PAPER *Special Section on VLSI Design and CAD Algorithms*

Manipulation of Large-Scale Polynomials Using BMDs

Dror ROTTER[†], ***Nonmember***, **Kiyoharu HAMAGUCHI**[††], **Shin-ichi MINATO**[†††], ***and*** **Shuzo YAJIMA**[††††], ***Members***

SUMMARY Minato has proposed a canonical representation for polynomial functions using zero-suppressed binary decision diagrams (ZBDDs). In this paper, we extend binary moment diagrams (BMDs) proposed by Bryant and Chen to handle variables with degrees higher than 1. The experimental results show that this approach is much more efficient than the previous ZBDDs' approach. The proposed approach is expected to be useful for various problems, in particular, for computer algebra.

key words: *zero-suppressed binary decision diagrams, binary moment diagrams, polynomials*

1. Introduction

Manipulating polynomials is one of the key operations in computer algebra. Minato proposed a method for representing polynomials based on Zero-suppressed BDDs (ZBDDs [2]), which can manipulate much larger polynomials than before. He showed that the signal probability of combinational circuits can be computed effectively using this ZBDD-based representation. This ZBDD-based approach is also expected to be useful in various problems such as probablistic fault simulation, power estimation, formal verification of arithmetic-level descriptions. In particular, an experiment in [3] suggests that his approach is efficient in computation of signal probability for logic circuits, which requires manipulation of large-scale polynomials.

In this paper, we propose a new way of handling large-scale polynomials using binary moment diagrams (BMDs [4]), which is much more efficient than the ZBDD-based approach. BMDs were introduced by Bryant and Chen as a representation of linear functions, that is, polynomials where the degree of each variable is at most 1. We use an idea similar to Minato [3] to extend BMDs for representing general polynomials. This BMD-based representation of polynomials is unique and compact as well as that of linear functions. The difference between Minato's approach and ours is that ZBDDs construct graph nodes to represent numbers, whereas $*$BMDs associate weights with the edges to represent numbers. This reduces the sizes of graphs to be manipulated, as we will discuss the detail in Sect. 5. We develop efficient algorithms for polynomial calculus based on BMD operations. Our experiments are still preliminary, but they show that our new approach is stronger than the previous approach. For example, for $\prod_{k=1}^{n}(x^k+1)^8$, we were able to generate $*$BMD for $n = 6500$, whereas the ZBDD-based program was able to generate ZBDD only for $n = 12$. Furthermore, for this polynomial we got quadratic run-time complexity, whereas with the ZBDD-based program the run-time complexity is exponential. The overall performance for the other examples was also much better.

This paper is organized as follows: Sect. 2 gives the basic properties of $*$BMDs. Section 3 presents a method for representing polynomials with $*$BMDs. Section 4 explains the operation algorithms for polynomial calculus, in particular, the multiplication algorithm. Section 5 compares the BMD-based approach with the ZBDD-based approach. In Sect. 6 we show experimental results.

2. Preliminaries

In this section, we summarize the definition and the property on Binary Moment Diagram (BMD) and Multiplicative BMD ($*$BMD). The details have been described in [4]. In the following, $\vee$ and $\wedge$ represent Boolean disjuntion and conjunction respectively, and $+$ and $\cdot$ represent arithmetic addition and multiplication respectively.

2.1 Binary Moment Diagram

A Binary Moment Diagram (BMD) is a directed acyclic graph that represents a function from $\{0,1\}^n$ to the set of integers. Each node of BMD is either of a variable node or a constant node. The outdegrees of variable nodes and constant nodes are 2 and 0 respectively. Each constant node is labeled by an integer. The integer labeled to any constant node is different from the others. The nodes whose indegrees are one, are called root

Manuscript received February 20, 1997.
Manuscript revised May 6, 1997.

[†]The author is with the Graduate School of Information Science, Nara Institute of Science and Technology, Ikoma-shi, 630–01 Japan.
[††]The author is with the Department of Informatics and Mathematical Science, Graduate School of Engineering Science, Osaka University, Toyonaka-shi, 560 Japan.
[†††]The author is with NTT System Electronics Laboratories, Atsugi-shi, 243–01 Japan.
[††††]The author is with the Faculty of Informatics, Kansai University, Takatsuki-shi, 560–11 Japan.

nodes. We represent the set of variable nodes by V_V and the set of constant nodes by V_C respectively.

For each variable node $v \in V_V$, We define $Var(v)$, $Low(v)$ and $High(v)$. $Var(v) \in \{x_1, x_2, \ldots, x_n\}$ is the variable labeled to the node v. $Low(v)$ and $High(v)$ represent nodes linked by directed edges going out of v. We call the edges $(v, Low(v))$ and $(v, High(v))$ 0-edges and 1-edges respectively. We assume a total ordering among the Boolean variables $x_1, x_2, \ldots, x_n$. The appearance of the variables strictly obeys this ordering along any path from roots to contants nodes. The level of x_i represented by $level(i)$ is the order of the variable. If $level(i) < level(j)$, then i is called higher than j, and j is called lower than i.

Let N be the set of integers. We assume a function $f : \{0,1\}^n \rightarrow N$. We represent, by $f_{\bar{x}_i}$ and f_{x_i}, the functions substituted for the variable x_i with 0 and 1 respectively. The moment decomposition of a funtion for a variable x_i is defined as follows:

$$f = f_{\bar{x}_i} + x_i \cdot f_{\dot{x}_i} \quad (f_{\dot{x}_i} = f_{x_i} - f_{\bar{x}_i})$$

The functions $f_{\bar{x}_i}$ and $f_{\dot{x}_i} = f_{x_i} - f_{\bar{x}_i}$ are respectively called the constant moment and the linear moment of f for the variable x_i.

Based on the definition of moment decomposition, each node of BMD represents a funtion. Let $Val(v)$ represent the integer labeled to a constant node v. Suppose that ϕ is an assignment of Boolean values to variables. Then v represents the funtion $f\langle v \rangle$ which is recursively defined as follows:

$$f\langle v \rangle = \begin{cases} Val(v) & v \text{ is a constant node.} \\ f\langle Low(v) \rangle & \phi(Var(v)) = 0 \\ f\langle Low(v) \rangle + f\langle High(v) \rangle & \phi(Var(v)) = 1 \end{cases}$$

Two nodes $v_1(Var(v_1) = x_i)$ and $v_2(Var(v_2) = x_j)$ are equivalent if and only if the following conditions are all satisfied.

$$\begin{array}{rcl} level(i) & = & level(j) \\ Low(v_1) & = & Low(v_2) \\ High(v_1) & = & High(v_2) \end{array}$$

Redirecting all the edges toward v_2 to v_1, we can delete v_2 without changing the funtion. A node whose $High(v)$ is a contant node labeled by 0, is called redundant. Redirecting all the edges toward v to $Low(v)$, we can delete v without changing the function. It is known that BMDs without equivalent nodes are unique, and so are BMDs without equivalent nodes and redundant nodes. The important property is that BMDs can uniquely represent an arbitrary linear functions, that is, polynomials such that the degrees of the variables are all one. The number of the nodes of a BMD is called the size of the BMD.

In order to reduce the number of nodes, we can extend BMDs so that edges have weights. Suppose that an edge has an integer weight w and enters a node v. Then the weight w is multiplied to the function represented by v. Let v_1 and v_2 represent functions $f\langle v_1 \rangle$ and $f\langle v_2 \rangle$ respectively. If either of the following conditions are satisfied for an integer w, then the two nodes v_1 and v_2 can be shared.

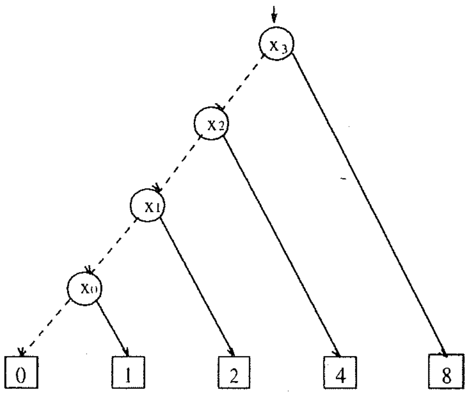


Fig. 1 $*$BMD representing $\sum_{i=0}^{3} 2^i x_i$.

$$\begin{array}{l} w \cdot f\langle v_1 \rangle = f\langle v_2 \rangle \\ f\langle v_1 \rangle = w \cdot f\langle v_2 \rangle \end{array} \tag{1}$$

In order to maintain uniqueness of representations, we need to normalize edge weights. In this paper, we use one of the normalized forms Bryant and Chen proposed [5] shown in the following:

> Let w_0 and w_1 be weights of the 0-edge and the 1-edge of a node v respectively. We replace the w_0 and w_1 with w_0/w and w_1/w, where w is the greatest common divisor of w_0 and w_1. We give w as a weight to the edge toward the node v.

We regard the weights of edges incident to constant nodes as the constant values labeled to the nodes. Two pairs $\langle w_1, v_1 \rangle$ and $\langle w_2, v_2 \rangle$ represent the same function if and only if $w_1 = w_2$ and $v_1 = v_2$. We call the BMDs satisfying the above rule as Multiplicative BMD ($*$BMDs). It is known that $*$BMD can represent addition, multiplication and exponentiation with graphs whose size is linear to the number of inputs. Figure 1 shows $*$BMD representing $\sum_{i=0}^{3} 2^i x_i$.

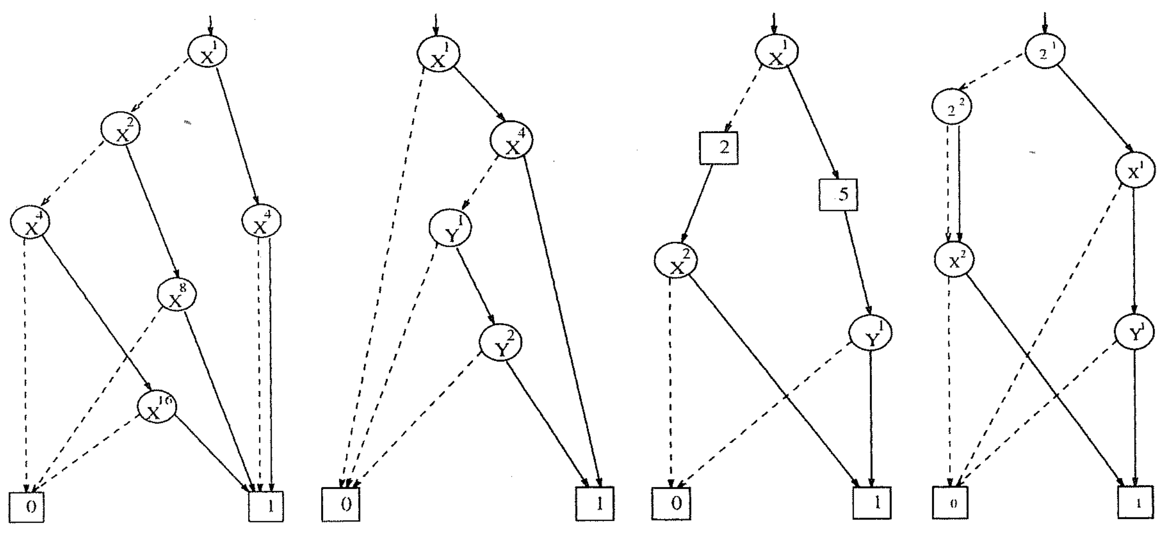
The solid lines represent 1-edges and the dotted lines represent 0-edges. The numbers in the boxes are weights to the edges.

2.2 Operations over Binary Moment Diagrams

Let f and g be functions from Boolean vectors to integers. Then,

$$\begin{array}{rcl} f + g & = & f_{\bar{x}} + x \cdot f_{\dot{x}} + g_{\bar{x}} + x \cdot g_{\dot{x}} \\ & = & (f_{\bar{x}} + g_{\bar{x}}) + x \cdot (f_{\dot{x}} + g_{\dot{x}}). \end{array}$$

This means that we can compute the constant moment and the linear moment of the addition $f+g$ as $(f_{\bar{x}} + g_{\bar{x}})$



(a) $x^4x^{16} + x^2x^8 + x^1x^4 + x^1$ (b) $x^1y^1y^2 + x^1x^4$ (c) $2x^2 + 5x^1y^1$ (d) $x^2 + 2^2x^2 + 2^1x^1y^1$

Fig. 2 *BMDs (a-c) and ZBDD (d) for polynomials.

and $(f_{\dot{x}} + g_{\dot{x}})$ respectively. Using this property, we can compute $f + g$ recursively. We can also derive the following property on multiplication.

$$f \cdot g = f_{\bar{x}} \cdot g_{\bar{x}} + x(f_{\bar{x}} \cdot g_{\dot{x}} + f_{\dot{x}} \cdot g_{\bar{x}}) + x^2 f_{\dot{x}} \cdot g_{\dot{x}}.$$

If we define linear product then obviously $x^2 = x$. In order to emphasize that we consider multiplication under $x^2 = x$, we use $\hat{\cdot}$ instead of $\cdot$. Then,

$$f \,\hat{\cdot}\, g = f_{\bar{x}} \,\hat{\cdot}\, g_{\bar{x}} + x(f_{\bar{x}} \,\hat{\cdot}\, g_{\dot{x}} + f_{\dot{x}} \,\hat{\cdot}\, g_{\bar{x}} + f_{\dot{x}} \,\hat{\cdot}\, g_{\dot{x}}).$$

This means that we can compute the constant moment and the linear moment of $f\hat{\cdot}g$ as $f_{\bar{x}}\hat{\cdot}g_{\bar{x}}$ and $f_{\bar{x}}\hat{\cdot}g_{\dot{x}}+ f_{\dot{x}}\hat{\cdot}g_{\bar{x}}+ f_{\dot{x}}\hat{\cdot}g_{\dot{x}}$ respectively. Again we can compute $f\hat{\cdot}g$ recursively.

3. Representation of Polynomials

We show a way to deal with polynomials. Here we consider only positive integer numbers for the degrees. This can be easily extended to handle negative integers.

The basic idea is that an integer number can be written as a sum of 2's exponential numbers by using binary coding. Namely, a variable x^k can be broken down into:

$$x^k = x^{(k_1+k_2+\ldots+k_m)} = x^{k_1}x^{k_2}\cdots x^{k_m},$$

where $k_1, k_2, \ldots, k_m$ are different 2's exponential numbers. In this way, we can represent a term x^k as a combination of n items $x^1, x^2, x^4, x^8, \ldots, x^{2^{n-1}}$, according to the binary representation of k $(0 < k < 2^n)$.

For example, a polynomial $x^{20} + x^{10} + x^5 + x$ can be written as $x^4x^{16} + x^2x^8 + x^1x^4 + x^1$. If we assume each of these items x^1, x^2, x^4, x^8, and $x^{2^{n-1}}$ as a distinct variable, then we can regard $x^4x^{16} + x^2x^8 + x^1x^4 + x^1$ as a linear function. Since binary encodings of degrees

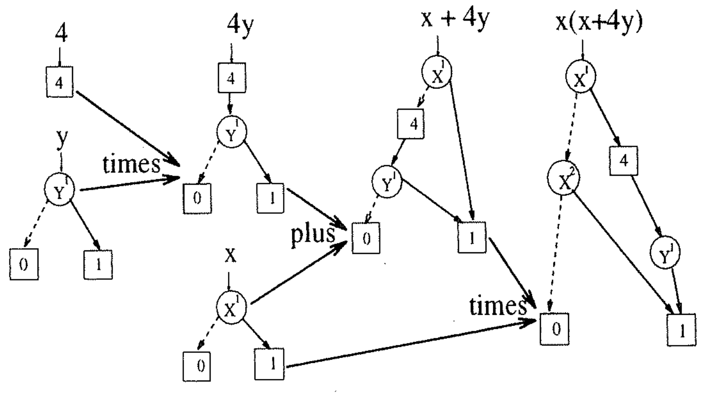


Fig. 3 Generation of *BMD for arithmetic expressions.

are unique, we can represent the polynomial uniquely with a BMD or *BMD. This formula can be represented by using *BMDs, as shown in Fig. 2 (a). In this example, we give the highest order to x^1 and lower orders to the other variables. the descending ordering. As is described in Sect. 4.1, this ordering is convenient in performing arithmetic operations.

When we deal with more than one sort of variables, such as x^i, y^j and z^k, we decompose them as $x^1, x^2, x^4, \ldots, y^1, y^2, y^4, \ldots$, and $z^1, z^2, z^4, \ldots$ Fig. 2 (b) and Fig. 2 (c), shows examples with two sorts of variables.

4. Manipulation of Polynomials with *BMDs

In this section we describe the algorithms for *BMDs. Polynomials can be manipulated by arithmetic operations, such as addition, subtraction, and multiplication. Based on this knowledge, we first generate *BMDs for trivial formulas which are single variables or constants, and then apply those arithmetic operations to construct more complicated polynomials. An example is shown Fig. 3. To generate a ∗BMD for the formula $x^2 + 4xy$ from arithmetic expression $x \times (x+4y)$, we first generate graphs for "x," "y" and "4," then apply some arithmetic operations to the expression.

4.1 Algorithms

- Addition $F+G$: Same as usual ∗BMD operations.
- Multiplication $F \times G$:
  Needs to handle variables with degrees. Based on the following decomposition rules, the result is constructed recursively. TOP is the top variable of F and G. F_0 (F_1) is $F_{TOP=0}$ ($F_{TOP=1} - F_{TOP=0}$).

  [Decomposition rules]

  1. If the top variables of F and G are identical
  $$F \cdot G = F_0 \cdot G_0 + TOP(F_0 \cdot G_1 + F_1 \cdot G_0) + TOP^2 F_1 \cdot G_1$$
  2. Otherwise
  $$F \cdot G = F_0 \cdot G_0 + TOP(F_0 \cdot G_1 + F_1 \cdot G_0 + F_1 \cdot G_1)$$

  The detail of the algorithm is similar to those for ∗BMD [4]. Firstly, the graphs for F and G are traversed to generate F_0, F_1, G_0 and G_1. Then, recursively $F_0 \cdot G_0$, $F_0 \cdot G_1$, $F_1 \cdot G_0$ and $F_1 \cdot G_1$ are computed. Finally, the graph for $F \cdot G$ are constructed according to the appropriate decomposition rule, by using the ∗BMD algorithms for multiplication and addition.

  Polynomials are represented as combinations of x^k ∗BMD nodes, and are most likely to use x^1 more often than variables with higher degrees. Thus, the algorithm is performed efficiently when the variables are ordered as $x^1, x^2, x^4, x^8, \ldots$; that is, $(x^k)^2$ is always next to x^k.

The time complexity is very similar to the ∗BMD complexity [4]. Although the proposed algorithm handles TOP^2, both of the multiplication algorithms for the original ∗BMD operations and for those in this paper require, at each step, up to 6 recursive calls, plus up to 3 calls to the addition algorithm. The number of the calls to the addition algorithm may blow up exponentially, as is mentioned in [4]. Since the execution of the algorithm is accelerated by maintaining a hash table that stores the result of recent operations, we can expect to avoid this exponential blow-up usually, as has been observed for OBDDs [1] or ∗BMDs. We also remark that the time complexity are not always related to the resultant size of a ∗BMD after an operation. In some cases, the size of ∗BMD is linear, but the time complexity can be exponential [4].

4.2 Example

We show an example of polynomial multiplication. Suppose that $F = x^5$ and $G = x^3 + 2x$. The graphs for F and G are shown in Fig. 4. For this case the top variables are identical, therefore we use decomposition rule 1, and we need to obtain the graphs for:

$$F_0 \cdot G_0 = 0$$
$$F_0 \cdot G_1 = 0$$
$$F_1 \cdot G_0 = 0$$
$$F_1 \cdot G_1 = x^4(x^2 + 2)$$

In order to compute $F_1 \cdot G_1$ (shown in Fig. 5), we use decomposition rule 2, because the root variables are different. Here we need to obtain the graphs for:

$$F_{1_0} \cdot G_{1_0} = 2x^4$$
$$F_{1_0} \cdot G_{1_1} = x^4$$
$$F_{1_1} \cdot G_{1_0} = 0$$
$$F_{1_1} \cdot G_{1_1} = 0$$

The construction of these graphs are straightforward. Then we obtain by decomposition rule 2:

$$F_1 \cdot G_1 = 2x^4 + x^2(x^4 + 0 + 0) = 2x^4 + x^2 x^4,$$

With this result, we move back to decomposition rule 1:

$$F \cdot G = 0 + x^1(0 + 0) + x^2(2x^4 + x^2 x^4)$$

We have to perform a multiplication of F_2 and G_2 shown in Fig. 6. We have the same top variable x^2,

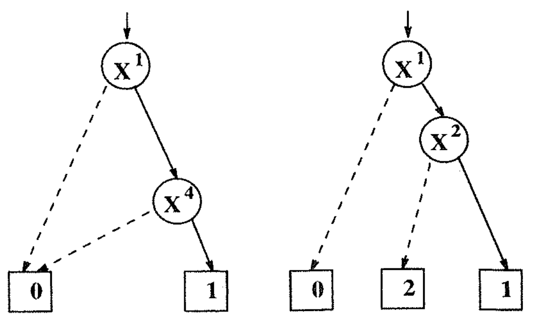


Fig. 4 $F = x^5$, $G = x^3 + 2x$.

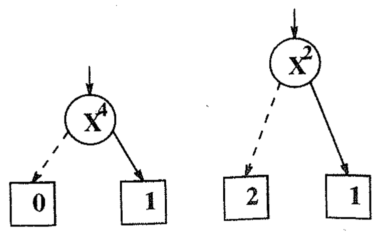


Fig. 5 $F_1 = x^4$, $G_1 = x^2 + 2$.

for $F_2 = x^2$ and $G_2 = 2x^4 + x^2x^4$, therefore we use decomposition rule 1, and we have:

$$F_{2_0} \cdot G_{2_0} = 0$$
$$F_{2_0} \cdot G_{2_1} = 0$$
$$F_{2_1} \cdot G_{2_0} = 2x^4$$
$$F_{2_1} \cdot G_{2_1} = x^4$$

Thus:

$$F_2 \cdot G_2 = 0 + x^2(0 + 2x^4) + x^4x^4,$$

The term x^4x^4 gives us x^8. Then, we can obtain the graph for

$$F_2 \cdot G_2 = 0 + x^2 \cdot 2x^4 + x^8 = 2x^6 + x^8,$$

as is shown in Fig. 7. It is easy to see that this becomes the result of $F \cdot G$.

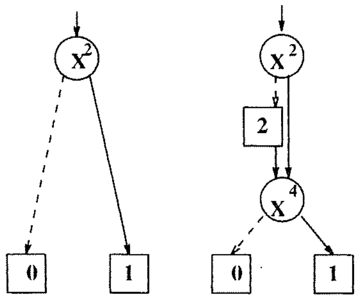


Fig. 6 $F_2 = x^2, \quad G_2 = 2x^4 + x^2x^4.$

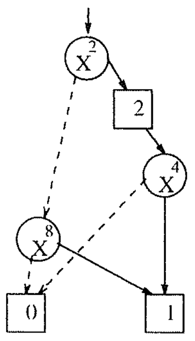


Fig. 7 $F_2G_2 = FG = 2x^6 + x^8.$

5. *BMDs and ZBDDs

In this section, we compare the BMD-based approach with the ZBDD-based approach shown in [3]. The difference lies in how to represent coefficients of polynomials. In the ZBDD-based approach, the coefficients such as 2 or 5 in Fig. 2 (c) are also represented by using vertices instead of weights to edges. We can observe that any integer c can be represented as follows:

$$c = 2^{k_1} + 2^{k_2} + \cdots + 2^{k_m},$$

where $k_1, k_2, \ldots, k_m$ are different positive integer numbers. Assuming each of 2^{k_i} as a distinct variable, we can regard any polynomial as a linear function whose coefficients are all 1. The *BMD representing this linear function is equivalent to the ZBDD representing the same function. More precisely, the ZBDD omits the weights, which are all 1's, to the edges. The algorithm for manipulating the ZBDDs is also similar. Fig. 2 (d) shows the ZBDD of Fig. 2 (c). Although whether it is always the case is not known, this way of representing integers usually causes very complicated operations over graphs. For example, in the BMD-based approach, the computation 1000×1000 is performed over two graphs only with weight 1000, and the multiplication is done by performing only one arithmetic multiplication. In the ZBDD-based approach, however, the computation is performed over two larger graphs. Besides, the sizes of ZBDDs depend on which integers should be handled. For example, in the BMD-based approach, the number "1,000,000,000" is represented as one terminal graph node with weight "1,000,000,000." Whereas in the ZBDD-based approach, 15 nodes are needed to represent this number. And also, handling polynomials with many coefficients causes complex graph manipulation. In general, we can expect better or at least more stable performance from the proposed BMD-based approach.

6. Experimental Results

Unfortunately, it is not easy to analyze the time complexity of the proposed algorithm, because of the involved recursive calls and the hashing of the intermedi-

Table 1 Results for polynomials (1).

	x^n		n^2x^n		$\Sigma_{k=0}^{n} x^k$		$\Sigma_{k=1}^{n}(k \times x^k)$		$(x+1)^n$	
n	#node	time(s)	#node	time(s)	#node	time(s)	#node	time(s)	#node	time(s)
50	4	0.1	4	0.1	11	0.2	51	0.3	51	2.6
100	4	0.1	4	0.1	13	0.3	101	0.7	101	15.0
150	5	0.1	5	0.1	15	0.5	151	1.1	151	63.0
200	4	0.1	4	0.1	15	0.6	201	1.6	201	141.4
250	7	0.1	7	0.1	15	0.8	251	2.2	251	344.9
300	5	0.1	5	0.1	17	0.9	301	2.7	301	743.2
350	7	0.1	7	0.1	17	1.1	351	3.5	351	1188.5
400	4	0.1	4	0.1	17	1.3	401	4.0	401	1747.2
450	5	0.1	5	0.1	17	1.5	451	4.8	451	2690.5

Table 2 Results for polynomials (2).

	$\prod_{k=1}^{n}(x_k+1)$		$\prod_{k=1}^{n}(x_k+k)$		$\prod_{k=1}^{n}(x_k+1)^4$		$\prod_{k=1}^{n}(x_k+1)^8$	
n	#node	time(s)	#node	time(s)	#node	time(s)	#node	time(s)
500	501	3.2	501	3.2	2001	38	4001	130
1000	1001	12.4	1001	12.5	4001	151	8001	481
1500	1501	29.5	1501	29.5	6001	344	12001	1053
2000	2001	55.2	2001	55.2	8001	617	16001	1849
2500	2501	90.2	2501	89.9	10001	971	20001	2868
3000	3001	134.7	3001	134.5	12001	1403	24001	4135
3500	3501	188.9	3501	188.7	14001	1915	28001	6001
4000	4001	253.0	4001	252.8	16001	2507	32001	7610
4500	4501	326.4	4501	326.0	18001	3180	36001	9164
5000	5001	409.6	5001	409.5	20001	3933	40001	11708
5500	5501	502.3	5501	502.4	22001	4763	44001	13649
6000	6001	606.1	6001	624.1	24001	5675	48001	16233
6500	6501	732.4	6501	729.6	26001	6666	52001	19203
7000	7001	806.7	7001	818.1	28001	7737	($\star$)	-

time(s)	time in seconds	machine	SPARCstation10/51
#node	number of nodes	($\star$)	Memory overflow (495 MB)

Table 3 Comparison results.

	$(x+1)^n$			
	ZBDD		$*$BMD	
n	#node	time(s)	#node	time(s)
1	1	0.1	2	0.1
2	4	0.1	3	0.1
3	5	0.1	4	0.1
5	10	0.1	6	0.1
10	23	0.1	11	0.1
20	69	0.3	21	0.4
30	150	1.0	31	0.9
50	346	4.7	51	2.6
100	1209	24.5	101	15.0
200	4231	159.8	201	141.4
255	6690	338.3	256	380.4

ate results. We evaluated the algorithm through experiments.

Based on the techniques described above, we modified the $*$BMD package for manipulation of polynomials. The $*$BMD package is implemented in the C language. Our program requires 76 bytes per node. We performed experiments on SPARCstation 10/51 (SunOS 4.1.4). We constructed $*$BMDs for the polynomials tested by Minato[3] as ZBDDs, such as $(x+1)^n$ and $\prod_{k=1}^{n}(x^k+1)$.

As shown in Table 1 and Table 2, within a feasible time and space, we can generate $*$BMDs for extremely large-scale polynomials. The column **#node** shows the sizes of *resultant* $*$BMDs. We remark that, in the intermediate steps of constructing $*$BMDs, much more of nodes are required. And also, as we mentioned in Sect. 4.1, the resultant sizes of $*$BMDs are not directly related to the runtime.

The degrees of the polynomials that we have succeeded to handle are generally much higher than achieved by Minato. For example, for $\prod_{k=1}^{n}(x^k+1)^8$, we are able to generate $*$BMD for $n = 6500$ with quadratic run-time complexity, whereas the ZBDD-based program has generated ZBDD only for $n = 12$ with exponential run-time complexity (Table 2 and Table 3).

In order to check on the other polynomials, we performed experiments with the same degree of polynomials for ZBDD and $*$BMD. We used a ZBDD-based program for manipulating polynomials[3]. The examples that we tested are rather artificial, but the results shown in Table 3 and Table 4, suggest that the overall performance of $*$BMD-based approach is better than ZBDD-based approach. The result for $(x+1)^n$ shows O (n^3) in the run-time complexity, and the performance is just as good as that by the ZBDD-based approach. From the results for $\prod_{k=1}^{n}(x_k+1)$ and the others, however, we can observe that the proposed $*$BMD-based approach usually handles polynomials with many different coefficients much more efficiently, which we can expect to be the case when we manipulate large-scale polynomials.

7. Conclusion

We have proposed a useful extension of the $*$BMDs to handle higher-degree polynomials, and have described efficient algorithms for manipulating those polynomials. Our experimental results outperform the ZBDD-based approach in the overall performance. We can generate $*$BMDs for extremely large-scale polynomials within a feasible time and space. We can determine that our representation is more efficient in general. An important feature of our representation is that it maintains uniqueness, i.e., the canonical form of polynomial under a fixed variable ordering.

In future, we would like to evaluate our method over real application such as computing signal probability in logic circuits. The probability can be ex-

Table 4 Comparison results.

	$\prod_{k=1}^{n}(x_k+1)$				$\prod_{k=1}^{n}(x_k+k)$				$\prod_{k=1}^{n}(x_k+1)^4$				$\prod_{k=1}^{n}(x_k+1)^8$			
	ZBDD		*BMD		ZBDD		*BMD		ZBDD		*BMD		ZBDD		*BMD	
n	#nd	t(s)	#nd	t(s)	#nd	t(s)	#nd	t(s)	#nd	t(s)	#nd	t(s)	#nd	t(s)	#nd	t(s)
1	1	0.1	2	0.1	1	0.1	2	0.1	7	0.1	5	0.1	15	0.1	9	0.1
2	2	0.1	3	0.1	4	0.1	3	0.1	25	0.1	9	0.1	109	0.1	17	0.1
3	3	0.1	4	0.1	10	0.1	4	0.1	70	0.1	13	0.1	481	0.3	25	0.2
4	4	0.1	5	0.1	17	0.1	5	0.1	145	0.1	17	0.1	1553	1.7	33	0.3
5	5	0.1	6	0.1	32	0.1	6	0.1	264	0.1	21	0.1	3862	7.3	41	0.4
6	6	0.1	7	0.1	59	0.1	7	0.1	451	0.2	25	0.1	8069	24.1	49	0.5
7	7	0.1	8	0.1	123	0.1	8	0.1	709	0.3	29	0.2	15099	69.8	57	0.6
8	8	0.1	9	0.1	199	0.1	9	0.1	1056	0.5	33	0.2	26279	120.4	65	0.6
9	9	0.1	10	0.1	331	0.2	10	0.1	1499	1.0	37	0.2	42822	227.1	73	0.7
10	10	0.1	11	0.1	619	0.7	11	0.1	2053	1.4	41	0.2	65916	576.7	81	0.8
11	11	0.1	12	0.1	1131	1.4	12	0.1	2730	2.0	45	0.2	97137	785.7	89	0.9
12	12	0.1	13	0.1	1866	3.7	13	0.1	3575	4.7	49	0.3	138966	1160.0	97	0.9
13	13	0.1	14	0.1	3334	5.7	14	0.1	4586	5.8	53	0.3	(⋆)	-	105	1.0
15	15	0.1	16	0.1	9338	17.1	16	0.1	7151	8.6	61	0.3	(⋆)	-	121	1.2
20	20	0.1	21	0.1	(⋆)	-	21	0.1	17374	43.3	81	0.5	(⋆)	-	161	1.6

t(s)	time in seconds	machine	SPARCstation10/51
#nd	number of nodes	(⋆)	Memory overflow (495 MB)

pressed exactly in polynomials using probabilistic variables. Another direction will be the integration of this method with computer algebra system.

References

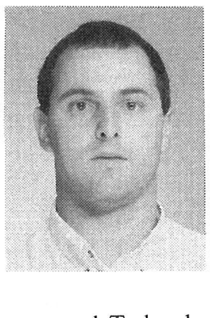
[1] R.E. Bryant, "Graph-based algorithms for Boolean function manipulation," IEEE Trans. Comput., vol.C-35, no.8, pp.677–691, Aug. 1986.

[2] S. Minato, "Zero-Suppressed BDDs for set manipulation in combinatorial problems," ACM/IEEE DAC '93, pp.618–624, June 1993.

[3] S. Minato, "Implicit manipulation of polynomials using zero-suppressed BDDs," IEEE/ACM ED&TC'95, pp.449–454, March 1995.

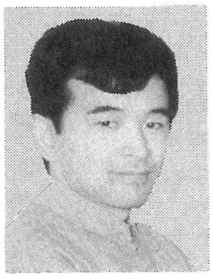
[4] R.E. Bryant and Y.-A. Chen, "Verification of arithmetic functions with binary moment diagrams," Technical Report CMU-CS-94-160, Carnegie Mellon University, 1994.

[5] R.E. Bryant and Y.-A. Chen, "Verification of arithmetic circuits with binary moment diagrams," 32nd ACM/IEEE Design Automation Conference, June 1995.



Dror Rotter was born in Haifa, Israel, on March 11, 1966. He received the B.Sc. degree in computer science from the Technion - Israel Institute of Technology, Haifa, Israel, in 1994. From 1995 to 1996 he was a research student at the department of Information Science, Faculty of Engineering, Kyoto University, Japan. Since 1996 he has been a master course student in the Graduate School of Information Science at Nara Institute of Science and Technology, Japan. His research interests include formal verification and computer aided design.

Kiyoharu Hamaguchi was born in Osaka, Japan, in 1964. He received the B.E., M.E. and Ph.D. degrees in information science form Kyoto University, Kyoto, Japan, in 1987, 1989 and 1993 respectively. In 1994, he joined the Department of Information Science, Faculty of Engineering, Kyoto University, as a Lecturer. He is currently with the Department of Informatics and Mathematical Science, Graduate School of Engineering Science, Osaka University. His current interests include formal verification and computer aided design.



Shin-ichi Minato was born in Ishikawa, Japan, on August 30, 1965. He received the B.E., M.E., and D.E. degrees in Information Science from Kyoto University in 1988, 1990, and 1995, respectively. Since 1990, he has been working for Nippon Telegraph and Telephone (NTT). His research interests include data structures and algorithms for digital system synthesis and verification. From January to December in 1997, he is a visiting scholar of Computer Science Department at Stanford University. He published "Binary Decision Diagrams and Applications for VLSI CAD" (Kluwer Academic Publishers, 1996). He is also a member of IEEE and IPSJ.

Shuzo Yajima was born in Takarazuka, Japan, on December 6, 1933. He received the B.E., M.E., and Ph.D. degrees in electrical engineering from Kyoto University, Kyoto, Japan, in 1956, 1958, and 1964, respectively. He developed Kyoto University's first digital computer, KDC-I, in 1960. In 1961 he joined the faculty of Kyoto University. From 1971 to 1997 he was a Professor in the Department of Information Science, Faculty of Engineering, Kyoto University. He is currently with the Faculty of Informatics, Kansai University. He has engaged in research and education in logic circuits, switching, and automata theory. Dr. Yajima was a Trustee of the Institute of Electronics and Communication Engineers of Japan and Chairman of the Technical Committee on Automata and Languages of the Institute. He served on the Board of Directors of the Information Processing Society of Japan.