PAPER *Special Section of Selected Papers from the 8th Karuizawa Workshop on Circuits and Systems*

Implicit Representation and Manipulation of Binary Decision Diagrams

Hitoshi YAMAUCHI†, Nagisa ISHIURA††, *and* Hiromitsu TAKAHASHI†, *Members*

SUMMARY This paper presents *implicit representation of binary decision diagrams (implicit BDDs)* as a new efficient data structure for Boolean functions. A well-known method of representing graphs by binary decision diagrams (BDDs) is applied to BDDs themselves. Namely, it is a BDD representation of BDDs. Regularity in the structure of BDDs representing certain Boolean functions contributes to significant reduction in size of the resulting implicit BDD representation. Since the implicit BDDs also provide canonical forms for Boolean functions, the equivalence of the two implicit BDD forms is decided in time proportional to the representation size. We also show an algorithm to manipulate Boolean functions on this implicit data structure.

key words: *binary decision diagram (BDD), representation of Boolean functions, logic design verification, logic synthesis, implicit representation of graphs*

1. Introduction

A binary decision diagram (BDD) is a graph based data structure for representing Boolean functions. It is devised by Akers [1], and has been widely used in the area of computer-aided design of VLSI since an effective manipulation algorithm was developed by Bryant [2]. The BDD brought about great improvement in logic design verification [3], logic synthesis, test generation [4], etc. However, as BDDs are used in more sophisticated applications dealing with larger instances, we come to encounter BDDs whose size exceeds the memory capacity. This arouse a demand for yet more succinct representation for Boolean functions.

In this paper, we propose *implicit representation and manipulation* of BDDs as an efficient way of dealing with Boolean functions. The graph structure of a BDD is indirectly expressed in terms of Boolean functions by giving each node a binary code, instead of expressing edges in the graph one by one. Then the Boolean functions which implicitly represent the original BDD are expressed in the form of multiple-output BDDs.

The size of the implicit representation does not necessarily depend on the original BDD size and regularity in the original BDD structure leads to drastic reduction in resulting implicit representation. Since the canonicity of the representation is preserved, the equivalence checking on the implicit representation is decided in time proportional to the representation size.

Experimental results show that the asymptotic behaviors of the representation sizes of majority, adder and selector functions are largely improved. The memory requirement on many of the MCNC and ISCAS benchmarks are also reduced.

This paper is organized as follows: After formalizing BDD in Sect. 2, we describe the details of the implicit representation of BDDs in Sect. 3. We give algorithms to manipulate the implicit representation in Sect. 4, and show some experimental results in Sect. 5.

Manuscript received June 29, 1995.
Manuscript revised September 28, 1995.

†The authors are with the Faculty of Computer Science and System Engineering, Okayama Prefectural University, Soja-shi, 719-11 Japan.
††The author is with the Faculty of Information System Engineering, Osaka University, Suita-shi, 565 Japan.

2. Binary Decision Diagrams

2.1 OBDD and LOBDD

A binary decision diagram (BDD) is a data structure for representing Boolean functions using an acyclic directed graph.

Figure 1 shows examples of BDD representation; (a) and (b) are both BDDs representing two Boolean functions $f = x_3x_2 + x_1$, $g = x_3 + x_2 + x_1$. The non-terminal nodes in a BDD are labeled by variables ($\in \{x_1, x_2, \cdots, x_n\}$) and are called *variable nodes*. The variable of variable node v is denoted by $var(v)$. The terminal nodes in a BDD are labeled by Boolean values ($\in \mathcal{B} = \{0, 1\}$) and are called *constant nodes*. The Boolean value of the constant node v is denoted by $c(v)$. The number of the nodes in a BDD is referred to as the *size* of the BDD. A BDD has ordered m *initial nodes* denoted as $\vec{i} = (i_1, \cdots, i_m)$. For each node v is defined the level of v (denoted by $l(v)$) as follows:

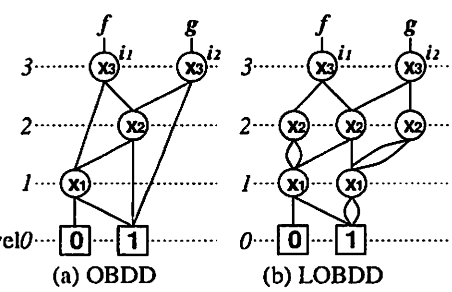


Fig. 1 Binary decision diagrams.

$$\begin{cases} l(v) = 0 & \cdots v \text{ is a constant node,} \\ \text{if } var(v) = x_j \text{ then } l(v) = j & \\ & \cdots v \text{ is a variable node.} \end{cases}$$

Each variable node has ordered two edges called the *0-edge* and the *1-edge* of the node. The nodes connected to the 0-edge and the 1-edge are represented as $e(v,0)$ and $e(v,1)$, respectively. In the figures in this paper, the 0-edge of a node is shown on the left side of the node, and the 1-edge on the right side. The BDDs handled in this paper are of a particular class called *Ordered BDDs* (OBDDs), in which each node v satisfies the following condition:

$$l(v) > l(e(v,0)), \ \ l(v) > l(e(v,1)).$$

The BDDs in Fig. 1 are both OBDDs.

Each node in an OBDD represent a Boolean function. The Boolean function f_v represented by node v is defined as follows:

$$f_v = \begin{cases} c(v) & \cdots v \text{ is a constant node,} \\ \overline{var(v)} \cdot f_{e(v,0)} + var(v) \cdot f_{e(v,1)} & \\ & \cdots v \text{ is a variable node.} \end{cases}$$

An OBDD represents an n-variable m-output Boolean function $(f_{i_1}, \cdots, f_{i_m})$.

An OBDD is called a *levelized OBDD* (LOBDD) if each variable node v satisfies

$$l(e(v,0)) = l(v) - 1, \ \ l(e(v,1)) = l(v) - 1.$$

Namely, an LOBDD is an OBDD whose edges do not "jump over" levels. An LOBDD in Fig. 1 (b) represents the same Boolean functions represented by the unlevelized BDD in (a). The level of an initial node of an LOBDD is always n. We mainly deal with LOBDDs in this paper. We reformulate the LOBDD as follows.

Definition 2.1: An LOBDD L representing an n-variable m-output Boolean function is a 4-tuple $L = (\vec{N}, \vec{e}, c, \vec{i})$, where

- $\vec{N} = (N_0, \cdots, N_n)$: N_j is the set of nodes of level j.
- $\vec{e} = (e_1, \cdots, e_n)$: $e_j(v,x)$ is a mapping $N_j \times \mathcal{B} \rightarrow N_{j-1}$ which represents the node connected to x-edge of node v.
- c : $c(v)$ is a mapping $N_0 \rightarrow \mathcal{B}$ which represents the Boolean value labeled to constant node v.
- $\vec{i} = (i_1, \cdots, i_m)$: $i_k \in N_n$ is an initial node. □

Two nodes u and v of level j in an LOBDD are equivalent if the functions represented by u and v are equal. Namely, $u \equiv v$ if

$$\begin{cases} c(u) = c(v) & \cdots u, v \text{ are constant nodes,} \\ e_j(u,0) \equiv e_j(v,0) \text{ and } e_j(u,1) \equiv e_j(v,1) & \\ & \cdots u, v \text{ are variable nodes.} \end{cases}$$

An LOBDD is called *reduced* if the LOBDD has no equivalent nodes. The process of transforming an LOBDD into the reduced form by eliminating all the equivalent nodes is called *reduction*. The reduced OBDD form [2] is obtained from the reduced LOBDD by eliminating all the redundant nodes (where a redundant node is a node v satisfying $e(v,0) = e(v,1)$). As well as the reduced OBDDs, the reduced LOBDDs are canonical forms for Boolean functions under a fixed variable order.

2.2 Operations for LOBDD

Typical operations for OBDD and LOBDD are Boolean binary operations, Boolean unary operations, equivalence checking, and substitution. The most fundamental and important ones among them are Boolean binary operations.

A Boolean binary operation for LOBDD is an operation which generates LOBDD $L_{f \circ g}$ representing Boolean function $f \circ g$ from two LOBDD L_f and L_g representing Boolean function f and g, respectively, where $\circ$ is an arbitrary Boolean operators such as AND and OR. These operations are achieved based on an algorithm that recursively traverses given two BDDs [2]. In this paper, we formalize the algorithm for Boolean binary operation using a notion of *product LOBDDs*, which is a generalization of the recursive algorithm.

A product LOBDD is a counterpart of the product finite state machine or the product automaton. The Fig. 2 illustrates how the product (AND) LOBDD of the two LOBDDs are constructed.

1. For each u and u', where u is a node of level l in the first LOBDD and u' is a node of level l in the second LOBDD, a node (u, u') is created as a node of level l in the product LOBDD.
2. Create an x-edge from node (u, u') to (v, v') in the product LOBDD, if x-edge of node u in the first LOBDD points to node v and x-edge of node u' in the second LOBDD points to node v'.
3. Set the Boolean value of constant node (v, v') to $c(v) \circ c(v')$. If the $\circ$ represents AND, for example,

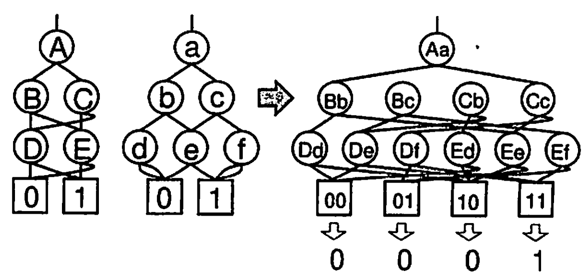


Fig. 2 Product LOBDD.

set the value of the constant nodes in the product LOBDD as shown in Fig. 2.

Formal definition of the product LOBDD is as follows. Here, it is assumed that each LOBDD has only one initial node and represent only one Boolean function.

Definition 2.2: A product LOBDD $prod(L_f, L_g, \circ) = (\vec{N}, \vec{e}, c, \vec{i})$ generated form two LOBDD $L_f = (\vec{N^f}, \vec{e^f}, c^f, \vec{i^f})$ and $L_g = (\vec{N^g}, \vec{e^g}, c^g, \vec{i^g})$ with respect to binary operator $\circ$ is defined as follows.

- $N_j = N_j^f \times N_j^g$.
- $e_j((y^f, y^g), x) = (e_j^f(y^f, x), e_j^g(y^g, x))$.
- $c((y^f, y^g)) = c^f(y^f) \circ c^g(y^g)$.
- $\vec{i} = ((i_1^f, i_1^g))$. □

An LOBDD representing $f \circ g$ is given by $L_{f \circ g} = prod(L_f, L_g, \circ)$.

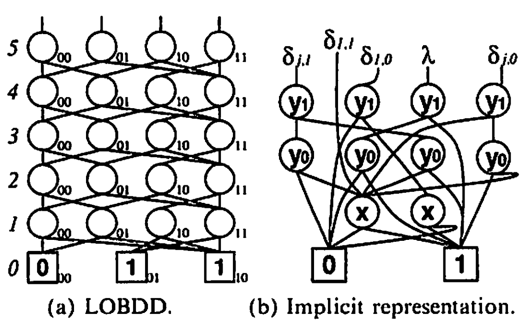
3. Implicit Representation of BDDs

In this paper, we present implicit ways of representing and manipulating LOBDDs. We first discuss the *implicit representations of LOBDDs* (iLOBDDs, for short).

3.1 Implicit Representations of LOBDD (iLOBDDs)

We can represent the connectivity in an LOBDD by Boolean functions by giving a binary code vector to each node in the LOBDD. For example, let us focus on the connectivity between levels 5 and 4 of the LOBDD in Fig. 3 (a). First of all, each node is given a binary code unique in its level. The codes are shown on the right side of the nodes in the figure. The 0-edge and the 1-edge of node 01 of level 5 connect to nodes 00 and 11 of level 4, respectively. Let Boolean function $\delta_5(y, x)$ denote the node of level 4 connected to the x-edge of node y of level 5. Then, this connectivity is represented as

$$\delta_5((0,1),0) = (0,0), \ \ \delta_5((0,1),1) = (1,1).$$



(a) LOBDD. (b) Implicit representation.

Fig. 3 LOBDD and its implicit representation.

We can represent all the other edges in the same way. Then δ_5 can be regarded as a 3-input 2-output Boolean functions. In the form of a Boolean expression,

$$\begin{aligned}\delta_5((y_1, y_0), x) &= (\delta_{5,1}((y_1, y_0), x), \delta_{5,0}((y_1, y_0), x)) \\ &= (y_1 y_0 + y_1 x + y_0 x, y_1 \overline{y_0} + x),\end{aligned}$$

δ_4, δ_3 and δ_2 are computed in the same way, all of which turn out equal to δ_5 in this example. For levels between 1 and 0, we have

$$\delta_1((y_1, y_0), x) = (x, \ y_1 \overline{x}).$$

Since the Boolean values of constant nodes 00, 01 and 10 are 0, 1 and 1, respectively, we can represent those Boolean values as a 2-input 1-output Boolean function

$$\lambda((y_1, y_0)) = y_1 + y_0.$$

In this way, the graph structure of an LOBDD is represented by Boolean functions δ_j and λ. We represent those Boolean functions by a multiple-output OBDD. We call this OBDD an iLOBDD (see Fig. 3 (b)).

In general, iLOBDD I representing an n-variable m-output LOBDD L is defined as follows, where the nodes of level j of the LOBDD L are coded by w_j bits and σ_j: $N_j \rightarrow \mathcal{B}^{w_j}$ represents the code of a node of level j.

Definition 3.1: iLOBDD I representing n-variable m-output LOBDD $L = (\vec{N}, \vec{e}, c, \vec{i})$ under coding $\sigma = (\sigma_0, \cdots, \sigma_n)$ is a 3-tuple $I = (\vec{\delta}, \lambda, \vec{s})$, where

- $\vec{\delta} = (\delta_1, \cdots, \delta_n)$: mapping δ_j: $\mathcal{B}^{w_j} \times \mathcal{B} \rightarrow \mathcal{B}^{w_{j-1}}$ represents the code of the node connected to a node of level j via an edge of the node which satisfies $\delta_j(\sigma_j(v), x) = \sigma_{j-1}(e_j(v, x))$.
- λ : $\mathcal{B}^{w_0} \rightarrow \mathcal{B}$ is the Boolean value of a constant node which satisfies $\lambda(\sigma_0(v)) = c(v)$.
- $\vec{s} = (s_1, \cdots, s_m)$ represents initial nodes, and satisfies $s_j = \sigma_n(i_j)$. □

The Boolean function represented by an iLOBDD is defined as the Boolean function represented by the LOBDD which is represented by the iLOBDD implicitly. The total size of the multi-output OBDDs representing Boolean functions constructing an iLOBDD is called the *size of the iLOBDD*.

The following two factors contribute to reduction of the size of iLOBDDs.

i) Similarity of the connectivity among different pairs of levels: As in Fig. 3, if connectivities are similar (or equal) among different pairs of levels, subgraphs of the OBDD implicitly representing these connections are shared and the number of the nodes are reduced.

ii) Regularity of the connectivity between each pair of levels: If the connection between a pair of levels is regular, Boolean function δ_j representing this connection becomes simple and makes the OBDD small.

In the case of the OBDDs and the LOBDDs, each Boolean function has a unique reduced form under an arbitrary fixed variable order. The iLOBDDsare also canonical under the following conditions.

Proposition 3.1: Two iLOBDD I_f and I_g representing the same Boolean function are congruent if the following conditions are satisfied.

1) I_f and I_g represent reduced LOBDDs with the same variable order.

2) I_f and I_g represent LOBDDs under the same coding.

3) For the codes not representing any node, the values of functions of I_f and I_g are equal. □

1) is a condition where the forms of the LOBDDs represented by I_f and I_g are congruent. Under the condition 2), the value of δ_j and λ for the codes representing nodes are unique. 3) ensures that the values of the functions are equal for all the codes. The three conditions thus guarantee that functions δ_j and λ of I_f and I_g match completely.

In order to make 3) hold, we only have to define the values of the functions as 0 for those codes that do not represent any nodes. This condition is also satisfied by deciding the values of the functions using generalized cofactor [9].

3.2 Coding of Nodes

Memory efficiency of iLOBDD depends greatly on how the nodes are coded. In this paper, we propose two coding methods for iLOBDD, named a *minimum path coding* and a *binary coding*. These coding methods have the following features.

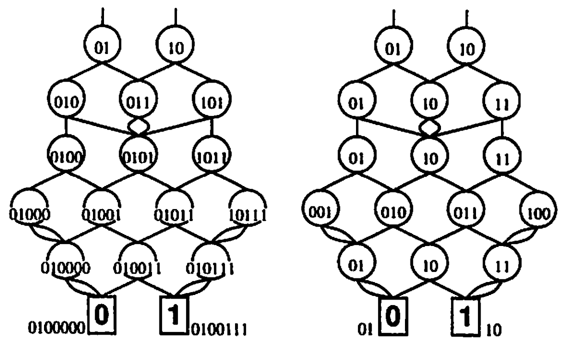
i) Given an LOBDD, the codes for the nodes are determined uniquely.

ii) There are implicit algorithms that convert an iLOBDD of an arbitrary coding into the iLOBDD of these codings.

A consequence of property i) is that the iLOBDD representing a given LOBDD is unique. Thus a iLOBDD is a canonical representation of a Boolean function if the LOBDD represented by the iLOBDD is a reduced LOBDD. We call these codings *standard codings* of iLOBDDs.

Minimum path coding

The minimum path coding uses $\lceil \log(m+1) \rceil +$ $(n-j)$ bits† to represent a node in level j of an n-input m-output LOBDD. The code $\sigma(v)$ for a node v is determined according to the following rule (refer to Fig. 4 (a)).



(a) Minimum path coding. (b) Binary coding.

Fig. 4 Standard codings.

i) Binary representation of k is assigned to the k-th initial node s_k.

ii) If the x-edge of node u is connected to node v, the code of node v is determined as $\sigma(u)||x$ which is the concatenation of $\sigma(u)$ and x. If more than two nodes $(u_1, x_1), \cdots, (u_k, x_k)$ satisfy $\delta(u_i, x_i) = v$ $(i = 1, \cdots, k)$, then the code of v is determined as $\sigma(v) = \min\{\sigma(u_1)||x_1, \cdots, \sigma(u_k)||x_k\}$, where min chooses the code whose value as an integer is the smallest.

Binary coding

If there are W_j nodes in level j of an LOBDD, the binary coding uses $\lceil \log(W_j + 1) \rceil$ bits† to code a node in level j. The rule of this coding is defined as follows (refer to Fig. 4 (b)).

i) Give ascending sequential numbers to the nodes of the LOBDD by the preorder depth-first traversal starting from the initial nodes (assuming an arbitrary order for the initial nodes).

ii) For each level j, sort the nodes of the level in the ascending order of the number given in i), and assign the binary expressions of $1, \cdots, W_j$ to the nodes according to the determined order.

3.3 Representation of Multiple Boolean Functions

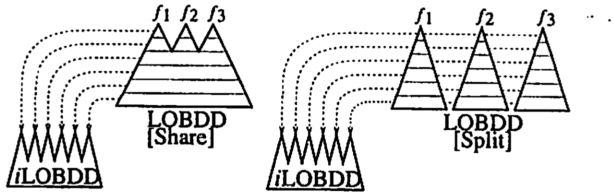
There are two ways of representing m Boolean functions simultaneously by iLOBDDs:

i) Represent an m-output LOBDD. (*share* system)

ii) Represent m 1-output LOBDDs. (*split* system)

(see Fig. 5).

†We need $\lceil \log N + 1 \rceil$ bits to code N nodes because we exclude code $00 \cdots 0$ for notational and implementational convenience.



(a) Share system. (b) Split system.

Fig. 5 Representation of multiple functions.

In the case of explicit LOBDDs, the representation size is always the larger in the split system because homogeneous subgraphs remain unshared. However, in the case of iLOBDDs, the split system is not necessarily disadvantageous because subgraphs in the OBDDs that construct an iLOBDD is shared. On the contrary, the split system may be advantageous in some cases, because it provides shorter code length and it can reflect the regularity in the LOBDD of each function. As shown in the later section, the split system outperforms the share system on many instances in experiments.

4. Manipulation of iLOBDD

By "implicit manipulation," we refer to constructing iLOBDD $I_{f\circ g}$ that represents $f \circ g$ directly from iLOBDDs I_f and I_g that represents f and g. In this section, we propose an algorithm of *implicit manipulation* for binary operations, which is the most important and the most time consuming among operations on BDDs.

4.1 Implicit Manipulation of Binary Operations

In order to achieve binary operations, we have to represent plural Boolean functions. We assume that objective Boolean functions are represented by iLOBDDs of the split system. Namely, we compute a 1-output iLOBDD $I_{f\circ g}$ representing $f \circ g$ from two 1-output iLOBDDs I_f and I_g and a Boolean operator $\circ$.

This operation is achieved by constructing the product LOBDD implicitly from I_1 and I_2 in accordance with the definition in Sect. 2.2. In this section, we also present implicit algorithms to achieve reduction and code conversion (into canonical coding) on the iLOBDD obtained by the binary operation.

4.2 Construction of the Product iLOBDD

Let the LOBDDs represented by I_f and I_g be denoted as L_f and L_g, respectively. We call iLOBDD I representing $prod(L_f, L_g, \circ)$ as the product iLOBDD of I_f and I_g with respect to the operator $\circ$. The algorithm to construct the product iLOBDD $I = (\vec{\delta}, \lambda, \vec{s})$ from $I_f = (\vec{\delta^f}, \lambda^f, \vec{s^f})$ and $I_g = (\vec{\delta^g}, \lambda^g, \vec{s^g})$ is obtained by converting the definition in Sect. 3.2 to following procedure.

1) For the initial nodes: $s_1 \leftarrow s_1^f || s_1^g$.

2) Repeat 2a) for $j = n, \cdots, 1$.

   2a) $\delta_j(y_f||y_g, x) \leftarrow \delta_j^f(y_f, x)||\delta_j^g(y_g, x)$.

3) $\lambda(y_f||y_g) \leftarrow \lambda^f(y_f) \circ \lambda^g(y_g)$.

4.3 Reduction

As mentioned in Sect. 2.1, the reduction of an LOBDD is to remove all the equivalent nodes in the LOBDD. Removal of equivalent nodes is realized by 1) detecting a set of equivalent nodes and choosing an arbitrary representative and 2) reconnecting all the other edges connecting to the other nodes in the equivalent node set to the representative (see Fig. 7). Since there is no edge that jumps over levels, reduction is achieved by applying the removal of equivalent nodes level by level starting from level 0 up to level n. The outline of the procedure of reduction of an LOBDD is as follows.
`for` $j = 0$ `to` $n-1$

1) Partition the set of the nodes of level j into the sets $E^1, \cdots, E^p$ of equivalent nodes.

2) Select a representative node y_0^k for each E^k.

3) For each E^k, redirect the edges pointing y_l^k ($\in E^k - \{y_0^k\}$) to y_0^k.

The following is the details of the procedure to carry out 1) to 3) implicitly on iLOBDD.

1) Partition of the set of nodes into the equivalent classes:

We use *equivalence relation function* E_j: $\mathcal{B}^{w_j} \times \mathcal{B}^{w_j} \rightarrow \mathcal{B}$ to represent the result of this step. This Boolean function takes the codes of two nodes as inputs and returns the value 1 if and only if the two nodes belong to the same set. The equivalence relation function of level j E_j is computed as follows.

a) if $j = 0$ $\quad E_0(y, y') \leftarrow \lambda(y) \equiv \lambda(y')$,

b) if $j > 0$ $\quad E_j(y, y') \leftarrow \forall x(\delta_j(y, x) \equiv \delta_j(y', x))$,

where $p \equiv q$ means $\overline{p \oplus q}$ and $v \equiv u = (\overline{v_1 \oplus u_1} \cdot \cdots \cdot \overline{v_n \oplus u_n})$ for $v = (v_1, \cdots, v_n)$ and $u = (u_1, \cdots, u_n)$.

2) Selection of the representative node from an equivalent set:

This operation is realized by using the compatibility projection [7]. For each of the equivalent sets represented by an equivalence relation function, the compatibility projection chooses the node which is the minimum when it is regarded as the binary coding of an integer. The result is obtained in the form of function $\hat{E}(y, y')$ that returns 1 if and only if y' is the representative of the set that y belongs to. We denote this as

$$\hat{E}(y, y') \leftarrow cproj(E(y, y')).$$

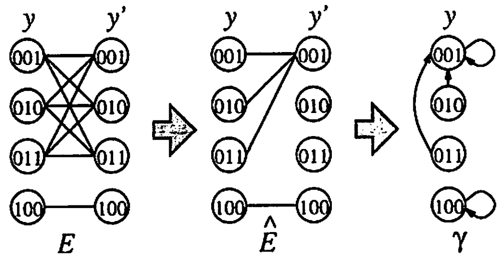


Fig. 6 Selection of representative nodes using compatibility projection.

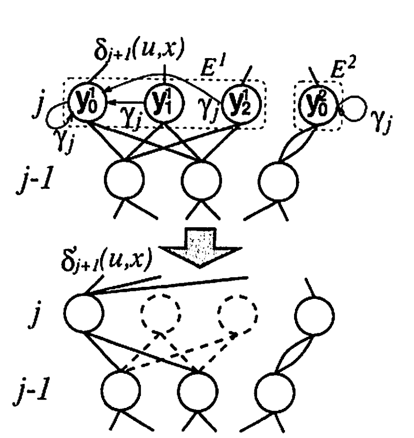


Fig. 7 Reduction of an LOBDD.

(See Fig. 6.)

In the final step, we convert $\hat{E}$ into a Boolean function γ that takes code y as input and returns the code of the representative of the set that y belongs to. This function of level j is obtained by

$$\gamma_j(y) = (\gamma_{j,1}(y), \cdots, \gamma_{j,w_j}(y)),$$
$$\gamma_{j,k}(y) \leftarrow \exists y'(y'_k \cdot \hat{E}_j(y, y')),$$

where $\exists y f(y, x)$ means existential quantification and is defined as follows when $y = (y_1, \cdots, y_n)$.

$$\exists y f(y, x) = \exists y_1 \cdots \exists y_n f(y, x),$$
$$\exists y_i f((y_1, \cdots, y_{i-1}, y_i, y_{i+1}, \cdots, y_n), x)$$
$$= f((y_1, \cdots, y_{i-1}, 0, y_{i+1}, \cdots, y_n), x)$$
$$+ f((y_1, \cdots, y_{i-1}, 1, y_{i+1}, \cdots, y_n), x).$$

3) Reconnection of the edges:

Function δ'_{j+1} that represents the connectivity of the edges after the reconnection is expressed as

$$\delta'_{j+1}(y, x) \leftarrow \gamma_j(\delta_{j+1}(y, x))$$

(See Fig. 7).

The algorithm of reduction of an iLOBDD is summarized as follows.

1) $E_0(y, y') \leftarrow \lambda(y) \equiv \lambda(y')$.

2) Repeat 2a) to 2d) for $j = 1, 2, \cdots, n$.

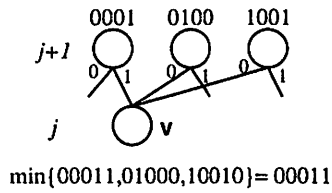


Fig. 8 Selection of the code in the minimum path coding.

2a) $\hat{E}_{j-1}(y, y') \leftarrow cproj(E_{j-1}(y, y'))$.

2b) $\gamma_{j-1}(y) = (\gamma_{j-1,1}(y), \cdots, \gamma_{j-1,w_j}(y))$,
$\gamma_{j-1,k}(y) \leftarrow \exists y'(y'_k \cdot \hat{E}_{j-1}(y, y'))$.

2c) $\delta'_j(y, x) \leftarrow \gamma_{j-1}(\delta_j(y, x))$.

2d) $E_j(y, y') \leftarrow \forall x\ (\delta'_j(y, x) \equiv \delta'_j(y', x))$.

4.4 Recoding

Figure 8 shows how the minimum path codes of level j are determined from the minimum path codes of level $j + 1$. The candidates for the code of node v are 00011, 01000 and 10010. Namely, the concatenations of y and x satisfying $\delta_{j+1}(y, x) = v$ are the candidates of the code of v. Chosen from the candidates as the minimum path code of v is the minimum one when regarded as the binary coding of an integer. This is also computable using compatibility projection.

Recoding is achieved level by level, starting from the initial nodes (level n) down to the constant nodes (level 0).

1) Give arbitrary codes to the nodes of level n (the initial nodes).

2) **for** $j = n - 1$ **down to** 0

2a) For each node y_k of level j, generate a set C_j^k of the candidates of the codes that are assignable to the node.

2b) For each C_j^k, choose the minimum code from the candidates as the representative.

2c) Convert the function of the edges of level $j + 1$ to the new function that outputs the code of the representative selected in 2b).

The details of each step are as follows.

1) Construction of the set of the candidate codes:

We use relation function C_j: $\mathcal{B}^{w_j} \times \mathcal{B}^{w'_j} \rightarrow \mathcal{B}$ to represent the result of this step. $C_j(y, y')$ returns 1 if and only if y' is a candidate of the new code of the node whose original code is y. Relation function C_j is computed by the following procedure, where $\bar{P}_{\hat{C}_{j+1}}$: $\mathcal{B}^{w'_{j+1}} \rightarrow \mathcal{B}^{w_{j+1}}$ takes the new code of a node of level $j + 1$ as an input and returns the original code of the node.

a) if $j = n$ $\quad C_n(y, y') \leftarrow (y \equiv s_1) \cdot (y' \equiv binary(1))$

b) if $j \leq n - 1$

Table 1 Experimental results (MCNC benchmarks).

circuit	in	out	OBDD	LOBDD (share)	LOBDD (split)	iLOBDD (share)	iLOBDD (split)
add6	12	7	28	97	221	61	8
alu1	12	8	15	96	222	50	3
alu2	10	8	52	117	256	101	21
alu3	10	8	51	119	245	106	19
apla	10	12	88	140	389	146	36
dk17	10	11	54	114	302	100	13
sao2	10	4	80	110	166	122	31
average			1.00	2.12	4.17	1.78	0.44

$C_j(y,y') \leftarrow \exists\breve{y}\exists x((y \equiv \delta_{j+1}(\bar{P}_{\hat{C}_{j+1}}(\breve{y}),x)) \cdot (y' \equiv \breve{y}||x))$

2) Selection of representative codes:

Selection of the representative codes whose integer values are the smallest is carried out in the same way as in reduction. By using compatibility projection, we derive from C_j the relation function $\hat{C}_j$ that returns 1 if and only if the second argument is the representative code of the first argument (the original code).

We also generate functions $P_{\hat{C}_j}(y)$ and $\bar{P}_{\hat{C}_j}(y')$ that returns the representative code of y and vise versa. These functions are defined as follows.

$$P_{\hat{C}_j}(y) = (P_{\hat{C}_j,1}(y), \cdots, P_{\hat{C}_j,w'_j}(y)),$$
$$P_{\hat{C}_j,k}(y) \leftarrow \exists y' \, (y'_k \cdot \hat{C}_j(y,y')),$$
$$\bar{P}_{\hat{C}_j}(y) = (\bar{P}_{\hat{C}_j,1}(y), \cdots, \bar{P}_{\hat{C}_j,w_j}(y)),$$
$$\bar{P}_{\hat{C}_j,k}(y') \leftarrow \exists y \, (y_k \cdot \hat{C}_j(y,y')).$$

(Note that $\hat{C}_j$ is a one to one relation.)

3) Conversion of the functions representing the edges:

We can obtain the new function δ' by $\delta'_{j+1}(y',x) \leftarrow P_{\hat{C}_j}(\delta_{j+1}(\bar{P}_{\hat{C}_{j+1}}(y'),x))$.

The following is a summary of the algorithm of the recoding to the minimum path coding.

1) $C_n(y,y') \leftarrow (y \equiv s_1) \cdot (y' \equiv binary(1))$.
2) Generate functions $P_{\hat{C}_n}(y)$ and $\bar{P}_{\hat{C}_n}(y')$.
3) Repeat 3a) to 3d) for $j = n-1, \cdots, 0$
   3a) $C_j(y,y') \leftarrow \exists\breve{y} \, \exists x \, ((y \equiv \delta_{j+1}(\bar{P}_{\hat{C}_{j+1}}(\breve{y}),x)) \cdot (y' \equiv \breve{y}||x))$.
   3b) $\hat{C}_j(y,y') \leftarrow cproj(C_j(y,y'))$.
   3c) Generate functions $P_{\hat{C}_j}(y)$ and $\bar{P}_{\hat{C}_j}(y')$.
   3d) $\delta'_{j+1}(y',x) \leftarrow P_{\hat{C}_j}(\delta_{j+1}(\bar{P}_{\hat{C}_{j+1}}(y'),x))$.
4) $\lambda'(y') \leftarrow \lambda(\bar{P}_{\hat{C}_0}(y'))$.

Code conversion to the binary coding is achieved by applying "code compression" to the iLOBDD of the minimum path coding obtained by the algorithm above. This code compression process is also done implicitly [5].

5. Experimental Result

5.1 Size of the Representation

We compared the size of the representation on MCNC and ISCAS85 benchmarks and some combinational circuits. This experiment followed the following steps.

1) The reduced OBDD is generated from the circuit descriptions.
2) The reduced OBDD is converted into the reduced LOBDD.
3) The iLOBDD is constructed from the reduced LOBDD.

We used the binary coding for the codes of the iLOBDD. The values of the functions for the codes that do not have corresponding nodes were decided by generalized cofactor operation [9]. We used the SBDD package of [8] to represent the OBDDs.

Table 1 shows the results on MCNC benchmarks. The columns show the circuit name, the number of the inputs, the number of the outputs, the numbers of the nodes of the reduced OBDD, of the reduced LOBDD (the share system and the split system) and of the iLOBDD (the share system and the split system). Since the negative edges, the inverse edges and the variable shift edges [8] are implemented in the package used to represent the OBDDs and the iLOBDDs, the numbers of the nodes shown are smaller than the representation by the plain OBDDs. In this package, each node requires about 20 bytes. The bottom line of the table shows the average of the relative performance normalized to "OBDD". The average involves 32 circuits including those fail to appear in the table. We used the variable orderings in [6] which are optimum in terms of the OBDD size.

From the comparison between OBDD and iLOBDD (the share system and the split system), we can see the effect of the implicit representation method. Although iLOBDDs of the share system did not do better than OBDDs on any benchmarks, iLOBDDs of the split system outperformed OBDD on 27 circuits out of 32. On the average, memory requirement was reduced to 44% of the OBDD.

Table 2 Experimental result (ISCAS85 benchmarks).

circuit	in	out	OBDD	iLOBDD (split)
C432	36	7	1706	964
C499	41	32	25331	7274
C1355	41	32	25331	7274
C1908	33	25	27205	31325
average			1.00	0.48

As a way of representing multiple output functions, the share system is better in explicit LOBDDs but the split system is better in implicit iLOBDDs. Comparison between LOBDD and iLOBDD shows that the number of the nodes are reduced to 11% in the split system. This is commonly observed across all the other circuits we made experiments on. We can conclude that the implicit representation has a significant effect.

Table 2 shows the results on ISCAS85 benchmarks. We used the variable orderings which we find in the original files.

From the comparison between the OBDD and iLOBDD, we can observe that memory requirement is also reduced on large scale Boolean functions. The numbers of the nodes are much smaller on all the circuits except for C1908 and memory requirement is reduced to 48% on the average.

Table 3 shows the comparison between OBDDs and iLOBDDs (of the split system) for some classes of Boolean functions. The orders in the bottom lines are the results of regression analysis.

Table 3(a) shows the results for n-input majority functions. The size of the OBDDs increases in $O(n^{1.91})$, while that of the iLOBDDs increases in $O(n^{1.35})$. In general, similar effect will be observed for symmetric functions with regularity.

Table 3(b) and (c) are the results for the selector functions that output one of n data inputs $d_0, \cdots, d_{n-1}$ selected by $\log n$ control inputs $c_0, \cdots, c_{\log n-1}$. (b) and (c) are the results of different two variable orderings; $c_0, \cdots, c_{\log n-1}, d_0, \cdots, d_{n-1}$ in (b) and $d_0, \cdots, d_{n-1}, c_0, \cdots, c_{\log n-1}$ in (c). It is known that the size of the OBDD is $O(n)$ with ordering (b) but $O(2^n)$ with ordering (c). The iLOBDD is not as good as the OBDD with the ordering (b), but the iLOBDD shows stunning performance with ordering (c). It achieves exponential reduction and thus the size of the iLOBDD of the selection functions are bounded by polynomial in either orderings.

Table 3(d) and (e) are the results for the binary adder functions that take $2n+1$ inputs (two n-bit data $a_0, \cdots, a_{n-1}, b_0, \cdots, b_{n-1}$ and 1-bit carry c_0) and output $n+1$ results of adder (n-bit data $s_0, \cdots, s_{n-1}$ and 1-bit carry c_n). The variable order in (d) is an interleaving order of two inputs from msb to lsb $(b_{n-1}, a_{n-1}, \cdots, b_0, a_0, c_0)$, and the order in (e) is a cascaded order of two inputs from lsb to msb $(c_0, a_0, \cdots, a_{n-1}, b_0, \cdots, b_{n-1})$. In the case of the OBDDs, it is known that the size is $O(n)$ with order (d), and exponential with order (e). In the case of the iLOBDDs, the sizes are constant with the order (d) and $O(n)$ even with the order (e). In both cases, significant reduction in memory size is achieved.

Table 3 Results for some classes of functions.

(a) Majority functions.

n	OBDD	iLOBDD
4	6	3
16	72	28
64	1056	175
256	16512	830
	$O(n^{1.91})$	$O(n^{1.35})$

(b) Selector functions $\left(\begin{array}{c}\text{with variable order} \\ c_0, \cdots, c_{\log n-1}, d_0, \cdots, d_{n-1}\end{array}\right)$.

n	OBDD	iLOBDD
4	4	7
16	16	66
64	64	317
	$O(n)$	$O(n^{1.28})$

(c) Selector functions $\left(\begin{array}{c}\text{with variable order} \\ d_0, \cdots, d_{n-1}, c_0, \cdots, c_{\log n-1}\end{array}\right)$.

n	OBDD	iLOBDD
4	15	1
8	255	1
16	65535	1
	$O(2^n)$	$O(1)$

(d) Adder functions $\left(\begin{array}{c}\text{interleaving order} \\ \text{from msb to lsb} \\ b_{n-1}, a_{n-1}, \cdots, b_0, a_0, c_0\end{array}\right)$.

n	OBDD	iLOBDD
4	21	8
8	41	8
16	81	8
32	161	8
	$O(n)$	$O(1)$

(e) Adder functions $\left(\begin{array}{c}\text{cascaded order} \\ \text{from lsb to msb} \\ c_0, a_0, \cdots, a_{n-1}, b_0, \cdots, b_{n-1}\end{array}\right)$.

n	OBDD	iLOBDD
4	64	22
6	238	36
8	916	50
10	3610	64
12	14368	78
	$O(2^{0.98n})$	$O(n)$

As we have seen, iLOBDD brings about great improvement in memory requirement for some classes of functions. There are functions whose iLOBDD size stays polynomial (of number of inputs n) while their OBDD size becomes exponential. On the other hand, we can show with an easy proof that iLOBDD size of functions is always polynomial if the OBDD size of the functions is polynomial.

5.2 Computation Time

We implemented an iLOBDD package and measured the time to construct iLOBDDs of n-bit adder functions. The result is shown in Table 4. The iLOBDDs are constructed by applying AND, OR, NOT, and EXOR operations to the iLOBDDs representing one variable Boolean functions. The variable order was an interleaving order from msb to lsb. This experiment was done on a workstation HP712/80 (with 128MB memory).

Implicit manipulation of iLOBDDs took large computation time. This may be caused by the complexity of the operation for iLOBDDs. There is a lot of room for improvement in coding of the package. We could at least obtain the same order of speed performance as the conventional OBDD package by running

Table 4 Execution time on n-bit adder.

n	CPU (sec)
4	54.43
8	216.98
12	659.48
16	1851.35

explicit manipulation algorithm on the implicit data structure.

6. Conclusion

We proposed a new efficient data structure of Boolean functions based on an implicit representation of BDDs and showed algorithms to carry out various operations on this data structure.

Comparison of the memory requirement between the conventional BDDs and the new implicit BDDs showed that our new data structure brought about significant reduction in memory requirement. We can expect that our implicit representation can handle large scale Boolean functions which exceeds the capacity of the conventional BDDs.

One big challenge is to develop faster manipulation package for iLOBDDs. Another important issue is to find better coding methods for iLOBDDs. It is also an interesting research theme to formulate implicit representation of OBDDs as well as LOBDDs.

Acknowledgment

The authors would like to thank Professor I. Shirakawa of Osaka University for his support and suggestions. We would also like to thank Professor E. Clarke of CMU and Dr. K. Hamaguchi of Kyoto University for offering their BDD package and helpful comments.

References

[1] S.B. Akers, "Binary decision diagrams," IEEE Trans. Comput., vol.C-27, no.6, pp.509–516, June 1978.
[2] R.E. Bryant, "Graph-based algorithms for Boolean function manipulation," IEEE Trans. Comput., vol.C-35, no.8, pp.677–691, Aug. 1986.
[3] J.R. Burch, E.M. Clarke, and K.L. McMillan, "Sequential circuit verification using symbolic model checking," Proc. ACM/IEEE 27th DAC, pp.46–51, June 1990.
[4] H. Choi, T. Kohara, N. Ishiura, I. Shirakawa, and A. Motohara, "Test generation for sequential circuits based on Boolean function manipulation" (in Japanese), IEICE Trans. Fundamentals, vol.J76-A, no.6, pp.835–843, June 1993.
[5] K. Hamaguchi and E. Clarke, private communication, Oct. 1994.
[6] N. Ishiura, H. Sawada, and S. Yajima, "Minimization of binary decision diagrams based on exchanges of variables," Proc. IEEE ICCAD-91, pp.472–475, Nov. 1991.
[7] B. Lin and A.R. Newton, "Implicit manipulation of equivalence classes using binary decision diagrams," Proc. IEEE ICCD, 1991.
[8] S. Minato, N. Ishiura, and S. Yajima, "Shared binary decision diagram with attributed edges for efficient Boolean function manipulation," Proc. ACM/IEEE 27th DAC, pp.52–57, June 1990.
[9] H.J. Touati, H. Savoj, B. Lin, R.K. Brayton, and A. Sangiovanni-Vincentelli, "Implicit state enumeration of finite state machines using BDD's," Proc. IEEE ICCAD-90, pp.130–133, Nov. 1990.

Hitoshi Yamauchi was born in Okayama, Japan, on January 31, 1971. He received the B.E. and M.E. degree from University of Osaka Prefecture in 1993 and 1995, respectively. In 1995, he joined the Faculty of Computer Science and System Engineering of Okayama Prefectural University. His research interests include computer-aided design of VLSI and computer algorithms. He is a member of the Institute of Systems, Control and Information Engineers.

Nagisa Ishiura was born in Kyoto, Japan, in 1961. He received the B.E., M.E. and Ph.D. degrees in information science from Kyoto University, Kyoto, Japan, in 1984, 1986, and 1991, respectively. In 1987, he joined the Department of Information Science, Kyoto University, where he was an instructor until April 1991. He joined the Department of Information Systems Engineering, Osaka University, Osaka, Japan, as a Lecturer where he was promoted to Associate Professor in December 1994. His current interests include design verification and test generation of digital circuits, logic synthesis, and hardware description languages. He is a member of IEEE and Information Processing Society of Japan.

Hiromitsu Takahashi received the B.E., M.E. and Ph.D. degrees from Osaka University in 1965, 1968 and 1971, respectively. He is a Professor at the Faculty of Computer Science and System Engineering of Okayama Prefectural University. From 1971 to 1994, he was with the Department of Mathematical Sciences of University of Osaka Prefecture. His research interests include graph theory and computer algorithms. He is a member of IEEE, the Mathematical Society of Japan and Information Processing Society of Japan.