective branch&bound technique. Furthermore, symmetries of variables can be used to further prune the search space. Then we present a heuristic approach to FBDD minimization that can further optimize the best known OBDD size for (nearly) all considered examples. In many cases a significant improvement can be obtained.

II. Preliminaries

Boolean variables can assume values from $\mathbf{B} := \{0, 1\}$ and are denoted by Latin letters. In the following we consider Boolean functions $f : \mathbf{B}^n \to \mathbf{B}^m$ over the variable set $X_n = \{x_1, \ldots, x_n\}$. We also write $f = (f_1, \ldots, f_m)$ with $f_i : \mathbf{B}^n \to \mathbf{B}$ for $1 \leq i \leq m$.

As well-known, each Boolean function $f : \mathbf{B}^n \to \mathbf{B}$ can be represented by a *Binary Decision Diagram* (BDD) [4], i.e. a directed acyclic graph where a Shannon decomposition

$$f = \overline{x}_i f_{low} + x_i f_{high} \quad (1 \leq i \leq n)$$

is carried out in each node.

A BDD is called *free* (FBDD) if each variable is encountered at most once on each path from the root to a terminal node. A BDD is called *ordered* (OBDD) if it is free and if the variables are encountered in the same order on all such paths. For free and ordered BDDs the functions f_{low} and f_{high} correspond to the cofactors to $x_i = 0$ and $x_i = 1$, respectively. A BDD is called *reduced* if it neither contains nodes with isomorphic sub-graphs nor nodes with both edges pointing to the same node. In the following, only reduced BDDs are considered.

For OBDDs, a *level* is a set of nodes labeled with the same variable. This expression can also be used in case of FBDDs: a level of a free BDD is the set of nodes with an equal distance from the root. For an illustration see Figure 1. Note that in general an assignment of nodes to levels is not unique. For example in Figure 1, the node in level 2 labeled with x_4 could also belong to level 3.

The number of nodes of an FBDD for function f is denoted by $\#nodes(f)$. The number of nodes in level k is given by $\#nodes_k(f)$. (Analogously to OBDDs [3] complemented edges can be used for FBDDs. This is done in the following without mentioning it further. The methods presented below hold for both cases, i.e. with and without complemented edges.)

As the exact FBDD minimization algorithm is based on principles from OBDD minimization, these basic concepts are reviewed here. (For more details see [8, 7].)

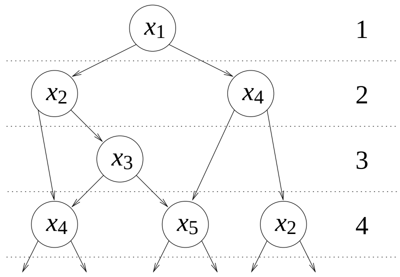

Fig. 1. Partitioning free BDDs to *levels*

Remark 1 We describe the theory for the case that the OBDD is built top-down (analogously to [7]), since this will also be necessary in the FBDD minimization algorithm below.

All previously presented algorithms for exact OBDD minimization are based on the following lemma from [8] although the basic algorithm has been improved in several ways.

Let π denote a permutation of the variables in X_n. If $x_i = \pi(k)$ for a variable x_i, then x_i is the kth element of the variable ordering π, i.e. x_i is in the kth *level* of the OBDD. For a set of variables $I \subseteq X_n$, let $\Pi(I)$ be the set of permutations π on X_n whose first $|I|$ members constitute I.

Lemma 1 [8] Let $f : \mathbf{B}^n \to \mathbf{B}^m$, $I \subseteq X_n$, $k = |I|$, and $x_i \in I$. Then there exists a constant c such that $\#nodes_k(f, \pi) = c$ for each $\pi \in \Pi(I)$ with $\pi(k) = x_i$.

More informally spoken, the lemma states that the number of nodes in a level k of an OBDD does not change if the variable ordering is changed above or below that level.

The lemma can be used as follows: For $I \subseteq X_n$, let

$$min\_cost_I = \min_{\pi \in \Pi(I)} \sum_{i=1}^{|I|} \#nodes_i(f, \pi),$$

and let π_I denote some permutation leading to that minimum. Assume for a fixed $I \subseteq X_n$ with $|I| = k$ that we know $min\_cost_{I'}$ for all $I' \subseteq I$ with $|I'| = k-1$. Then

$$min\_cost_I = \min_{x_i \in I}(min\_cost_{I \setminus \{x_i\}} + \#nodes_k(f, \pi_{x_i})),$$

with $\pi(k) = x_i$ and π_{x_i} being the permutation with

$$\pi_{x_i}(I \setminus \{x_i\}) = I \setminus \{x_i\}.$$

So the optimal ordering can be computed iteratively by computing for increasing k's min_cost_I for each k-element subset I, until $k = n$.

III. Symmetric Functions

For (partially) symmetric functions the minimization problem for FBDDs can be simplified extremely. A Boolean function $f : \mathbf{B}^n \to \mathbf{B}$ is said to be *symmetric* with respect to a set $S \subseteq X_n$ if f remains invariant under all permutations of the variables in S. For completely specified functions the symmetry is an equivalence relation which partitions the set X_n into disjoint classes $S_1, \ldots, S_k$ that will be named the *symmetry sets.* A function f is called *partially symmetric* if it has at least one symmetry set S_i with $|S_i| > 1$. If a function f has only one symmetry set $S = X_n$, then f is called *totally symmetric*. For totally symmetric functions the optimal FBDD can be directly determined:

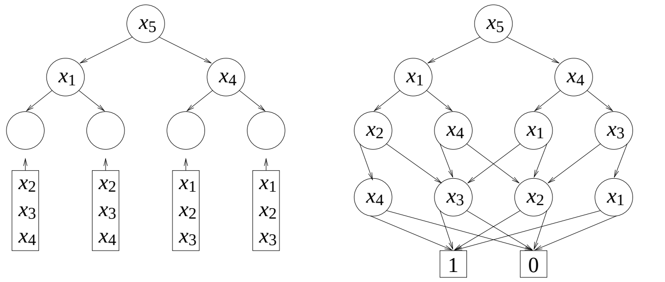

Fig. 2. Partial FBDD and optimal FBDD for HWB_5

Lemma 2 Let f be a totally symmetric function. Then the optimal FBDD for f and the optimal OBDD for f are isomorphic.

The proof directly follows from [13]. For partially symmetric functions, the following lemma holds.

Lemma 3 Let $f : \mathbf{B}^n \to \mathbf{B}^m$ be a partially symmetric Boolean function and let S be a symmetry group ($|S| > 1$) of f. Let $\prec$ define an order on the variables in S. Then there exists an optimal FBDD where the variables in S are ordered on all paths with respect to $\prec$.

IV. Exact FBDD Minimization

Because of Lemma 1, the number of OBDD nodes in a level k only depends on the set of variables I that are above that level in the variable ordering. A similar property holds for FBDDs. Variables above a level k can be exchanged without altering the number of nodes in that level (as far as the read-once property still holds). The set I of variables in OBDD minimization can be expressed by a set C of cofactors in case of FBDD minimization. However, this set can have exponential size in contrast to the linear size of I.

Example 1 Let $\mathrm{wt} : \mathbf{B}^n \to \{0, \ldots, n\}$ given by

$$\mathrm{wt}(x_1, \ldots, x_n) = |\{x_i : 1 \leq i \leq n \text{ and } x_i = 1\}|$$

be the number of variables that have value one. Then the *Hidden Weighted Bit function* (HWB) [5] is defined as:

$$\mathrm{HWB}_n(x) = \begin{cases} 0, & \mathrm{wt}(x) = 0 \\ x_{\mathrm{wt}(x)}, & \mathrm{wt}(x) > 0 \end{cases}$$

(For this function, an exponential lower bound of the OBDD representation is known [5].) An example situation for an FBDD for HWB_5 is given in Figure 2. The FBDD F is incomplete, and level 3 has to be constructed. The remaining cofactor nodes that are still to be represented are given by empty circles. The list of variables in the support of each cofactor is given in the rectangles below these circles. In the next step, all of the $3^4 = 81$ different assignments of possible variables to the cofactor nodes have to be examined. One of these assignments leads to the optimal FBDD for HWB_5 which is given in Figure 2.

A. Algorithm

In this section, the main algorithm is described. A sketch is given in Figure 3. Initially, the functions that are to be represented are given as an OBDD. For each level k, a hash table stores entries including a partial FBDD

1 compute_minimal_FBDD$(f_1, \ldots, f_m)$
2   insert $(F = \emptyset, C_\emptyset = \{f_1, \ldots, f_m\}, \mathit{min\_cost}_\emptyset = 0)$ into *table*;
3   for $(k = 1; k \leq n; k++)$
4   k: level under consideration; *next_table*$= \emptyset$;
5   for (each $(F, C_F, \mathit{min\_cost}_F) \in \mathit{table}$)
6   for (each $c \in C_F$) get list L_c of possible variables for c;
7   for (each assignment α of variables from L_c to c)
8                 $F' = F + \alpha$;
9   compute $C_{F'}$;
10                 $\mathit{cost} = \mathit{min\_cost}_F + |C_{F'}| -$ number of dummy variables in level $k - 1$;
11   if $(C_{F'} \notin \mathit{next\_table}$ or $\mathit{cost} < \mathit{min\_cost}_{F'})$
12   store $(F', C_{F'}, \mathit{cost})$ into *next_table*;
13   *table* = *next_table*;
14   return best FBDD F;

Fig. 3. Sketch of the algorithm

F, a set of cofactors C_F, and the cost so far, i.e. number of nodes in levels 1 to k.

The algorithm starts with a table with only one entry. This entry consists of the empty FBDD, the set of cofactors with respect to the empty FBDD, which is the set of original functions $f_1, \ldots, f_m$, and the cost 0. Then, the iteration over all levels starts. In each level, the hash table *next_table* for the next level is computed. This is done by considering all elements of *table*.

For each element $(F, C_F, \mathit{min\_cost}_F)$ in *table*, all possible extensions to F have to be computed and added to *next_table*. This concept is analogous to the OBDD minimization algorithm from Section II. But while for OBDD minimization, all remaining variables iteratively had to be brought to the considered level k, for FBDD minimization this has to be done separately for each cofactor in C_F. Therefore, for each cofactor c the set of possible variables L_c is determined in Line 6. In principle, this is the set of all variables which do not appear on any path from a root of the FBDD to the node representing cofactor c, but it can be reduced by using only those variables which are in the support of c. However, to allow later reconvergencies in the FBDD, a dummy variable must be inserted if this set is smaller than the original set of possible variables (which always has size $n - k + 1$). L_c can be further reduced by taking advantage of symmetries. If two variables in L_c belong to the same symmetry group, the one with the larger index can be deleted (see Lemma 3).

For each possible assignment of variables to cofactors (this number is usually exponential), the new partial FBDD F' and the new set of cofactors $C_{F'}$ are computed (Lines 8 and 9). The new cost is computed from the old cost, but has to respect that dummy variables have cost 0, since they do not appear in the final FBDD (Line 10). The new partial FBDD F' is added to *next_table* if it does not already exist (i.e. the resulting set of cofactors $C_{F'}$ is not yet in *next_table*) or if the cost is less than the cost of the existing entry with the same set $C_{F'}$. Since all possible cases are considered it follows, that the algorithm in Figure 3 computes the optimal FBDD.

B. Pruning Technique

The effectiveness of the algorithm can be further improved by using a pruning technique: As initial upper bound, the shared size of the OBDD is used, since the optimum FBDD has always less or equal size than the initial OBDD. Throughout the algorithm, the OBDD's size can be used for those parts of the FBDD that are not yet known: The actual upper bound thus is the sum of the already known costs, i.e. the number of nodes in the upper levels, and the shared size of the cofactors that are still to be represented.

We used the actual cost plus the number of cofactors that are to be represented in the next step as lower bound. Note that this is the same lower bound as used in [7] for OBDD minimization.

V. Heuristic

For larger functions the complete algorithm discussed above cannot be applied, due to the huge number of cofactors. For this we make use of a "restricted version" of the exact algorithm in the following that gives us a simple, but powerful heuristic: Instead of considering all nodes of a given level simultaneously, they are considered one after another. Again, the FBDD is built top down, as in the exact algorithm.

While building the FBDD, the set of nodes (cofactors) which still have to be realized is stored in a queue. In the beginning, there are m nodes in the queue to realize a representation for the given function f. Then, the node v of the queue's head is considered. First, a variable x_i for node v is chosen heuristically, which will be described below. As the node finally has to represent a given function g, the cofactor $g_{x_i=0}$ is computed. If there is not yet a node in the queue which represents $g_{x_i=0}$, then such a node is added to the queue. The other cofactor $g_{x_i=1}$ is handled simultaneously. The algorithm terminates if the queue is empty.

The method to choose a variable x_i for the node representing function g works as follows: Variable x_i is the variable of the support of g that minimizes the sharing size of the OBDD representation of $g_{x_i=0}$, $g_{x_i=1}$ and the functions for the other nodes of the queue. By considering only variables from the support of g it is guaranteed that the algorithm terminates after a finite number of steps.

VI. Experimental Results

The algorithm described above has been implemented in *C*. All experiments have been carried out on a *SUN Ultra 1* with 128 MByte of main memory. All runtimes are given in CPU seconds. We used the CUDD package [11] as underlying OBDD package. All numbers given use OBDDs and FBDDs with complemented edges.

In a first series of experiments we demonstrate the effect of our pruning technique. For this, we compared our exact FBDD algorithm to the OBDD sizes for HWB (see Section IV). Results are given in Table I. In column "in", the number of inputs is given. The size of the optimal OBDD and the optimal FBDD are given in column "OBDD" and

TABLE I
RESULTS FOR THE HIDDEN WEIGHTED BIT FUNCTION

			without pruning		pruning	
in	OBDD	FBDD	time	MB	time	MB
4	8	7	0.1	1	0.1	1
5	13	12	6.4	1	0.2	1
6	19	18	—	> 1000	9.0	5

TABLE II
RESULTS FOR SEVERAL BENCHMARKS

circuit	in/out	OBDD	FBDD	time	MB
adder2	4/ 2	6	6	0.1	1
adder3	6/ 3	11	11	0.2	1
adder4	8/ 4	16	16	42.6	7
mult2	4/ 4	12	10	0.1	1
bcd.div3	4/ 4	14	14	0.2	1
c17	5/ 2	7	7	0.1	1
cm138a	6/ 8	18	18	118.3	45
cm42a	4/ 10	20	20	27.0	12
cm82a	5/ 3	12	12	0.1	1
dc1	4/ 7	22	21	4.0	3
dekoder	4/ 7	19	19	2.3	2
majority	5/ 1	8	8	0.1	1
newapla2	6/ 7	17	17	51.9	10
newbyte	5/ 8	17	17	31.6	19
newcwp	4/ 5	9	9	0.1	1
rd53	5/ 3	17	17	0.1	1
wim	4/ 7	20	20	5.7	3
xor5	5/ 1	6	6	0.1	1

"FBDD", respectively. Time and space usage for the algorithm without any pruning technique and with the pruning technique described in Section B are given in the last four columns. This first experiment clearly demonstrates the advantage of pruning.

In a second series of experiments, we compared our exact algorithm to the OBDD sizes for several benchmark functions from LGSynth91. By making use of pruning we are able to find the optimal FBDD for functions with up to 8 variables within a few seconds, while a complete enumeration fails for more than 5 variables (see Table II). In column "circuit", the name of the circuit is given, and its number of primary inputs and outputs are given in the next column. Again, the sizes for the best OBDDs and FBDDs are given in columns "OBDD" and "FBDD", respectively. Time and space usage for the algorithm are given in the last two columns. Our experiments show for the first time exact results for FBDDs on benchmark circuits. It turns out that for most of these small functions the optimal FBDD size is identical to the optimal OBDD size.

But the situation changes completely for larger functions: In a third series of experiments, we studied the quality of the heuristic presented in Section V. The results are given in Table III. The OBDD sizes are given for the best known ordering taken from [11]. The FBDD is built starting with that ordering. As can be seen the sizes could be reduced in almost all cases within small runtimes. This is the first time that for benchmark circuits FBDDs could be constructed that are smaller in size than the best known OBDDs. The reduction is in some cases even more than 10%, while keeping all the properties of OBDDs, like efficient manipulation, etc.

VII. CONCLUSIONS

In this paper we presented minimization techniques for free BDDs. For the first time an exact algorithm for FBDD optimization has been proposed that allows to handle functions with up to 8 variables due to a clever pruning technique. For larger functions we proposed a new heuristic that computes FBDDs that are smaller in size than the best known OBDDs.

TABLE III
HEURISTIC RESULTS

circuit	in/out	OBDD	FBDD	time	MB
apex1	45/ 45	1246	1178	7.3	1
apex6	135/ 99	498	465	0.8	1
c1908	33/ 25	5526	5103	232.2	4
c2670	233/140	1850	1815	48.7	2
c3540	50/ 22	23828	21000	3824.8	6
c432	36/ 7	1064	1063	17.3	1
c499	41/ 32	25866	25866	8431.4	17
c5315	178/123	1719	1594	25.1	1
c7552	207/108	2213	2092	143.3	3
c880	60/ 26	4053	3575	143.2	2
dalu	75/ 16	689	666	4.3	1
des	256/245	2945	2941	36.5	1
frg2	143/139	963	920	6.8	1
i10	257/224	20660	18814	6031.6	11
i3	132/ 6	133	133	0.5	1
i8	133/ 81	1276	1211	6.7	1
too_large	38/ 3	319	313	1.6	2

REFERENCES

[1] P. Ashar and S. Malik. Fast functional simulation using branching programs. In *Int'l Conf. on CAD*, pages 408–412, 1995.

[2] J. Bern, C. Meinel, and A. Slobodová. Some heuristics for generating tree-like FBDD types. *IEEE Trans. on CAD*, 15:127–130, 1996.

[3] K.S. Brace, R.L. Rudell, and R.E. Bryant. Efficient implementation of a BDD package. In *Design Automation Conf.*, pages 40–45, 1990.

[4] R.E. Bryant. Graph - based algorithms for Boolean function manipulation. *IEEE Trans. on Comp.*, 35(8):677–691, 1986.

[5] R.E. Bryant. On the complexity of VLSI implementations and graph representations of Boolean functions with application to integer multiplication. *IEEE Trans. on Comp.*, 40:205–213, 1991.

[6] R. Drechsler and B. Becker. *Binary Decision Diagrams - Theory and Implementation*. Kluwer Academic Publishers, 1998.

[7] R. Drechsler, N. Drechsler, and W. Günther. Fast exact minimization of BDDs. In *Design Automation Conf.*, pages 200–205, 1998.

[8] S.J. Friedman and K.J. Supowit. Finding the optimal variable ordering for binary decision diagrams. In *Design Automation Conf.*, pages 348–356, 1987.

[9] J. Gergov and C. Meinel. Efficient analysis and manipulation of OBDDs can be extended to FBDDs. *IEEE Trans. on Comp.*, 43:1197–1209, 1994.

[10] D. Sieling and I. Wegener. Graph driven BDDs - a new data structure for Boolean functions. *Theoretical Computer Science*, 141:283–310, 1995.

[11] F. Somenzi. *CUDD: CU Decision Diagram Package Release 2.2.0.* University of Colorado at Boulder, 1998.

[12] M. Tachibana. Heuristic algorithms for FBDD node minimization with application to pass-transistor-logic and DCVS synthesis. In *SASIMI*, 1996.

[13] I. Wegener. Optimal decision trees and one-time-only branching programs for symmetric Boolean functions. *Information and Control*, 62:129–143, 1984.