数理解析研究所講究録
906 巻 1995 年 196-203

Expressive Power of Binary Decision Diagrams Representing Sum-of-product Form
積和形論理式を表す二分決定グラフの表現能力

Koyo NITTA Yasuhiko TAKENAGA Shuzo YAJIMA
新田 高庸 武永 康彦 矢島 脩三
Department of Information Science, Faculty of Engineering, Kyoto University
京都大学工学部 情報工学教室

1 Introduction

It is often required to represent and manipulate Boolean functions efficiently. Binary Decision Diagrams (BDDs) [1, 2] are graph-based representations of Boolean functions. BDDs have recently attracted much attention because they satisfy the above requirements, to represent and manipulate Boolean functions efficiently.

As the study on BDDs has progressed, the range of applications has broadened. Coudert et al. [3] and Minato [4] have proposed methods to represent sets of combinations using BDDs. In these methods, a set of combinations can be represented implicitly using a BDD as a characteristic function of it. This means that we can treat two-level Boolean formulas, e.g. sum-of-product form, product-of-sum form and exclusive-or sum-of-product form, using BDDs, since a two-level Boolean formula can be regarded as a set of combinations. For example, a Boolean formula in sum-of-product form can be regarded as a set of products, which are combinations of literals. In this way, based on implicit set representation, we can manipulate two-level Boolean formulas efficiently by using BDDs.

We focus on BDDs representing Boolean formulas and study their expressive power. We say that a BDD representing a Boolean formula realizes a Boolean function if the formula is an expression of the function. We also discuss relations among their expressive power, *1-L* and *1-NL*. Here *1-L* (*1-NL*, respectively) is the class of languages accepted by log-space bounded on-line deterministic (nondeterministic, respectively) Turing machines.

This report is organized as follows. In Section 2, we explain how to represent sum-of-product form using BDDs, and define computational models used in this report. In Section 3, the expressive power of BDDs representing sum-of-product form is considered in terms of Ternary Decision Diagrams (TDDs) [5]. Section 4 shows relations among the class of functions expressible by polynomial size TDDs and other classes. Section 5 is a conclusion.

2 Preliminaries

2.1 Binary Decision Diagrams

A *Binary Decision Diagram (BDD)* represents a Boolean function. A BDD is a directed acyclic graph with a unique source node, called *the root node*, and two sink nodes, called *the constant nodes*. The two constant nodes are labeled by the Boolean constants 0 and 1, respectively. We denote the constant node labeled by 0, by c_0, and the constant node labeled by 1, by c_1. Non-sink nodes are called *variable nodes*. Each variable node is labeled by one of Boolean variables $\{x_1, \ldots, x_n\}$, and has two outgoing edges labeled by 0 and 1, called *the 0-edge* and *the 1-edge*, respectively. Each variable appears at most once on each path from the root node to one of the constant nodes. The order in which variables appear is consistent among all the paths. In this report, we assume that the variable order is $x_1, x_2, \ldots, x_n$.

We define the *size* of a BDD B, denoted by $|B|$, as the number of variable nodes. A set of nodes labeled by the same variable x_i is called *the i-th level*.

Given an assignment to the variables, the value of the function that a BDD represents is determined by traversing nodes from the root node to one of the constant nodes. On a variable node v

whose level is l, the edge labeled by the value of x_l is selected and it leads to the node pointed to by the edge. The value of the function is 0 if the traverse terminates at c_0, and 1 if the traverse terminates at c_1.

2.2 Binary Decision Diagrams Representing Sum-of-product Form

Coudert et al. [3] and Minato [4] have proposed methods to represent a set of combinations implicitly using a BDD. These methods enable us to represent a set of products. These method, which differ slightly from each other, are based on the idea that, on the nodes corresponding to the variable x_i, the set of products are divided into three sets; the set of products in which x_i occurs, the set of products in which $\overline{x_i}$ occurs, and the set of products in which neither x_i nor $\overline{x_i}$ occurs. These methods represent a set of products using two new variables for each x_i to distinguish these three cases. Each path from the root node to c_1 corresponds to a product.

We here explain Minato's method to represent a set of products using a BDD [4]. A product in n-variable formulas can be represented by a $2n$-bit vector $(x_1\overline{x_1}x_2\cdots x_n\overline{x_n})$, where each bit, x_i or $\overline{x_i}$, expresses whether the corresponding literal is included in the product or not. For example, $x_1\overline{x_2}x_4$ can be represented by (10010010).

Given a set P of products, the function $h(x_1,\overline{x_1},x_2,\overline{x_2},\ldots,x_n,\overline{x_n})$ is considered such that $h(x_1,\overline{x_1},x_2,\overline{x_2},\ldots,x_n,\overline{x_n}) = 1$ if and only if the product corresponding to the vector $(x_1\overline{x_1}x_2\cdots x_n\overline{x_n})$ is in P. A BDD is said to represent P if the BDD represents h. In this way, we can represent a set of products using a BDD. In this method, for each variable x_i, the nodes labeled by the positive literal x_i and the nodes labeled by the negative literal $\overline{x_i}$ are used in a BDD. This type of BDD is called *Zero-suppressed BDD (0-sup-BDD)* because of its reduction rule [4]. We assume that the variable order in a 0-sup-BDD is x_1, $\overline{x_1}$, x_2, $\overline{x_2}$, $\ldots$, x_n, $\overline{x_n}$.

A BDD representing a set of products can be regarded as representing a function in sum-of-product form since sum-of-product form can be seen as a set of products. For example, we consider a Boolean function $f = x_1\overline{x_2}x_3 + \overline{x_1}\,\overline{x_3} + x_2\overline{x_3}$. We can represent f as a set of products $\{x_1\overline{x_2}x_3, \overline{x_1}\,\overline{x_3}, x_2\overline{x_3}\}$.

2.3 Ternary Decision Diagrams

A *Ternary Decision Diagram (TDD)* [5] has proposed to represent and manipulate a set of products. A TDD is similar to a BDD, with the exception that each variable node of a TDD has an extra outgoing edge labeled by an '$*$' symbol, called *the $*$-edge (don't-care edge)*, as well as the 0-edge and the 1-edge.

A TDD can also be regarded as representing a Boolean function. Given an assignment to the variables, the value of the function that a TDD represents is determined by traversing nodes form the root node to one of the constant nodes. On a variable node v whose level is l, we select the edge labeled by the value of x_l or the $*$-edge nondeterministically. The value of the function is 1 if there exists at least one path that terminates at c_1, otherwise 0.

We consider a *family* of TDDs as a computational model which computes a family of Boolean functions. A family $\{T_n\}$ of TDDs is a sequence T_1, T_2, $\ldots$, T_n, $\ldots$ of TDDs, where T_n is a TDD representing an n-variable Boolean function. A family $\{T_n\}$ of TDDs is said to accept a language $A \subseteq \{0,1\}^*$ if and only if for each n, T_n represents the characteristic function of $A^{(n)} = A \cap \{0,1\}^n$, i.e., $x_1\cdots x_n \in A$ iff $f_n(x_1,\ldots,x_n) = 1$, where f_n is the Boolean function which T_n represents.

We define the *size* of a TDD T, denoted by $|T|$, as the number of variable nodes. We extend the definition of the size to a family $\{T_n\}$ of TDDs as follows. The size of $\{T_n\}$ is said to be $S(n)$ if for each n, $|T_n|$ is bounded by $S(n)$.

In a family of TDDs, each TDD works only on the inputs of a fixed length. Even if all the TDDs in the family are simple, a family might represent a complicated language. To avoid such families of TDDs, we define *uniform* families of TDDs, that is, families of TDDs such that intuitively it is easy to construct the n-variable TDD from n. A family $\{T_n\}$ of TDDs is log-*uniform* if the function $n \rightarrow \overline{T_n}$ is computable by an $O(\log n)$-space bounded deterministic Turing machine, where $\overline{T_n}$ is the *standard encoding* of T_n defined as follows. We define that the standard encoding $\overline{T}$ of a TDD T

consists of a set of five-tuples (v, l, e_0, e_1, e_*), where v is a variable node or a constant node, l is the level of v, e_0 is the node pointed to by the 0-edge from v, e_1 is the node pointed to by the 1-edge from v, and e_* is the node pointed to by the $*$-edge from v.

We can transform 0-sup-BDDs representing sum-of-product form into TDDs with the same or less size. Conversely, we can also transform TDDs into 0-sup-BDDs with at most twice size.

Proposition 1 *For a 0-sup-BDD Z representing a Boolean function as sum-of-product form, there exists a TDD T which represents the same function such that $|T| \leq |Z|$.*

For a TDD T, there exists a 0-sup-BDD Z which represents the same function as sum-of-product form such that $|Z| \leq 2|T|$.

Proof : Let Z be a 0-sup-BDD representing sum-of-product form. Z has no path from the root node to c_1 that includes both the 1-edges from a node labeled by x_i and from a node labeled by $\overline{x_i}$ for any x_i. If any, we can remove the path without changing the set of products represented by Z, since the path corresponds to no product (because $x_i \wedge \overline{x_i} = 0$). We can assume that a node labeled by x_i, a node labeled by $\overline{x_i}$ and their edges in Z take the form as figure 1.

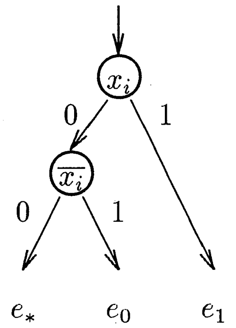

Figure 1: form of 0-sup-BDD

Then the 1-edge of a node labeled by x_i in Z corresponds to the 1-edge of a node labeled by x_i in T, the 1-edge of a node labeled by $\overline{x_i}$ in Z corresponds to the 0-edge of a node labeled by x_i in T, and the 0-edge of a node labeled by x_i or $\overline{x_i}$ in Z corresponds to the $*$-edge of a node labeled by x_i in T.

Conversely, a node labeled by x_i in T can be transformed into a couple of nodes labeled by x_i and $\overline{x_i}$ in Z. Note that for each node labeled by x_i in T, both nodes of the couple are not always needed. □

Due to this proposition, we can observe that the size of a 0-sup-BDD representing a Boolean function as sum-of-product form differs from the size of a TDD representing the same function within only constant factor. Therefore we can consider that the expressive power of 0-sup-BDDs representing sum-of-product form is the same as that of TDDs in terms of polynomial size. From now, we focus on TDDs instead of 0-sup-BDDs representing sum-of-product form.

3 Characterization of TDDs

We define the class of languages accepted by uniform families of polynomial size TDDs.

Definition 1 *U-PolyTDD is the class of languages accepted by uniform families of polynomial size TDDs.*

Each node of TDDs has three edges, the 0-edge, the 1-edge, and the $*$-edge. Corresponding to this feature, we consider restricted nondeterministic Turing machines.

We refer to a log-space bounded on-line nondeterministic Turing machine with the following restrictions on nondeterministic operations, as an *1-RNL* machine. We suppose the following restrictions.

- While it reads each single input symbol, it can use only one nondeterministic operation, i.e., there are at most two possible computations while the input head does not move.
- Either of the two computations after it reads 0 as a single input symbol, is the same as either of the two computations after it reads 1 as the input symbol.

Note that a log-space bounded on-line nondeterministic Turing machine with no restrictions may take many nondeterministic operations while it reads each single input symbol.

We designate the class of languages accepted by these restricted *1-NL* machines, by *1-RNL.*

Theorem 1 *U-PolyTDD = 1-RNL.*

Proof : ($\supseteq$) Let A be a language in *1-RNL* and M be a restricted log-space bounded on-line nondeterministic Turing machine that accepts A. We will show that for any M, there exists a uniform family $\{T_n\}$ of polynomial size TDDs that accepts A.

Let a configuration of M be a triple (h, q, u), where h is the head position of M on the input tape, q is a state of M, and u describes the contents of the work tape and the head position on the work tape.

A node of T_n corresponds to a configuration of M with an input of length n. The root node corresponds to the initial configuration $(1, q_{init}, u_{init})$ of M, where the first term indicates that the head position on the input tape of M is 1, q_{init} is the initial state of M, and u_{init} represents the binary representation of n and the head position 1 on the work tape. Let a node v of T_n correspond to a configuration (h, q, u) of M. If q is a rejecting state or an accepting state of M, v is c_0 or c_1 respectively. Otherwise, v is a variable node labeled by x_h, and the edges from v points to other nodes as follows. Let δ be the state transition function of M. The head position of the input tape may not move during the transition. To exclude such conditions, we consider δ' obtained by iterating δ until the input head moves. That is, a transition by δ' corresponds to several transitions by δ during which the input head position changes from h to $h+1$. Since M is restricted, we can assume;

$$\begin{aligned} \delta'((h,q,u),0) &= ((h+1,q_0,u_0),(h+1,q',u')) \text{ and} \\ \delta'((h,q,u),1) &= ((h+1,q_1,u_1),(h+1,q',u')) \end{aligned}$$

The 0-edge points to the node corresponding to $(h+1, q_0, u_0)$, the 1-edge points to the node corresponding to $(h+1, q_1, u_1)$, and the $*$-edge points to the node corresponding to $(h+1, q', u')$. Now T_n is obtained.

Since M is $O(\log n)$-space bounded, the number of different configurations of M is bounded by some polynomial in n. Hence the size of T_n is bounded by some polynomial in n. It is obvious that $\{T_n\}$ is log-uniform and accepts A.

($\subseteq$) Let A be a language in *U-PolyTDD*, and let $\{T_n\}$ be a uniform family of polynomial size TDDs that accepts A. We will show that for any $\{T_n\}$, we can design a restricted log-space bounded on-line nondeterministic Turing machine M that accepts A.

M knows the length n of the input without moving the head of the input tape. Since $\{T_n\}$ is log-uniform, M can generate the standard encoding $\overline{T_n}$ of T_n from n. First, M generates the root node $(v_{root}, 1, e_0, e_1, e_*)$ of $\overline{T_n}$. For an input $b_1 \cdots b_n$ ($b_1, \ldots, b_n \in \{0,1\}$), M repeats to generate a five-tuple (v, l, e_0, e_1, e_*) and select the next node until v is one of the constant nodes. As for the next node, e_{b_l} or e_* is selected nondeterministically. M accepts the input if there exists at least one computation path terminating at c_1, otherwise M rejects the input.

M can use only $O(\log n)$ space since the length of each five-tuple is bounded by $O(\log n)$, and M satisfies the above restrictions. It is obvious that M accepts A. □

4 Relations among Classes

In this section, we discuss relations among *U-PolyTDD*, *1-L* and *1-NL*. First, we show some properties on the size of TDDs.

Lemma 1 *If an n-variable function f_n is expressible by a Boolean formula in sum-of-product form in which the number of products is p, then f_n can be represented by a TDD T whose size is bounded by $n \times p$.*

Proof : In T, each path from the root node to c_1 corresponds to a product. Even if for any level, no nodes in the level can be shared, there are at most p nodes. Therefore the size of T is bounded by $n \times p$. □

As for *U-PolyTDD*, the following result is derived from this lemma.

Corollary 1 *Let A be a language. If for any n, the characteristic function f_n of $A^{(n)} = A \cap \{0,1\}^n$ is expressible by a Boolean formula in sum-of-product form in which the number of products is bounded by some polynomial in n, then $A \in$ U-PolyTDD.*

We show the relation between *U-PolyTDD* and *1-L*.

Theorem 2 *1-L $\subsetneq$ U-PolyTDD.*

Proof : We first show that *1-L* is included in *U-PolyTDD*. In the proof of Theorem 1, we let all the $*$-edges point to c_0. Then any *1-L* machine can be simulated by a uniform family $\{T_n\}$ of TDDs whose size is bounded by some polynomial in n.

Next, we show that this inclusion is strict. We define the following language;

$$A = \{ww \mid w \in \{0,1\}^*\}$$

Both A and its complement $\overline{A}$ are not included in *1-L* [6]. The characteristic function χ_{2n} of $\overline{A} \cap \{0,1\}^{2n}$ is expressible as follows.

$$\chi_{2n}(w_1 \cdots w_{2n}) = w_1\overline{w_{n+1}} + \overline{w_1}w_{n+1} + w_2\overline{w_{n+2}} + \overline{w_2}w_{n+2} \cdots + w_n\overline{w_{2n}} + \overline{w_n}w_{2n}$$

Since this formula is in sum-of-product form in which the number of products is $2n$, by Corollary 1, $\overline{A} \in$ *U-PolyTDD*. □

We discuss the relation between *U-PolyTDD* and *1-NL*. By Theorem 1 and the definition of *1-RNL*, it is obvious that *U-PolyTDD* $\subseteq$ *1-NL*.

We consider the language *TAGAP*, which is the set of the topologically arranged representations of directed acyclic graphs that have a directed path from the source node to a terminal node.

TAGAP =
$\{x_{11}x_{12} \cdots x_{1m} \cdots x_{mm} \mid (x_{ij})$ is the adjacency matrix of
a directed acyclic graph G such that G is topologically
arranged and there exists at least a path from v_1 to v_m of G.$\}$

We say that the representation of a directed acyclic graph G is *topologically arranged* if there is no edge (v_i, v_j) when $i > j$. Hartmanis et al. have shown that *TAGAP* is complete for *1-NL* under *1-L* reductions.

We show that *TAGAP* is in *U-PolyTDD*.

Lemma 2 *TAGAP $\in$ U-PolyTDD.*

Proof L:et M be a log-space bounded deterministic Turing machine. We design M to produce a family $\{T_n\}$ of TDDs that accepts *TAGAP*. For each m of the number of nodes in G, M works by the following algorithm. Here nodes of level $(i-1)m+1$ are labeled by x_{ij}.

begin
  for $i = 1, \dots, m-1$ **do**
    begin
      for $j = i+1, \dots, m$ **do**
        begin
          if $j = m$ **then** print $(v'_{ij}, (i-1)m+j, c_0, c_1, c_0)$
          else print $(v'_{ij}, (i-1)m+j, v'_{i(j+1)}, v'_{j(j+1)}, v'_{i(j+1)})$
        end
    end
end

M use only $O(\log m)$ space of its work tape and each TDD that M produces has the size of $O(m^2)$. We claim that a family $\{T_n\}$ of TDDs that M produces accepts *TAGAP*.

We prove the claim by induction on m of the number of nodes in G. The base case is obvious. In the inductive step, we assume that $T_{(i)}$ accepts $TAGAP \cap \{0,1\}^{i^2}$ for $1 \le i \le m$, i.e., $T_{(i)}$ accepts G such that G has i nodes and $G \in TAGAP$.

For $T_{(m)}$, we consider another TDD $T^1_{(m)}$ obtained by removing the nodes labeled by x_{1j} $(1 \le j \le m)$ and edges of these nodes from $T_{(m)}$. By the assumption and the above algorithm, it is clear that $T^1_{(m)}$ accepts the input if there exists a path from v_2 to v_m. That is, $T^1_{(m)}$ is isomorphic to $T_{(m-1)}$ except for labels. Similarly, another TDD $T^2_{(m)}$ can be obtained by removing the nodes labeled by x_{2j} $(2 \le j \le m)$ and edges of these nodes from $T^1_{(m)}$. $T^2_{(m)}$ is isomorphic to $T_{(m-2)}$ except for labels. We can continue this and obtain the sequence $\{T^j_{(m)}\}$, where $T^j_{(m)}$ is isomorphic to $T_{(m-j)}$ except for labels.

To construct $T_{(m+1)}$, we have only to test each variable x_{1j} $(1 \le j \le m+1)$ and connect the edges from the node labeled by the variable to other nodes appropriately. If $x_{1j} = 1$, there are two possibilities; one is that the edge from v_1 to v_j in G is included in the very path that makes G in *TAGAP*, the other is that the edge from v_1 to v_j is not included in the path. For the former case, we connect the 1-edge to the root node of $T^j_{(m+1)}$. For the latter case, we connect the $*$-edge to $v'_{1(j+1)}$ to test another edge from v_1 in G. The 0-edge also points to $v'_{1(j+1)}$. Note that all the 0-edges may point to c_0. □

By Lemma 2, if *U-PolyTDD* is closed under *1-L* reductions, *U-PolyTDD* = *1-NL*. But it is not clear whether *U-PolyTDD* is closed under *1-L* reductions or not.

We propose a reduction which is an *1-L* reduction restricted on relation between the length of the input and the length of the output. We show that *U-PolyTDD* is closed under the reductions. *Length restricted 1-L reductions*, denoted by $\le^l_{1\text{-}L}$, are *1-L* reductions restricted as follows.

Length restriction

For each i, the i-th output symbol depends only on the string from the *from*(i)-th input symbol to *to*(i)-th input symbol. Here *from* and *to* satisfy the following.

$$from(i) \le to(i) < from(i+1).$$

This restriction means that the length of the output depends only on the length of the input. It also means that to produce a single output symbol, a transducer M_r needs to read at least one unread input symbol.

Lemma 3 *U-PolyTDD is closed under* $\le^l_{1\text{-}L}$.

Sketch of Proof L:et A and B be languages. Assume that $A \le^l_{1\text{-}L} B$, and that $B \in$ *U-PolyTDD*. There exist two machines: an *1-L* generator M which generates a family $\{T_n\}$ of TDDs accepting B, and a restricted *1-L* transducer M_r which reduces A to B. We will construct a machine M' which generates a family $\{T'_n\}$ of TDDs that accepts A.

At first, M' on input n simulates M_r, and knows the length $g(n)$ of the output that M_r produces on input n. Because of the restriction mentioned above, $g(n)$ depends only on n.

Next M' simulates M on input $g(n)$ and generates $T_{g(n)}$. Whenever M' needs to know the value of the i-th input symbol of M, M' simulates M_r on the input string from *from*(i) to *to*(i). M' can produce the simulation of M_r as a form of a TDD because of Theorem 2. □

We define a new operation on languages, called *stretch*, and give more information on the relation between *U-PolyTDD* and *1-NL*.

Definition 2 *Let A be a language, and let $p(n)$ be some polynomial in n. stretch$^p(A)$ is defined as follows.*

$$\begin{aligned}&\mathit{stretch}^p(A) = \\ &\quad \{\ x_1\sharp y_{11}\cdots y_{1p(n)}\sharp x_2\sharp y_{21}\cdots y_{2p(n)}\sharp\cdots\sharp x_n\sharp y_{n1}\cdots y_{np(n)} \mid x_i = y_{i1} = \cdots = y_{ip(n)} \\ &\quad \textit{for } 1 \le i \le n, \textit{ and } x_1x_2\cdots x_n \in A \textit{ for each } n\ \}\end{aligned}$$

Let $\mathcal{C}$ be a class of languages. The class $\mathcal{R}^{st}(\mathcal{C})$ is defined as follows.

$$\mathcal{R}^{st}(\mathcal{C}) = \{A \mid \mathit{stretch}^p(A) \in \mathcal{C} \textit{ for some } p\}$$

Using this operation, together with the above results, the following result is obtained.

Theorem 3 $\mathcal{R}^{st}$(*U-PolyTDD*) = *1-NL*.

Proof : ($\subseteq$) It is obvious since for any A, if *stretch*(A) is in *1-NL*, then $A \in$ *1-NL*

($\supseteq$) For any $A \in$ *1-NL*, A is $\le_{\mathit{1\text{-}L}}$-reducible to *TAGAP*. We can show that *stretch*$^p(A)$ is $\le^l_{\mathit{1\text{-}L}}$-reducible to *TAGAP* for some polynomial p.

Let A be $\le_{\mathit{1\text{-}L}}$-reducible to *TAGAP*. We assume that A is reduced to *TAGAP* using the technique in [6]. In the technique, for any n, $A^{(n)} = A \cap \{0,1\}^n$ is $\le_{\mathit{1\text{-}L}}$-reducible to *TAGAP* whose length is some polynomial $q(n)$, where q depends only on n, and each bit of $TAGAP \cap \{0,1\}^{q(n)}$ depends only on a single bit of $A^{(n)}$. For $p(n) \ge q(n)$, there exists a restricted *1-L* machine M_r which can reduce *stretch*$^p(A)$ to *TAGAP*. From Lemma 2 and 3, $\mathcal{R}^{st}$(*U-PolyTDD*) $\supseteq$ *1-NL*. □

5 Conclusion

We have characterized the expressive power of BDDs representing sum-of-product form by restricted *1-NL* machines. The restrictions on *1-NL* machines that we suppose in this report do not matter in the case of *off-line* Turing machines. If we omit the restriction on appearance of variables in TDDs, the expressive power of polynomial size TDDs is equal to *NL*.

We have also discussed relations among *U-PolyTDD* , *1-L*, and *1-NL*. *U-PolyTDD* includes *1-L* strictly, and *U-PolyTDD* is included in *1-NL*. If *U-PolyTDD* is closed under *1-L* reductions, *U-PolyTDD* = *1-NL*. It is interesting whether this inclusion is strict or not.

Although we have considered only the case that a BDD is regarded as representing sum-of-product form, the same BDD can also be regarded as representing other two-level logic forms, such as product-of-sum form or ring-sum-of-product form. In these cases, similar results can be obtained by introducing the classes, *co-1-NL* or *1-*$\oplus L$.

References

[1] S. B. Akers. Binary decision diagrams. *IEEE Trans. Comput.*, Vol. C-27, No. 6, pp. 506–516, Jun 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] O. Coudert and J. C. Madre. Implicit and incremental computation of primes and essential primes of Boolean functions. In *Proc. 29th ACM/IEEE DAC*, pp. 36–39, Jun 1992.

[4] S. Minato. Zero-suppressed BDDs for set manipulation in combinatorial problems. In *Proc. of ACM/IEEE DAC '93*, pp. 272–277, Jun 1993.

[5] Kouichi Yasuoka. Ternary decision diagrams as a representation of sets of products. In *RIMS koukyuroku, Kyoto University*, 1995. (to appear).

[6] J. Hartmanis and S. Mahaney. Langueges simultaneously complete for one-way and two-way log-tape automata. *SIAM J. Comput.*, Vol. 10, No. 2, pp. 383–390, May 1981.