Hybrid Decision Diagrams

Overcoming the Limitations of MTBDDs and BMDs

E. M. Clarke
School of Computer Science
Carnegie Mellon University
Pittsburgh, PA 15213
e-mail: emc@cs.cmu.edu

M. Fujita
Fujitsu Labs of America Inc.
77 Rio Robles
San Jose, CA 95134
e-mail: masahiro@eecs.berkeley.edu

X. Zhao
School of Computer Science
Carnegie Mellon University
Pittsburgh, PA 15213
e-mail: xzhao@cs.cmu.edu

Abstract: Functions that map boolean vectors into the integers are important for the design and verification of arithmetic circuits. MTBDDs and BMDs have been proposed for representing this class of functions. We discuss the relationship between these methods and describe a generalization called hybrid decision diagrams which is often much more concise. We show how to implement arithemetic operations efficiently for hybrid decision diagrams. In practice, this is one of the main limitations of BMDs since performing arithmetic operations on functions expressed in this notation can be very expensive. In order to extend symbolic model checking algorithms to handle arithmetic properties, it is essential to be able to compute the BDD for the set of variable assignments that satisfy an arithmetic relation. In our paper, we give an efficient algorithm for this purpose. Moreover, we prove that for the class of linear expressions, the time complexity of our algorithm is linear in the number of variables.

1 Introduction

Functions that map boolean vectors into the integers are important for the design and verification of arithmetic circuits. In this paper, we investigate how to represent and manipulate such functions efficiently. In a previous paper [6], we have proposed two ways (MTBDDs and BDD arrays) for representing this class of functions using Binary Decision Diagrams. Recently, Bryant and Chen [4] have proposed Binary Moment Diagrams (BMDs) for representing this class of functions. In this paper, we show that the BMD of a function is the MTBDD that results from applying the inverse Reed-Muller transformation [9] to the function. Furthermore, it can be computed using the techniques that we have developed. The transformation matrix in this case is the Kronecker product [2] of a number of identical 2×2 matrices. We show that the Kronecker products of other 2×2 matrices behave in a similar way. In fact, the transformations obtained from Kronecker products of other matrices will in many cases more concise than the BMD. We have further generalized this idea so that the transformation matrix can be the Kronecker product of different matrices. In this way, we obtain a representation, called the Hybrid Decision Diagram (HDD), that is more concise than either the MTBDD or the BMD.

A similar strategy has been used by Becker [7]. However, his technique only works for the boolean domain and is not suitable for functions mapping boolean vectors into integers. When using his technique, all of the transformation matrices, the original function and the resulting function must have boolean values. Our technique, on the other hand, works over the integers. By allowing integer values, we can handle a wider range of functions. Moreover, we can obtain larger reduction factors since we have more choices for the transformation matrices. When our technique is applied to boolean functions, it can often achieve comparable and sometimes better results than dynamic variable reordering. Thus, in some cases, it can serve as an alternative to dynamic variable reordering. We conjecture that a combination of both techniques together may result in reductions that neither technique can achieve alone.

One of the main limitations of Bryant and Chen's work is that performing arithmetic operations on functions represented by BMDs is very expensive. We show how these operations can be implemented not only for BMDs, but for hybrid decision diagrams as well. Although the worst case complexity of some of these operations is exponential, our algorithms work quite well in practice. In addition, we show how logical operations can be performed on hybrid decision diagrams that are used to represent boolean functions.

Most of the properties that we want to verify about arithmetic circuits can be expressed as arithmetic relations. In order to extend symbolic model checking algorithms [5] to handle arithmetic properties, it is essential to be able to compute the BDD for the set of variable assignments that satisfy a relation. Bryant and Chen do not provide an algorithm for this. In this paper, we give an efficient algorithm for this purpose. Moreover, we prove that for the class of linear expressions, the time complexity of our algorithm is linear in the number of variables. Our techniques for handling arithmetic operations and relations are used intensively in the verification of a SRT division algorithm similar to the one that is used in the Pentium.

2 Multi-terminal binary decision diagrams

Ordered binary decision diagrams (BDDs) are a canonical representation for boolean formulas proposed by Bryant [3]. They are often substantially more compact than traditional normal forms such as conjunctive normal form and disjunctive normal form. They can also be manipulated very efficiently. Hence, BDDs have become widely used for a variety of CAD applications, including symbolic simulation, verification of combinational logic and, more recently, verification of sequential circuits.

A BDD is similar to a binary decision tree, except that its structure is a directed acyclic graph rather than a tree, and there is a strict total order placed on the occurrence of variables as one traverses the graph from root to leaf. Algorithms of linear complexity exist for computing BDD representations of $\neg f$ and $f \vee g$ from the BDDs for the formulas f and g.

Let $f : B^m \rightarrow Z$ be a function that maps boolean vectors of length m into integers. Suppose $n_1, \ldots, n_N$ are the possible values of f. The function f partitions the space B^m of boolean vectors into N sets $\{S_1, \cdots, S_N\}$, such that $S_i = \{\, \bar{x} \mid f(\bar{x}) = n_i \,\}$. Let f_i be the characteristic function of S_i, we say that f is in *normal form* if $f(\bar{x})$ is represented as $\sum_{i=1}^{N} f_i(\bar{x}) \cdot n_i$. This sum can be represented as a BDD with integers as its terminal nodes. We call such DAGs *Multi-Terminal BDDs* (MTBDDs) [6, 1]. Any arithmetic operation $\odot$ on MTBDDs can be performed in time linear to the MTBDD size of the arguments [6].

Let M be a $2^k \times 2^l$ matrix over Z. It is easy to see that M can be represented as a function $M : B^{k+l} \rightarrow Z$, such that $M_{ij} = M(\bar{x}, \bar{y})$, where $\bar{x}$ is the bit vector for i and $\bar{y}$ is the bit vector for j. Therefore, matrices with integer values can be represented as integer valued functions using the representation shown above. We can also perform various matrix operations using the MTBDD representation. In particular, matrix multiplication can be computed in the following way: Suppose that two matrices A and B have dimensions $2^k \times 2^l$ and $2^l \times 2^m$, respectively. Let $C = A \times B$ be the product of A and B, then C will have dimension $2^k \times 2^m$. If we treat A and B as integer-valued functions, we can compute the product matrix C as

$$C(\bar{x}, \bar{z}) = \sum_{\bar{y}} A(\bar{x}, \bar{y}) B(\bar{y}, \bar{z}),$$

where $\sum_{\bar{y}}$ means "sum over all possible assignments to $\bar{y}$". In practice, $\sum_{\bar{y}} M(\bar{x}, \bar{y})$ can be computed as:

$$\begin{aligned} & \sum_{y_1 y_2 \cdots y_m} M(\bar{x}, y_1, y_2, \ldots, y_m) \\ = & \sum_{y_1 y_2 \cdots y_{m-1}} \sum_{y_m} M(\bar{x}, y_1, y_2, \ldots, y_m) \\ = & \sum_{y_1 y_2 \cdots y_{m-1}} (M(\bar{x}, y_1, y_2, \ldots, y_{m-1}, 0) \\ & \qquad + M(\bar{x}, y_1, y_2, \ldots, y_{m-1}, 1)) \end{aligned}$$

In this way, each variable in $\bar{y}$ is eliminated by performing an addition. Although this operation works well in many cases, the worst case complexity can be exponential in the number of variables.

Such integer functions can also be represented as arrays of BDDs. These BDDs have boolean values and each of them corresponds to one bit of the binary representation of the function value. In general, it is quite expensive to perform operations using this representation.

3 Hybrid decision diagrams

Recently, Bryant and Chen[4] have developed a new representation for functions that map boolean vectors to integer values. This representation is called the Binary Moment Diagram (BMD) of the function. Instead of the Shannon expansion $f = xf_1 + (1-x)f_0$, they use the expansion $f = f_0 + xf'$, where f' is equal to $f_1 - f_0$. After merging the common subexpressions, a DAG representation for the function is obtained. They prove in their paper that this gives a compact representation for certain functions which have exponential size if represented by MTBDDs directly.

There is a close relationship between this representation and the inverse Reed-Muller transformation [9]. The matrix for the inverse Reed-Muller transformation is defined recursively by

$$S_0 = 1 \qquad S_n = \begin{pmatrix} S_{n-1} & 0 \\ -S_{n-1} & S_{n-1} \end{pmatrix}$$

which has a linear MTBDD representation. Let $\bar{i} \in B^n$ be the binary representation of integer $0 \leq i < 2^n$. A function $f : B^n \rightarrow N$ can be represented as a column vector where the value of the ith entry is $f(\bar{i})$. We will not distinguish between a function and its corresponding column vector. The inverse Reed-Muller transformation can be obtained by multiplying the transformation matrix and the column vector $\widehat{f} = S \times f$ using the technique described in previous section.

Theorem 1 *The MTBDD of $\widehat{f}$ is isomorphic to the BMD of f.*

Proof: The theorem is easy to prove by induction on the number of variables.
Base Case: If the number of variables is 0, the function is a constant and $\widehat{f} = f$. Both the MTBDD of $\widehat{f}$ and the BMD for f are terminal nodes and therefore isomorphic.
Induction Step: Let $f : B^n \rightarrow N$. The roots of both the BMD for f and the MTBDD for $\widehat{f}$ are x_n. The left child of the root of the BMD for f is the BMD for f_0, while the right child is the BMD for $f_1 - f_0$. When f is represented as a column vector, the upper half is f_0 and the bottom half is f_1. The inverse Reed-Muller matrix is $\begin{pmatrix} S_{n-1} & 0 \\ -S_{n-1} & S_{n-1} \end{pmatrix}$. The result of the transformation is therefore:

$$\begin{pmatrix} S_{n-1} & 0 \\ -S_{n-1} & S_{n-1} \end{pmatrix} \times \begin{pmatrix} f_0 \\ f_1 \end{pmatrix} = \begin{pmatrix} S_{n-1} \times f_0 \\ S_{n-1} \times (f_1 - f_0) \end{pmatrix}$$

If this vector is represented by MTBDD, the left child is the MTBDD for the inverse Reed-Muller transform of f_0 and the right child is the MTBDD for the inverse Reed-Muller transform of $f_1 - f_0$. By induction hypothesis, both children are isomorphic to the children of the root of the BMD for f. Therefore the BMD of f is isomorphic to the MTBDD for $\widehat{f}$. $\square$

The Kronecker product of two matrices is defined as follows:

$$A\otimes B = \begin{pmatrix} a_{11} & \dots & a_{1m} \\ \vdots & & \vdots \\ a_{n1} & \dots & a_{nm} \end{pmatrix} \otimes B = \begin{pmatrix} a_{11}B & \dots & a_{1m}B \\ \vdots & & \vdots \\ a_{n1}B & \dots & a_{nm}B \end{pmatrix}$$

The inverse Reed-Muller matrix can be represented as the Kronecker product of n identical 2×2 matrices:

$$\begin{aligned} S_n &= \begin{pmatrix} S_{n-1} & 0 \\ -S_{n-1} & S_{n-1} \end{pmatrix} = \begin{pmatrix} 1 & 0 \\ -1 & 1 \end{pmatrix} \otimes S_{n-1} \\ &= \underbrace{\begin{pmatrix} 1 & 0 \\ -1 & 1 \end{pmatrix} \otimes \dots \otimes \begin{pmatrix} 1 & 0 \\ -1 & 1 \end{pmatrix}}_{n} \end{aligned}$$

The inverse Reed-Muller transformation is not unique in this respect. Other transformations that are defined as Kronecker products of 2×2 matrices may also provide concise representations for functions mapping boolean vectors into integers. In particular, Reed-Muller matrix R_n and Walsh matrix W_n can be represented as Kronecker products of similar form. In fact, the Kronecker product of any non-singular 2×2 matrices can be used as a transformation matrix and will produce a canonical representation for the function.

The possibility of using BMDs to represent boolean functions is discussed in [4]. In general, the BMD does not appear to be better than the ordinary BDD for representing boolean functions. In order to see why this is true, consider the boolean Reed-Muller transformation, which is sometimes called the Functional Decision Diagram or FDD[8]. This transformation can be obtained by applying the modulo 2 operations to all of the terminal nodes of the BMD. Consequently, the size of FDD is always smaller than the size of the BMD. Since the inverse boolean Reed-Muller transformation is the same as the boolean Reed-Muller transformation, the FDD of the FDD is the original BDD. Therefore, for every function f such that $|\mathrm{FDD}_f| < |\mathrm{BDD}_f|$, there exists another function f' which is the boolean Reed-Muller transform of f such that $|\mathrm{BDD}_{f'}| < |\mathrm{FDD}_{f'}|$. In particular, both the BMD and the FDD representations for the middle bit of a multiplier are still exponential.

We have discussed transformations that can be represented as the Kronecker product of a number of identical 2×2 matrices. If the transformation matrix is a Kronecker product of different 2×2 matrices, we still have a canonical representation of the function. We call transformations obtained from such matrices *hybrid transformations*.

We can apply this idea to reduce the size of BDD representation of functions. Since there is no known polynomial algorithm to find the hybrid Kronecker transformation that minimizes BDD size, we use a greedy algorithm to reduce the size. If we restrict the entries in the matrix to the set $\{0, 1, -1\}$, then there are six matrices we can try. For each variable, we select the matrix that gives the smallest BDD size. The BDDs obtained from such transformations are called Hybrid Decision Diagrams (HDDs). We have tried this method on the ISCAS85 benchmark circuits. In some cases we have been able to reduce the size of BDD representation by a factor of 1300. However, reductions of this magnitute usually occur when the original function has a bad variable ordering. If dynamic variable ordering is used, then our method gives a much smaller reduction factor.

Note that our technique can achieve comparable and sometimes better results than dynamic variable reordering. Thus, in some cases, it can serve as an alternative to dynamic variable reordering. We conjecture that the combination of both techniques together may result in reductions that neither technique can achieve alone.

4 Arithmetic operations

In order to make the techniques described in the previous sections more useful, it is desirable to be able to perform various arithmetic operations on hybrid decision diagrams. In this paper, we only consider the cases of addition and multiplication of two integers.

Suppose that f is transformed into f' by the matrix T_1 and g is transformed into g' by the matrix T_2 using the techniques discussed in the previous sections. Scalar multiplication is simple to perform.

$$(cf)' = T_1 \times (cf) = cT_1 \times f = cf'$$

When $T_1 = T_2$, finding the sum of two function is also simple.

$$(f+g)' = T_1 \times (f+g) = T_1 \times f + T_1 \times g = f' + g'$$

If $T_1 \neq T_2$, the transformation applied to the sum must be determined first. Suppose we use T_2 as the transformation matrix for the result,

$$(f+g)' = T_2 \times (f+g) = T_2 \times f + T_2 \times g = T_2 \times T_1^{-1} \times f' + g'.$$

Next, we consider how to perform multiplication. We choose T_2 as the transformation matrix for $(f \cdot g)$. Suppose the top level variable is x_i. Assume the top level transform for f is $\begin{pmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{pmatrix}$ with inverse $\begin{pmatrix} a'_{11} & a'_{12} \\ a'_{21} & a'_{22} \end{pmatrix}$. Assume also the top level transform for g is $\begin{pmatrix} b_{11} & b_{12} \\ b_{21} & b_{22} \end{pmatrix}$ with inverse $\begin{pmatrix} b'_{11} & b'_{12} \\ b'_{21} & b'_{22} \end{pmatrix}$. Then $T_2 = \begin{pmatrix} b_{11} & b_{12} \\ b_{21} & b_{22} \end{pmatrix} \otimes S_2 = \begin{pmatrix} b_{11}S_2 & b_{12}S_2 \\ b_{21}S_2 & b_{22}S_2 \end{pmatrix}$.

$$\begin{aligned} (f \cdot g)' &= T_2 \times (f \cdot g) \\ &= \begin{pmatrix} b_{11}S_2 & b_{12}S_2 \\ b_{21}S_2 & b_{22}S_2 \end{pmatrix} \times \begin{pmatrix} f_0 \cdot g_0 \\ f_1 \cdot g_1 \end{pmatrix} \\ &= \begin{pmatrix} b_{11}(f_0 \cdot g_0)' + b_{12}(f_1 \cdot g_1)' \\ b_{21}(f_0 \cdot g_0)' + b_{12}(f_1 \cdot g_1)' \end{pmatrix}. \end{aligned}$$

Consequently, the left child for the hybrid decision diagram representing $f \cdot g$

$$\begin{aligned} (f \cdot g)'_l &= b_{11}(f_0 \cdot g_0)' + b_{12}(f_1 \cdot g_1)' \\ &= b_{11}((a'_{11}f_l + a'_{12}f_r) \cdot (b'_{11}g_l + b'_{12}g_r))' \\ &\quad + b_{12}((a'_{21}f_l + a'_{22}f_r) \cdot (b'_{21}g_l + b'_{22}g_r))' \end{aligned}$$

example circuit			without reordering			with reordering		
circuit	|input|	output	|BDD|	|BMD|	|HDD|	|BDD|	| BMD |	|HDD|
c1355	41	1327	9419	1217689	2857	4407	478903	1518
c1908	33	12	3703	140174	1374	1581	154488	632
c5315	178	676	679593	2820	521	108	5106	107

Table 1: Experimental results for hybrid transformations of some ISCAS85 circuits

$$\begin{aligned} &= (b_{11}a'_{11}b'_{11} + b_{12}a'_{21}b'_{21})(f_l \cdot g_l)' \\ &\quad +(b_{11}a'_{11}b'_{12} + b_{12}a'_{21}b'_{22})(f_l \cdot g_r)' \\ &\quad +(b_{11}a'_{12}b'_{11} + b_{12}a'_{22}b'_{21})(f_r \cdot g_l)' \\ &\quad +(b_{11}a'_{12}b'_{12} + b_{12}a'_{22}b'_{22})(f_r \cdot g_r)' \end{aligned}$$

while the right child

$$\begin{aligned} (f \cdot g)'_r &= b_{21}(f_0 \cdot g_0)' + b_{22}(f_1 \cdot g_1)' \\ &= b_{21}((a'_{11}f_l + a'_{12}f_r) \cdot (b'_{11}g_l + b'_{12}g_r))' \\ &\quad +b_{22}((a'_{21}f_l + a'_{22}f_r) \cdot (b'_{21}g_l + b'_{22}g_r))' \\ &= (b_{21}a'_{11}b'_{11} + b_{22}a'_{21}b'_{21})(f_l \cdot g_l)' \\ &\quad +(b_{21}a'_{11}b'_{12} + b_{22}a'_{21}b'_{22})(f_l \cdot g_r)' \\ &\quad +(b_{21}a'_{12}b'_{11} + b_{22}a'_{22}b'_{21})(f_r \cdot g_l)' \\ &\quad +(b_{21}a'_{12}b'_{12} + b_{22}a'_{22}b'_{22})(f_r \cdot g_r)' \end{aligned}$$

Since both $(f \cdot g)'_l$ and $(f \cdot g)'_r$ can be computed in term of $(f_l \cdot g_l)', (f_l \cdot g_r)', (f_r \cdot g_l)'$, and $(f_r \cdot g_r)'$, we can compute the transformation of the product in a recursive manner. If we store these intermidiate results, the total number of recursive calls to compute $(f \cdot g)'$ will be at most $|f'||g'|$. Because of the additions that are needed in the computation, the worst case complexity can still be exponential. However, in practice, this algorithm works quite well.

Now that we are able to add and multiply functions, we can perform all of the standard logical operations. For example $(\neg f)' = (1 - f)' = 1' - f'$ and $(f \wedge g)' = (f \cdot g)'$.

5 Equations and inequalities

Frequently, it is useful to be able to compute the set of assignments that make $f_1 \sim f_2$, where $\sim$ can be one of $=, \neq, <, \leq, >,$ or $\geq$. For example, the following inequality is extremely important for the correctness of the radix-4 SRT floating point division algorithm.

$$-2 \cdot \text{divisor} \leq 3 \cdot \text{remainder} \leq 2 \cdot \text{divisor}$$

Both *divisor* and *remainder* in the inequality can be regarded as arrays of boolean variables. In order to verify the correctness of the algorithm, it is necessary to determine the set of assignments to these variables that make the inequality true.

Finding the set of assignments that satisfy an inequality can be reduced to the problem of finding the set of assignments that make a function f positive. Equations can be handled in a similar manner. A straightforward way of solving the problem is to convert f to an MTBDD and then pick the terminal nodes with the correct sign. However, this does not work very well in general, because some functions have MTBDDs with exponential size but hybrid BDDs of polynomial size. For example, let $f_1 = \sum_{i=0}^{m} x_i 2^i$ and $f_2 = \sum_{j=0}^{m} y_j 2^j$. Both of these functions and their difference have linear size BMDs. The BDD for the set of assignments satisfying $f_1 - f_2 > 0$ also has linear size. But the MTBDD size for $f_1 - f_2$ is exponential.

We have developed an algorithm that can substantially reduce the cost for computing arithmetic relations between certain functions. In the process, we only need to know the sign of the function values. Thus, if we find out that all of the values in a sub-HDD have the same sign, we can conclude that all assignments in the sub-HDD will have the same value for the relation. Consequently, we don't need to continue to expand this sub-HDD.

To obtain a good algorithm for this problem, it is necessary to determine efficiently if a sub-HDD has uniform sign. This can be achieved by computing upper and lower bounds for the sub-HDD. The algorithm given below determines this information. If the intermediate results are stored, the algorithm takes time linear in the number of BDD nodes.

```
bound_values(f, upper, lower)
begin
  if(f is terminal node)
    upper = lower = f.value;

  bound_values(left(f), upper1, lower1);
  bound_values(right(f), upper2, lower2);

  let {{a11, a12}, {a21, a22}}
      be the inverse matrix at node f;

  upper11 = a11*(if a11>0 then upper1 else lower1);
  upper12 = a12*(if a12>0 then upper2 else lower2);
  upper21 = a21*(if a21>0 then upper1 else lower1);
  upper22 = a22*(if a22>0 then upper2 else lower2);

  lower11 = a11*(if a11>0 then lower1 else upper1);
  lower12 = a12*(if a12>0 then lower2 else upper2);
  lower21 = a21*(if a21>0 then lower1 else upper1);
  lower22 = a22*(if a22>0 then lower2 else upper2);

  upper = max(upper11+upper12,upper21+upper22);
  lower = min(lower11+lower12,lower21+lower22);
end
```

The improved algorithm for computing the BDD for the set of assignments that make the function f positive is given below. A similar algorithm is used to find the set of assignments that make a function zero.

```
bdd greater_than_0(f)
begin
  if(f is terminal node)
    if(f.value > 0) return(True);
    else            return(False);

  bound_values(f, upper, lower);
  if(upper <= 0) return(False);
  if(lower > 0)  return(True);

  let {{a11, a12}, {a21, a22}}
      be the inverse matrix at node f;
  left=greater_than_0(a11*left(f)+a12*right(f));
  right=greater_than_0(a21*left(f)+a22*right(f));
  return(bdd_if_then_else(level(f), left, right));
end
```

This algorithm works extremely well for verification of arithmetic circuits. The following theorem guarantees the efficiency of this algorithm for the set of *linear expressions* when the Hybrid Decision Diagrams are BMDs. Most of the formulas that occur duing the verification of the SRT division algorithm are in this class. These expressions have the form $f = \sum_{i=1}^{m} c_i f_i$, where $f_i = \sum_{j=0}^{n} x_{ij} 2^j$ for $1 < i < m$ and the c_i's are integer constants. We use the variable ordering $x_{1n}, x_{2n}, \ldots, x_{mn}, \ldots, x_{10}, x_{20}, \ldots, x_{m0}$.

Theorem 2 *The complexity of* greater_than_0 *for* f *is* $O(n^2 \sum_{k=1}^{m} |c_k|)$.

In the case of linear inequalities, all the new BMDs that are generated have the form of $c+g$, where c is a constant and g is an existing BMD. If we remember the constant without actually adding it to the BMDs, we are able to avoid generating new BMD nodes. After introducing this technique, the complexity for compute greater_than_0(f) can be further reduced to $O(n \sum_{k=1}^{m} |c_k|)$.

6 Directions for future research

In this paper, we have discussed the relationship between MTBDDs and BMDs. We have also described a generalization called hybrid decision diagrams which is often much more concise. An efficient implementation of arithemetic operations on hybrid decision diagrams is also given.

Computing the BDD for the set of variable assignments that satisfy an arithmetic relation is important for reasoning about arithmetic circuits. We give an efficient algorithm for this purpose. Moreover, we prove that for the class of linear expressions, the time complexity of our algorithm is linear in the number of variables.

There are a number of directions for future research. Currently, we use a greedy algorithm to choose the appropriate transformation matrix at each level in a hybrid decision diagram. Although it seems unlikely that there is an efficient algorithm to find the optimal transformation, it may be possible to develop a better heuristic that would permit an even more concise representation.

In hybrid decision diagrams, the transformation matrices for all the nodes at one level must be the same. If we allow these transformation matrices to differ, we should have more freedom in selecting the transformation and, therefore, be able to reduce the representation further.

Finally, our algorithm for solving arithmetic relations works extremely well for linear equations and inequalities. Although the current algorithm can handle some nonlinear equations and inequalities as well, it may be possible to extend this algorithm or to find a new algorithm that can handle more complicated nonlinear equations and inequalities.

References

[1] R. I. Bahar, E. A. Frohm, C. M. Gaona, G. D. Hachtel, E. Macii, A. Pardo, and F. Somenzi. Algebraic decision diagrams and their applications. In *Proceedings of the 1993 IEEE International Conference on Computer Aided Design*. IEEE Computer Society Press, November 1993.

[2] R. Bellman. *Introcution to matrix analysis*, chapter 5. McGraw-Hill, 1970.

[3] R. E. Bryant. Graph-based algorithms for boolean function manipulation. *IEEE Transactions on Computers*, C-35(8), 1986.

[4] R. E. Bryant and Y. A. Chen. Verification of arithmetic functions with binary moment diagrams. In *Proceedings of the 32nd ACM/IEEE Design Automation Conference*. IEEE Computer Society Press, June 1995.

[5] J. R. Burch, E. M. Clarke, K. L. McMillan, D. L. Dill, and L. J. Hwang. Symbolic model checking: 10^{20} states and beyond. *Information and Computation*, 98(2):142–170, June 1992.

[6] E. M. Clarke, K. McMillan, X. Zhao, M. Fujita, and J. Yang. Spectral transforms for large boolean functions with applications to technology mapping. In *Proceedings of the 30th ACM/IEEE Design Automation Conference*. IEEE Computer Society Press, June 1993.

[7] R. Drechsler, A. Sarabi, M. Theobald, B. Becker, and M. A. Perkowski. Efficient representation and manipulation of switching functions based on ordered kroenecker functional decision diagrams. In *Proceedings of the 32nd ACM/IEEE Design Automation Conference*. IEEE Computer Society Press, June 1994.

[8] R. Drechsler, M. Theobald, and B. Becker. Fast ofdd based minimization of fixed polarity reed-muller expressions. In *Proceedings of the 1994 European Design Automation Conference*. IEEE Computer Society Press, June 1994.

[9] D. E. Muller. Application of boolean algebra to switching circuit design and error detection. *IRE Trans.*, 1:6–12, 1954.