TECHNIQUES FOR FORMAL TRANSFORMATIONS OF BINARY DECISION DIAGRAMS[1]

G. Kolotov, I. Levin, V. Ostrovsky

School of Engineering, Tel-Aviv University
School of Engineering, Bar-Ilan University

ABSTRACT

Binary Decision Diagrams (BDDs), when used for representation of discrete functions, permit the direct technology mapping into multi-level logic networks. Complexity of a network derived from a BDD is expressed by its number of non-terminal nodes.
This paper discusses the problem of reducing the BDDs. The paper has two main contributions: a) bounds of potential complexity of the BDD have been determined and proven; b) a formal technique is presented for simplification of Boolean operations on a set of BDDs.

1. INTRODUCTION

The BDD is a graphic form of depicting the Shannon's expansion of a logic function. The concept of BDDs was first proposed by Lee [5] in 1959. It was then developed into a useful data structure by Akers [1] and subsequently by Bryant [3], who introduced a concept of *reduced, ordered* BDDs (ROBDDs), along with a set of efficient operators for their manipulation, and proved the canonicity property of ROBDDs. H.R. Andersen in his work [2] shows how to implement the BDD on a computer. Below, we summarize the most important points of their work.
Our paper deals with reducing the BDD size of a given function by splitting it into its linear and non-linear parts. The linear part of the function comprises the linear members of a Reed-Muller polynomial of the function. The linear part is defined by a linear transformation over the input variables while the non-linear part is an implementation of the rest of the function [4]. When viewed as a black box, the two parts comprise the original function. It is our task to choose the linear transformation in such a way that the complete implementation will be smaller in size than the original (smallest of all possible implementation, if at all possible).
Most of known algorithms that allow choosing the linear transformation result in permutation of the input variables. In some cases this provides suitable results. In other cases the results are far from optimal. The method used in our work is based on using the autocorrelation function [4] as a criterion for optimization of the transformation.
Our work concerns reducing the size of the BDDs. Needless to say, an ability to perform the required transformations upon the diagram itself is one of the central issues. In this paper we show a technique for formal transformation on the set of BDD. The proposed technique can be considered a first step on the way of creating an algebra of BDDs.
Another significant issue of our paper is the upper bound of the BDD size. We estimate the upper bound and a constructive way to build "the worst case" BDD.
The paper is organized as follows. Definitions are presented in section 2. Estimation of the upper bound of the BDD complexity is presented in section 3. Section 4 is dedicated to a formal technique for manipulation on the set of BDDs. Conclusions are presented in section 5.

2. DEFINITIONS

2.1. Properties of a BDD

The following properties determine the BDD of any function.

- Each node is defined by its variable and the two edges: T-edge that is followed when the value of the variable is 1 and F-edge that is followed otherwise.
- There are no redundant nodes: the T-edge and the F-edge should point to different nodes.
- All edges go strictly "downwards" in the given ordering of the variables.
- All nodes are unique.

As follows immediately from the above properties, the diagram is unique once the ordering of the variables has been determined. The ordering of the variables may significantly affect the size of the diagram, since the importance of the variables in the function and the impact

[1] This research was supported by BSF under grant No. 2000154.

on choosing their values is not uniform. Figure 1 shows an example of a BDD.

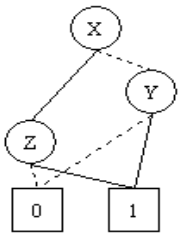

Figure 1: BDD of a 3-variable function: $F(X,Y,Z)=(X+Y)(\overline{X}+Z)$

2.2. Data Structure for BDD Representation

The simplest way of computer representation a BDD is in an array of structures. Each structure hold s the data of one node: the variable in Shannon's expansion, the index in the array where the T-edge points and the index where the F-edge points.

As usual, it is possible to speed up the operations by expending some memory. Some of the algorithms require the list of all nodes associated with a given variable. If prepared in advance (during the building of the diagram) such lists improve the running time of the algorithms.

3. THE MAXIMAL SIZE OF BDD

In what follows, we find the upper bound on the size of a BDD of any function of N variables. As we will show, this is, in fact, not a bound but a reachable maximal size.

3.1. The Upper Bound of a BDD Size

The maximal size of the BDD is the sum over the maximal sizes of its rows. The maximal number of nodes in the row for X_K, with $0 \le K < N$ is:

$$a_k = \min\left(2^k, 2^{2^{N-k}} - 2^{2^{N-k-1}}\right)$$

The maximal number $\Phi_{\max}$ of nodes in a BDD with N variables is, therefore:

$$\Phi_{\max} = \sum_{K=0}^{N-1} \min\left(2^K, 2^{2^{N-K}} - 2^{2^{N-K-1}}\right)$$

To understand the meaning of this expression, take a look at Figure 2. It shows the fall of the ratio between the maximal BDD size and that of a Truth Table of the same function.

N	1	4	5	8	10	12	15
BDD	1	9	17	77	269	765	4349
TT	2	16	32	256	1024	4096	32768
%	50	56	53	30	26	19	13

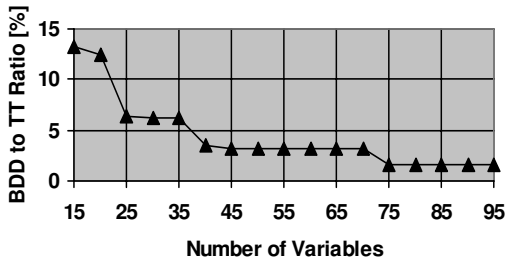

Figure 2. The upper bound on the BDD size as a function of the number of variables, relative to the size of the Truth Table.

3.2. The Proof by Construction

We will prove the above expression by constructing a BDD of the maximal possible size.

The number of nodes in each row of the BDD is limited by the two following values. The first is the total number of the nodes above it, because only they can point to the nodes in the current row. The second is the total number of the nodes below it, because the nodes in the current row can only point to them.

Since there are only two leaf nodes (0 and 1), the BDD is limited by a diamond-like shape. One point is root and the other (albeit cut off) the two leaves.

As the diagram grows from above, we try to create the largest possible number of nodes in each row. The row of X_0 can only hold 1 node. The row of X_1 can hold up to 2 nodes; that of X_2 up to 4 nodes; et cetera. The row of X_K can hold up to 2^K nodes, and will hold less if at least one node of the row of X_{K-1} will point to nodes in the rows below X_K.

Since there are only two leaf nodes, there can be only two nodes for X_{N-1} – the one pointing to 1 with T-edge and to 0 with F-edge, the other its exact opposite.

The maximal number of nodes for X_{N-2} is 12. To realize this, take a look at the definition of the BDD. Each node is uniquely identified by its variable (X_{N-2}) and the nodes it points to. Each node has two edges. There are 4 nodes below the X_{N-2} row. This means that there are exactly 12 distinct possibilities to create a node.

Recursively, each node of the current row must have a unique combination of 2 different values, each in the range [0, α), where α is the number of nodes below the current row. In other words, if C_K is the number of nodes in K^{th} row (this time, K is counted from below), and S_K is the

sum of all rows up to K^{th}, we arrive at the following recursive formula:

$$\forall k \in [-1, N-2],\ s_k = \sum_{i=0}^{k} c_k$$

$$c_{k+1} = s_k \cdot (s_k - 1) = s_k^2 - s_k$$

This recursion, when solved, yields:

$$s_k = 2^{2^K},\ c_k = 2^{2^K}\left(2^{2^K} - 1\right)$$

By shifting the index (so that it runs from 0), reversing it (so that 0 corresponds to the root row) and summing over the rows, we obtain the desired result.

All that remains now is to understand what happens at the row where the switch in the minimum occurs. That is to say, the BDD grows from above and shrinks towards the leaves, so there should be a row somewhere where the expression that determines the minimum is switched.

Let's look at an example of N=6 variables. There is 1 root node (level 0), 2 at level 1, 4 at level 2 and, finally, 8 at level 3. If we were building a Binary Decision Tree instead of a diagram, level 4 would have held 16 nodes. However, because we are limited by the requirement of the uniqueness of BDD nodes, this is where the switch occurs. Level 5 has 2 nodes and therefore level 4 can only have 12 nodes, not 16. Counted altogether, we have a BDD with 29 non-terminal nodes.

The last question to answer is this: is it possible to have a row with 2^K nodes and a row below it with much less than 2^{K+1} nodes? The answer is, of course, that it is and the construction holds. The reasoning is simple: if the row K+1 is the switch-row, the number of nodes in it and below it is S_{K+1}, which allows us to have much more nodes than required in the row K.

Figure 3 shows an example of a full-BDD function.

4. OPERATION ON BDDS

Just as the case is with Boolean functions, it is possible to combine any two BDDs with any kind of a binary, Boolean operator. The procedure can also be expanded to combine any number of BDDs. As shown by H.R. Andersen [2], this requires the number of operations proportional to the product of the sizes of the BDDs involved. This is clearly dual to combining the analytical (SOP) expressions, where the complexity is proportional to the product of the numbers of terms.

In 4.1 we give examples of operations that are significantly simplified by the BDDs. In 4.2 and 4.3 we show some special cases for which the complexity of the combining can be drastically reduced.

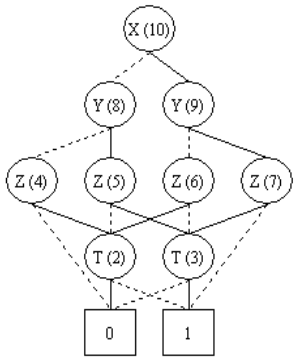

Figure 3. Function of 4 variables with maximal-size BDD.

4.1. Negation of a Function and of a Variable

To negate a function, one only has to swap between its 0 and 1 leaves. In practice, this means going over the nodes list and set every 0-pointing edge to 1, and every 1-pointing edge to 0.

To negate a variable, one has to swap between the T-edge and the F-edge of all its nodes.

A BDD can be implemented using logical gates. The most conventional way to do so is to use multiplexers. When testing a digital circuit, the engineer is required to generate faults and analyze the results. These faults can be easily tested with a BDD.

To produce the most standard of these faults - a 'stuck at 0' (or 'stuck at 1') – we replace the required node with either its T-Edge or its F-Edge.

Needless to say, the complexity of all the operations described until now is linear in the size of the input BDD, and can be further reduced if the lists of the per-variable nodes are kept (see 2.2).

4.2. Product and Sum of Disjoint BDDs

We call two Boolean functions (and their BDDs) "disjoint" when they have no common input variables. To combine them using a binary operator, such as AND, OR, XOR etc., one has to work with the Boolean representation of the operator (Truth Table, BDD, etc.) and replace the appropriate terminal nodes of one BDD with a copy of another (or with its negation).

For example, to create a BDD of product of the two input BDDs, one has to locate all 1-pointing edges of the first and change them so that they point to the root of the second.

The complexity of this operation is, of course, linear in the sum of the sizes of the input BDDs.

In order for the resulting BDD to be optimal (minimal number of nodes), one has to follow two simple rules. First, choose the ordering of the input BDDs so that the

larger (or its negation) is not used twice. Second, do not separate or mix the two sets of the input variables. The function's value cannot be determined until we exhaust all its BDD nodes. Thus, keeping them separate may result in increasing the size of the resulting BDD, and certainly will not decrease it.
This operation can be expanded to a set of disjoint functions, but creating an optimal BDD of the result is trickier and depends on the combining function.

4.3. Product and Sum of two XOR BDDs

We call a BDD of the parity-check function a XOR BDD. An example is given in Figure 4 below. When combining two such functions with an AND or an OR operator, the complexity is reduced to the sum of the BDD sizes, even if their input sets are not disjoint.

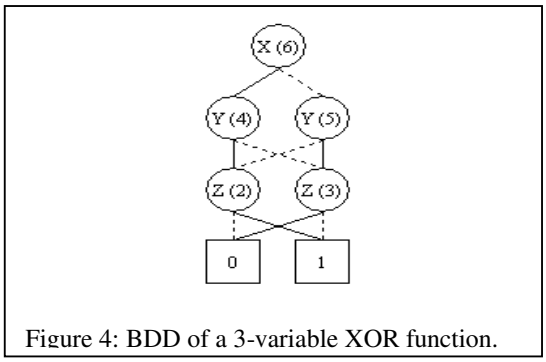

Figure 4: BDD of a 3-variable XOR function.

Suppose we want to compute $F(S_1)F(S_2)$, where F is a XOR of all the variables in the input set, and S_1 and S_2 are two sets of variables. Let's denote C the set of the variables common to S_1 and S_2, R_1 the remainder of S_1 when all the common variables are removed and R_2 the same for S_2. It is easy to show that the result can be calculated as follows:

$$F(S_1)F(S_2) = F(C)\overline{F(R_1)}\overline{F(R_2)} + \overline{F(C)}F(R_1)F(R_2)$$

Therefore, it is immediately obvious that the BDD of the result can be created from the BDDs of the parts. In order to create the BDD of minimal possible size, one has to follow the rules of 4.3.
The number of nodes in the resultant BDD is equal to the number of nodes in the BDD of the largest set, plus twice the number of nodes in the other sets. In the example of the Figure 5: $|S_1|=|S_2|=4$. The number of operations in a straightforward application of the AND operator would be proportional to $(2*4-1)^2=49$. However, since $|C|=3$, $|R_1|=|R_2|=1$, the number of operations in our construction is proportional to $(2*3-1)+1*2*2=9$.
Finally, we note that if any of the sets is empty (one set of variables is included in the other, or disjoint sets), we can further simplify the construction by substituting 0 instead of the XOR of this set.

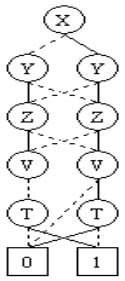

Figure 5: Correctly assembled BDD of a product of two XOR BDDs. S_1={X, Y, Z, V} and S_2={X, Y, V,T}

5. SUMMARY

In this paper, we have presented a novel technique of formal transformation of Boolean operations on the set of BDDs. This technique allows improving the linear transformation of input variables of the function to be implemented.
We also estimate the upper bound of the size of a BDD and show a way of constructing the BDD having the maximum size.
The first of the above-mentioned results opens a way for developing an algebra of BDDs, while the second result helps to estimate a potential complexity of a BDD without performing any optimization procedure.

6. REFERENCES

[1] S. B. Akers, "Functional testing with binary decision diagrams," in*: Eighth Annual Conf. Fault-Tolerant Computing, 1978, pp.75–82.*

[2] Henrik Reif Andersen. "An Introduction to Binary Decision Diagrams", *Lecture notes for 49285 Advanced Algorithms E97, October 1997. Department of Information Technology, Technical University of Denmark.*

[3] R. Bryant, "Graph-based algorithms for Boolean function manipulation," *IEEE Trans. Computers, vol. 35, pp.677–691, Aug.1986.*

[4] M. Karpovsky, R. Stankovic, J. Astola, "Reduction of Sizes of Decision Diagrams by Autocorrelation Functions", *IEEE TRANSACTIONS ON COMPUTERS,VOL.52, NO.5, MAY 2003.*

[5] C.Y. Lee, "Representation of switching circuits by binary decision programs," *Bell System Techn .J.* vol. 38, no.4, pp.985–999, June 1959.