Computer and Information Science
www.ccsenet.org/journal.html
Vol. 1, No. 2
May 2008

# An Efficient Method for Generating Optimal OBDD of Boolean Functions

Ashutosh Kumar Singh (Corresponding Author)
School of Engineering and Science
Curtin University of Technology
Saarwak Campus, CDT, 250, Miri, Malaysia
Tel: 60-85-443-939 E-mail: ashutosh.s@curtin.edu.my

Anand Mohan
Centre for Research in Microprocessor Applications (CRMA)
Department of Electronics Engineering, Institute of Technology
Banaras Hindu University
Varanasi- 221 005, India
Tel: 90-542-257-5272 E-mail: amohan@bhu.ac.in

**Abstract**

An efficient method of finding optimal (OBDD) of an *n* variable Boolean function is presented that offers a simple and straightforward procedure for optimal OBDD generation along with storage economy. This is achieved by generating *n*! fold tables and applying node reduction rules to each fold table directly instead of generating all *n*! OBDDs of the function.

**Keywords:** Fold table, Binary decision diagrams, Formal verification

## 1. Introduction

The pioneering work of Akers (1978) on graphical representation of Boolean functions using Binary Decision Diagrams (BDDs) offered an attractive and convenient technique for simplification and manipulation of complex Boolean functions. Modification of these BDDs was suggested by Bryant (1986) and since then different types of Decision Diagrams (DDs) have been introduced by researchers (see the book written by Sasao T. and Fujita M. 1996 for detail). The graphical representation of Boolean functions using BDDs have been potentially used for simplification of complex functions (Drecher R., Dreshler N. and Gunther W. 2000; Hong Y., et. al 2000; Scholl C. et. al 2000). As a result, BDDs and its variations are being extensively used in logic design, synthesis and testing of digital circuits (Jabir A. M., Pradhan D. K., Singh A. K. Rajaprabhu T. L. 2007; Minato S. 1996; Lai Y. T., Pedram M. and Vrudhula S. B. K. 1996; Gergov J. and Meinel C. 1994; Shen A., Devadas S. and Ghosh A. 1995). However, minimization of number of BDD nodes has been of focal interest in several applications such as formal verification. Ordered Binary Decision Diagram (OBDD) is an important form of decision diagram generated by imposing ordering relation among function variables such that the resulting form is canonical and it provides more compact representation of Boolean functions (Litan L. H. and Molitor P. 2000; Wang Y., Abd-el-Barr M. and McCrosky C. 1997; Wegener I. 1994; Bryant R. E. 1992; Liaw H. T. and Lin C. S. 1992; Friedman S. J. and Supowit K. J. 1990; Bern J., Meinel C. and Slobodova A. 1996). OBDD representation along with the use of the data structures for caching intermediate computations provides a way for the efficient implementation of many Boolean operations. However, one major drawback of OBDD representation is that identification of optimal (minimal node) OBDD requires generating all *n*! OBDDs of *n* variable function. The ordering of the function variables corresponding to optimal OBDD is called optimal ordering and for which many techniques are already reported (Wegener I. 1994; Bryant R. E. 1992; Bern J., Meinel C. and Slobodova A. 1996; Drechslor R., Becker B. and Gockel N. 1996) but they are inefficient in respect of computational complexity and storage requirements.

This paper describes a new efficient method for identification of optimal OBDD of a Boolean function without generating all *n*! OBDDs. The proposed method uses a set of fold table consisting of all *n*! ordering for *n* variable function and directly applying BDD reduction rules to each fold table without really generating decision diagrams for each possible ordering. Further, a new algorithm has been developed for fast generation of fold tables and the table consisting minimum number of nodes is used to generate optimal OBDD. Therefore it offers a simple and computationally efficient procedure for optimal OBDD generation along with storage economy.

## 2. Ordered Binary Decision Diagram (OBDD)

A Boolean function can be represented using OBDDs by imposing certain ordering relations among function variables where the canonicity of representation allows easy detection of many useful properties such as symmetry and unateness of variables.

***Definition_1***

*Binary Decision Diagrams* (BDDs) represents a Boolean functions as a rooted, directed acyclic graph with a vertex set containing two types of vertices, *non-terminal* and *terminal vertices*. A non-terminal vertex $v$ has two attributes i.e. (*i*) an argument *index* ($v$) $\in \{x_1, \ldots, x_n\}$ and (*ii*) two children indicated by *dashed* and *solid* lines for *low* ($v$) and *high* ($v$) respectively. A terminal vertex $v$ has an attribute *value* ($v$) $\in \{0, 1\}$ and has no outgoing edge.

An un-simplified BDD is basically a Binary Decision tree which contains $2^{n-1}$ non-terminal nodes. Considering an example function $f_1$ ($x_0$, $x_1$, $x_2$) = $\sum(3, 5, 6, 7)$, its BDD is shown in figure 1 (b) which is a direct mapping of the truth table of figure 1(a) in the tree form. In this tree the value of function is determined by tracing a path from the root to a terminal vertex. A BDD representation of an '$n$' variable function will initially have $2^n$-1 nodes[4, 12, 15] and the function value in the tree of figure 1(b) is determined by tracing a path from the root to a terminal vertex. The BDD can be further simplified using following node reduction rules (Bryant R. E. 1992; Drechslor R., Becker B. and Gockel N. 1996).

*(i) Deletion Rules:* If one or more non-terminal nodes are such that their both branches corresponding to "0" & "1" lead to a non-terminal successor node or to a terminal node then that non-terminal node can be deleted as shown in figure 2 (a).

*(ii) Merging Rules:* If two or more terminal (or non-terminal) nodes of the same label have the same "0" and "1" successors i.e. their left and right sons are equivalent then they can be merged in a single node shown in figure 2 (b).

The application of above reduction rules to the BDD of the function $f$ ($x_0$, $x_1$, $x_2$) = $\sum(3, 5, 6, 7)$ gives simplified BDD given in figure 1 (c).

***Definition_2***

OBDDs are generated by imposing a total ordering "<" over the set of variables so that for any vertex $u$ and either non-terminal child $v$; their respective variables must be ordered *var (u) < var (v)*. The OBDD generated using any ordering arrangement can be reduced to give simplified representation of a Boolean function. The OBDD shown in figure 1 is generated considering variable ordering $x_0 < x_1 < x_2$, however, in principle the variable ordering can be selected arbitrarily. Thus for a three variable function the total number of OBDDs can be 3! but the selection of an appropriate ordering is critical for efficient reduction of OBDD nodes.

***Definition_3***

The size of the OBDD is defined as the total number of terminal and non-terminal nodes in OBDD, for example, the size of the OBDD shown in figure 1 (c) is 6.

## 3. Effect of Variable Ordering

The nodal complexity of OBDDs for a given function greatly depends on variable ordering and hence it is possible that different OBDDs of same function can have different number of nodes. The identification of suitable ordering for generating OBDD of a function that has fewer nodes is not very crucial in the case of simple and medium complexity functions, however, for complex functions ($n \geq 5$) variable ordering has dramatic effect on the computational and storage requirements which directly effects the efficiency of the Boolean function manipulation algorithms in generating fewer node OBDD. Most applications requiring OBDD generation choose some ordering of the variables at the beginning and construct all possible OBDDs to identify the optimal OBDD having least number of nodes. This requires more computation as well as storage for $n$! OBDDs of an '$n$' variable function.

The effect of variable ordering on the number of OBDD nodes is demonstrated considering a six variable example function $f = x_0 \cdot x_3 + x_1 \cdot x_4 + x_2 \cdot x_5$ which will have 6! orderings and corresponding number of OBDDs. For simplicity, the OBDDs for only two out of total 6! orderings are shown in figure 3. The significance of variable ordering can be appreciated by observing the difference between the number of OBDD nodes for the two orderings of figure 3 (a) and (b). Although the difference between the numbers of nodes in the two OBDDs is only eight in our example but it may become extremely large for complex functions ($n \geq 5$). Therefore developing an efficient method for identification of appropriate variable ordering to generate *optimal OBDD* is an interesting problem to achieve minimization of storage requirements and reducing computations.

## 4. Identification of Optimal OBDD

This section describes a new algorithm for generating optimal OBDD of a Boolean function based on pre-calculation of nodes for each possible variable ordering. Some *definitions* that have been used are:

*(1)* If $I \subseteq \{0, 1, 2, \ldots, n\text{-}1\}$ then ϕ *(I)* is defined as the *set of ordering* on {0,1, ...,n-1}

ϕ *(I)* = $\{\phi : \phi$ is an ordering on $\{0, 1, 2, \ldots, n\text{-}1\}$

For example, an OBDD of a three variable function with ordering $x_1 < x_0 < x_2$ will be represented here as OBDD (1, 0, 2).

*(2)* A *fold table* symbolized by $TABLE_I$ is constructed corresponding to each of the $n$! variable orderings in which the entries in each table are made according to the sequence of minterm values of the given Boolean function. The total number of fold tables will be $n$! and generating all of them becomes a tedious work for large number of variables. For a particular ordering "$\phi$" the table is denoted by $TABLE_\phi$.

*(3)* If $v \in \{0, 1, 2, \ldots,(n\text{-}1)\}$ and $\phi$ is an ordering on $\{1, 2, \ldots, (n\text{-}1)\}$ $value_v\ (\phi)$ denotes the number of nodes on label $v$ in the OBDD ($\phi$).

Therefore the task of identifying the optimal OBDD can be simplified if an ordering "$\phi$" is determined using simplified procedure such that it minimizes $\sum_{v=0}^{n-1} value_v(\phi)$ .

## 4.1 Fold Table Generation

If $f(x_0, x_1, \ldots, x_{n\text{-}1})$ is an "$n$" variable Boolean function, where $x_i \in \{0, 1\}$ and $i = \{0, 1, \ldots, (n\text{-}1)\}$ then it shall have $2^n$ ordered $n$-tuples. The function $f$ assumes a particular value for each of $n$-tuples which may be considered as defining $n$-bit unsigned binary integer having decimal value ($d$) in the range 0 to $2^n$-1. The relation between the unsigned binary integer and its corresponding decimal value can be expressed as:

$$(x_{n\text{-}1}, x_{n\text{-}2}, \ldots, x_1, x_0) \Leftrightarrow \sum_{l=0}^{n-1} 2^i x_i = d \tag{1}$$

If the value of the function $y$ corresponding to decimal value $d$ of an $n$-tuple is expressed as $y_d$ then $f(x_0, x_1, \ldots, x_{n\text{-}1}) = y_d$ and the function values ($y_0, y_1, y_2 \ldots y_{2^n-1}$) define a finite sequence. The fold table generation involves interchanging input variables that changes decimal value of $n$-tuples reordering of function value and therefore it can be viewed as occurrence of finite sequences. Considering that $x_i$ and $x_j$ are two function variables that are to interchanged to generate new decimal value $d'$ of an $n$-tuple then

$$d' = \sum_{\substack{l=0 \\ l \neq i,j}}^{n-1} x_l 2^l + x_j 2^i + x_i 2^j \tag{2}$$

Equation (2) can be rewritten as:

$$d' = \sum_{\substack{l=0 \\ l \neq i,j}}^{n-1} x_l 2^l + x_j 2^i + x_i 2^j + x_i 2^i + x_j 2^j - x_i 2^i - x_j 2^j \tag{3}$$

Rearranging equation (3) and putting the value of $d$ from equation (1) we get

$$d' = d + x_j 2^i + x_i 2^j - x_i 2^i - x_j 2^j \tag{4}$$

Therefore equations (1) and (4) can be used to determine all $2^n$ sequences of entries for all $n$! tables of the fold table for an $n$-variable function by changing the positions of two variables at a time.

## 4.2 Node Reduction using Fold Table

This subsection discusses the method of finding a particular variable ordering for generating optimal OBDD of a given Boolean function by direct application of reduction rules to the fold table instead of generating OBDDs. The proposed method exploits the property that the value of the variables on the first $k$ labels depends only on their ordering and not on the ordering of the remaining ($n$-$k$) variables[14] for recording entries in the fold table by considering each $k$ labels ($k \leq n$). The "0" and "1" values of the function are stored as $w_0$ and $w_1$ respectively and for each such pair ($w_0$, $w_1$) it is determined whether or not a new node is required. This is achieved using following two criterions that are directly related to the *deletion* and *merging* rules:

*(i)* If $w_0 = w_1$ then do not create a new node since its both branches (0 & 1) point to the same vertex (deletion rule).

*(ii)* If there are $m$ nodes having $w_0 \neq w_1$ then at the same label and if their left sons and right sons are equivalent then don't create new node since it would be equivalent to $m$ (merging rule).

Otherwise create new node if both the above criterions are violated.

### *4.3 Algorithm for Optimal OBDD Generation*

Optimal OBDD generation algorithm requires following input parameters:

(1) Fold Table *($TABLE_I$)*; which is actually a mapping from $(0, 1)^{n-k}$, where $k = \left|I\right|$, $v \in I$ and also satisfying $\phi[k] = v$ to those nodes of OBDD ($\phi$) that are either internal nodes labeled with the member of $I$ or terminal nodes ("0" or "1").

(2) $\sum_{v=0}^{n-1} value(\phi)$, which is total number of nodes for each ordering, where $\phi \in \Phi(I)$.

The generation of fold table "$TABLE_I$" is achieved using equations (3) and (4) for each $\phi$, where $\phi \in \Phi(I)$ and computation of total number of nodes for each ordering. This is achieved by considering $TABLE_\phi$ for a particular ordering and storing each $k$ label of $TABLE_\phi$ with paired function value ($w_0, w_1$). The number of such pairs is given by $2^{\{(n-1)-k\}}$ for $k \le n$ and applying node generation criterion on the paired values ($w_0, w_1$) to compute total number of nodes. This process is iterated until all orderings have been considered and the specific ordering "$\phi$" that gives the minimum value of $\sum_{v=0}^{n-1} value(\phi)$ is identified as the *optimal ordering* corresponding to optimal OBDD of the function. Therefore the algorithm for identification of optimal variable ordering can be given as in figure 4.

The application of the algorithm given in figure 4 can be illustrated considering an example function $f(x_0, x_1, x_2, x_3) = x_1x_3 + x_0x_1'x_2'$ for ordering $TABLE_{(3, 2, 1, 0)}$. Now computation of the "*value*" of each label for the selected ordering can be achieved by assigning $k$=0, 1, 2 or 3 for generating paired output for each assignment of $k$ as given in the fold table 1. Recalling that total number of ($w_0, w_1$) pairs would be $2^{\{(n-1)-k\}}$, the number of ($w_0, w_1$) pairs would be 8, 4, 2 and 1 for $k$=0, 1, 2 and 3 respectively. Analyzing table 1 (a) it is found that out of total 8 ($w_0, w_1$) pairs; the first, second, fifth and sixth pairs have $w_0 = w_1$ and thus creation of OBDD nodes is not required for these pairs [criteria 4.2 (*i*)]. The remaining pairs are equivalent and hence they all can be represented using only one node in OBDD [criteria 4.2 (*ii*)]. Similarly out of total four pairs in table 1 (b) the first, second and fourth pairs don't need any node in OBDD representation, however, one node would be necessary for third pair. Finally, there is no scope of node reduction in tables 1 (c) and 1 (d) because their pairs are not covered by either of the node reduction criterions. Therefore they require as many numbers of nodes as the number of pairs and thus the OBDD for ordering $\phi(3, 2, 1, 0)$ will have total "five" nodes as shown in figure 5.

Now if the labels in OBDD are considered such that the first label starts from bottom corresponding to $v = 0$ and the subsequent higher labels are obtained by moving up to root node then it is clear that for the first label of the OBDD "4" nodes are deleted and "3" nodes are merged into one node. Further, "3" nodes are deleted at the second label (for $v$=1) while no node reduction is possible for third and fourth labels. Similarly the total number of OBDD nodes corresponding to remaining orderings of the function variables can be pre-calculated without really generating the OBDDs. The variable ordering that gives minimum number of OBDD nodes is selected to generate optimal OBDD of the function.

The illustrated method of optimal OBDD generation is equally applicable to all Boolean functions without necessitating actual generation of the OBDDs of a function. Therefore the proposed method offers both computational simplicity and storage economy in generating optimal OBDD which makes it more attractive for optimal OBDD generation of complex functions.

## 5. Conclusion

This paper described a new computationally efficient method for generating optimal OBDD of complex Boolean functions without generating all $n$! OBDDs of an '$n$' variable function. Our proposed method uses fast generation of *fold table* which is used to compute the total number of OBDD nodes for each possible orderings of function variables. Subsequently, the particular ordering corresponding to minimum number of nodes in OBDD is selected to generate optimal OBDD. The suggested algorithm eliminates the necessity of really generating all OBDDs of the function. Therefore it is computationally efficient as well as economical in storage requirements which make it suitable for manipulation of complex Boolean functions.

## References

Abusaleh M. Jabir, Dhiraj K. Pradhan, Ashutosh K. Singh Rajaprabhu T. L. (2007). "A Technique for Representing Multiple Output Binary Functions with Applications to Verification and Simulation" *IEEE transaction on computer*, vol. 56, no. 8, pp. 1133-1145.

P. W. Chandana Prasad, M. Maria Dominic and Ashutosh Kumar Singh (2003). "Improved Variable Ordering for ROBDD's", *Proc. of ICADL*'03, pp. 544-547, December 8-11, Kuala Lumpur, Malaysia.

Drecher R., Drechsler N. and Gunther W. (2000). "Fast Exact Minimization of BDD's," *IEEE Trans. CAD of Integrated Circuits and Systems*, vol. 19, no. 3, pp. 384-389.

Hong Y., Beerel P. A., Burch J. R., and McMillan K. L. (2000). "Sibling-Substitution-Based BDD Minimization Using Don't Cares," *IEEE Trans. CAD of Integrated Circuits and Systems*, vol. 19, no. 1, pp. 44-54.

Scholl C., Moller D., Molitor P. and Drechler R. (2000). "BDD Minimization Using Symmetries," *IEEE Trans. Comput.,* vol. 18, no. 2, pp. 81-99.

Litan L. H. and Molitor P. (2000). "Least Upper Bound for the Size of OBDDs Using Symmetry Properties," *IEEE Trans. Comput.*, vol. 49, no. 4, pp. 360-368.

Wang Y., Abd-el-Barr M. and McCrosky C. (1997). "An Algorithm for Total Symmetric OBDD Detection," *IEEE Trans. Comput.,* vol. 46, no. 6, pp. 731-733.

Bern J., Meinel C. and Slobodova A. (1996). "Global Rebuiliding of OBDD's Avoiding Memory Requirement Maxima", *IEEE Trans Comput.,* vol. 15, no. 1, pp. 131-134.

Minato S. (1996). "Fast Factorization Method for Implicit Cube Set Representation," *IEEE Trans. CAD. of Integrated Circuits and Systems,* vol. 15, no. 4, pp. 377-384.

Lai Y. T., Pedram M. and Vrudhula S. B. K. (1996). "Formal Verification Using Edge-Valued Binary Decision Diagrams," *IEEE Trans. Comput.,* vol. 45, no. 2 pp. 247-255.

Drechslor R., Becker B. and Gockel N. (1996). "Genetic Algorithm for Variable Ordering of OBDDs," *IEE Proc. Comput.-Digit. Tech.,* vol. 143, no. 6, pp. 364-368.

Sasao T. and Fujita M., (1996). "Representations of Discrete Functions", Kluwer Academic Publisher.

Shen A., Devadas S. and Ghosh A. (1995). "Probabilistic Manipulation of Boolean Functions Using Free Boolean Diagrams," *IEEE Trans. CAD. of Integrated Circuits and Systems,* vol. 14, no. 1, pp. 87-95.

Gergov J. and Meinel C. (1994). "Efficient Boolean Manipulation With OBDD's can be Extended to FBDD's," *IEEE Trans. Comput.*, vol. 43, no. 10, pp. 1197-1208.

Wegener I. (1994). "The Size of Reduced OBDDs and Optimal Read Once Branching Program for almost all Boolean Functions," *IEEE Trans. Comput.,* vol. 43, no. 11, pp. 1262-1269.

Bryant R. E. (1992). "Symbolic Boolean Manipulation with Ordered Binary-Decision Diagrams," *ACM computing surveys* vol. 24, no. 3, pp. 293-318.

Liaw H. T. and Lin C. S. (1992). "On the OBDD–Representation of General Boolean Functions," *IEEE Trans. Comput.,* vol. 41, no. 6,, pp. 661-664.

Friedman S. J. and Supowit K. J. (1990). "Finding the Optimal Variable Ordering for Binary Decision Diagram," *IEEE Trans. Comput.*, vol. 39, no. 5, pp. 710-714.

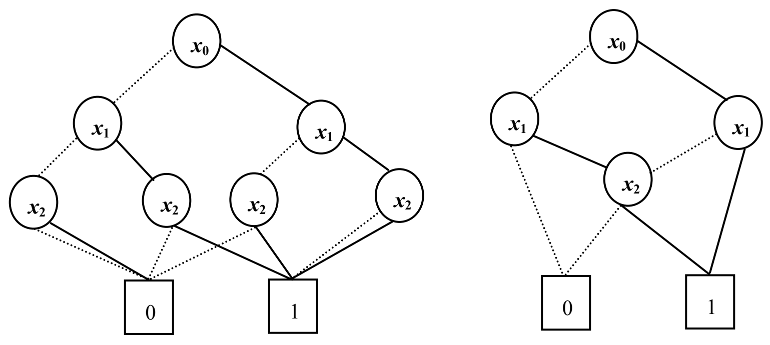
Bryant R. E. (1986). "Graph Based Algorithm for Boolean Function Manipulation," *IEEE Trans. Comput.,* vol. C-35, no. 8, pp. 677-691.

Akers S. B. (1978). "Binary Decision Diagrams," *IEEE Trans. Comput.,* vol. C-27, no. 6, pp. 509-516.

Table 1. Fold Table of the Function $f(x_0, x_1, x_2, x_3) = x_1x_3 + x_0x_1'x_2'$

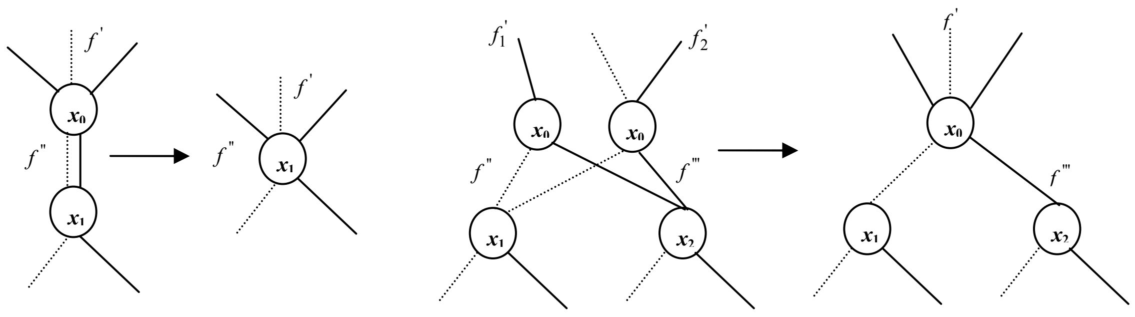
| (a) | | | (b) | | | (c) | | | (d) | | |
|---|---|---|---|---|---|---|---|---|---|---|---|
| $x_0$ | $f$ | | $x_1$ | $f$ | | $x_2$ | $f$ | | $x_3$ | $f$ | |
| 0 | 0 | $w_0$ | 0 | 0 | $w_0$ | 0 | 0 | $w_0$ | 0 | 0 | $w_0$ |
| 1 | 0 | $w_1$ | 0 | 0 | | 0 | 0 | | 0 | 0 | |
| 0 | 0 | $w_0$ | 1 | 0 | $w_1$ | 0 | 0 | | 0 | 0 | |
| 1 | 0 | $w_1$ | 1 | 0 | | 0 | 0 | | 0 | 0 | |
| 0 | 0 | $w_0$ | 0 | 0 | $w_0$ | 1 | 0 | $w_1$ | 0 | 0 | |
| 1 | 1 | $w_1$ | 0 | 1 | | 1 | 1 | | 0 | 1 | |
| 0 | 0 | $w_0$ | 1 | 0 | $w_1$ | 1 | 0 | | 0 | 0 | |
| 1 | 1 | $w_1$ | 1 | 1 | | 1 | 1 | | 0 | 1 | |
| 0 | 1 | $w_0$ | 0 | 1 | $w_0$ | 0 | 1 | $w_0$ | 1 | 1 | $w_1$ |
| 1 | 1 | $w_1$ | 0 | 1 | | 0 | 1 | | 1 | 1 | |
| 0 | 0 | $w_0$ | 1 | 0 | $w_1$ | 0 | 0 | | 1 | 0 | |
| 1 | 0 | $w_1$ | 1 | 0 | | 0 | 0 | | 1 | 0 | |
| 0 | 0 | $w_0$ | 0 | 0 | $w_0$ | 1 | 0 | $w_1$ | 1 | 0 | |
| 1 | 1 | $w_1$ | 0 | 1 | | 1 | 1 | | 1 | 1 | |
| 0 | 0 | $w_0$ | 1 | 0 | $w_1$ | 1 | 0 | | 1 | 0 | |
| 1 | 1 | $w_1$ | 1 | 1 | | 1 | 1 | | 1 | 1 | |

| $x_0$ | $x_1$ | $x_2$ | $f$ |
|---|---|---|---|
| 0 | 0 | 0 | 0 |
| 0 | 0 | 1 | 0 |
| 0 | 1 | 0 | 0 |
| 0 | 1 | 1 | 1 |
| 1 | 0 | 0 | 0 |
| 1 | 0 | 1 | 1 |
| 1 | 1 | 0 | 1 |
| 1 | 1 | 1 | 1 |



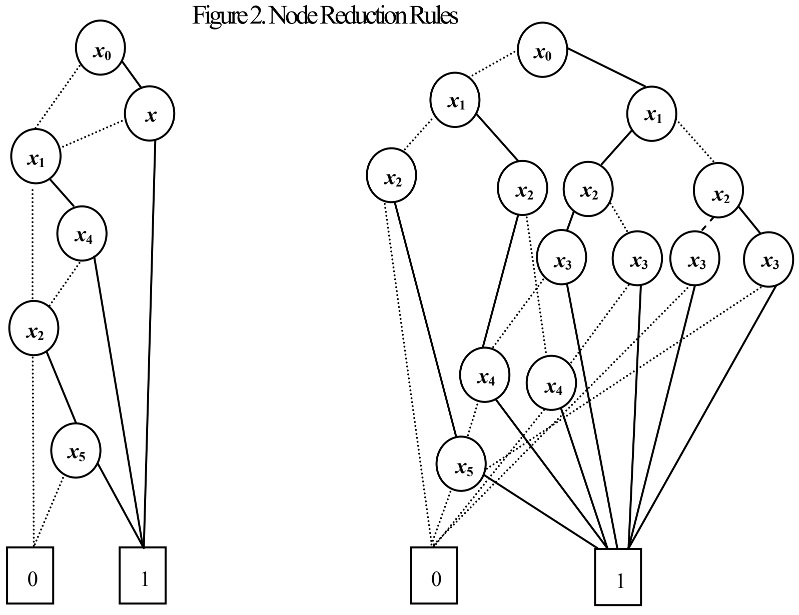
(a) (b) (c)

Figure 1. Truth Table and BDD of $f_1(x_0, x_1, x_2) = \sum(3, 5, 6, 7)$



(a) Deletion Rules (b) Merging Rules

Figure 2. Node Reduction Rules



(a) (b) $x_0<x_1<x_2<x_3<x_4<x_5$

Figure 3. Effect of Variable Ordering on OBDD Nodes for $f = (x_0 \cdot x_3 + x_1 \cdot x_4 + x_2 \cdot x_5)$

```
For any $\phi$; $\phi \in \phi(I)$
    [Compute $\sum_{v=0}^{n-1} value(\phi)$ ]
        BEGIN
        Store first label ($k$) with paired output values starting from 0
            If $w_0 = w_1$
            Then count ← count (do not create new node)
            Else Begin
                If $w_0 \neq w_1$ and their left and right sons are equivalent
                Then count ← count (do not create new node)
            Except Both case count +1 ← count (create new node)
            Store this count value as $\sum_{v=0} value(\phi)$
              Store next label with increment $k$ by 1 till $k \leq$ n-1 and repeat above procedure
Store $\sum_{v=0}^{n-1} value(\phi)$
    Similarly for each $\phi$; $\phi \in \phi(I)$
    [Compute $TABLE_{\phi}$, and $\sum_{v=0}^{n-1} value(\phi)$ ]
      Store $\sum_{v=0}^{n-1} value(\phi)$ for all $\phi$ and selected that $\phi$ which provides minimum value of
      $\sum_{v=0}^{n-1} value(\phi)$ that particular $\phi$ will be *optimal*.
```

Figure 4. Algorithm for Identification of Optimal Variable Ordering

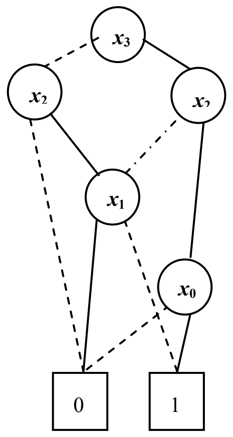


Figure 5. OBDD (3, 2, 1, 0) for $f(x_0,x_1,x_2,x_3) = x_1x_3 + x_0x_1'x_2'$