International Conference on Computer Aided Design (ICCAD'99)
pp. 50-53, San Jose, November 1999

Efficient Manipulation Algorithms for Linearly Transformed BDDs

Wolfgang Günther Rolf Drechsler

Institute of Computer Science, Albert-Ludwigs-University
79110 Freiburg im Breisgau, Germany
{guenther,drechsle}@informatik.uni-freiburg.de

Abstract

Binary Decision Diagrams (BDDs) are the state-of-the-art data structure in VLSI CAD. But due to their ordering restriction only exponential sized BDDs exist for many functions of practical relevance. Linear Transformations (LTs) have been proposed as a new concept to minimize the size of BDDs and it is known that in some cases even an exponential reduction can be obtained.

In addition to a small representation, the efficient manipulation of a data structure is also important. In this paper we present polynomial time manipulation algorithms that can be used for Linearly Transformed BDDs (LT-BDDs) analogously to BDDs. For some operations, like synthesis algorithms based on ITE, it turns out that the techniques known from BDDs can be directly transferred, while for other operations, like quantification and cofactor computation, completely different algorithms have to be used. Experimental results are given to show the efficiency of the approach.

1 Introduction

Binary Decision Diagrams (BDDs) [2] are the state-of-the-art data structure in VLSI CAD. They allow a compact representation for many functions and efficient algorithms are known to manipulate them. However, the size of BDDs largely depends on the underlying variable order and for some functions even no efficient representation exists, like for the Boolean multiplication [3]. To overcome these problems, many generalizations of the basic BDD concept have been proposed which allow a more compact representation for Boolean functions (see e.g. [5, 7, 13]).

A very promising one is based on spectral techniques, but instead of transforming the whole function table of size 2^n, only the input vector is linearly transformed. There are functions for which *Linearly Transformed BDDs* (LT-BDDs) are exponentially smaller than BDDs [8]. To store a linear transformation, a non-singular Boolean matrix of size $n \times n$ is sufficient. In the meantime also efficient heuristics to find a good linear transformation have been presented [10, 8, 9]. In [10], the widely used sifting algorithm [12] for dynamic BDD minimization has been extended by a linear operator which combines neighboring variables of the BDD, while in [8] a window optimization algorithm is proposed. All algorithms run very fast, since the basic operation is a local operation (very similar to variable reordering [6]). By this, it is possible to obtain much smaller BDD sizes than with sifting alone. All these approaches have mainly focused on reducing the representation size of Boolean functions.

However, for data structures in general not only compact representation is of importance but also efficient algorithms to manipulate them. Therefore, to use LT-BDDs, e.g. in sequential verification, basic operations like existential quantification must be realized efficiently, which cannot be done in the same way as for BDDs.

In this work, we present manipulation algorithms for LT-BDDs which can be carried out in polynomial time. It is shown that the classical synthesis operations based on ITE [1] can directly be applied to LT-BDDs. However, since in LT-BDDs the same variable can occur several times, i.e. LT-BDDs do not have a read-once property, the algorithms for cofactor computation and existential quantification are completely different. Experimental results show that for functions where large reductions in size can be obtained using LT-BDDs, also a large reduction in runtime for existential quantification is achieved. On the other hand, for functions where LT-BDDs result in no gain, the runtime for existential quantification is in the same range as for BDDs.

The paper is structured as follows: in Section 2, we briefly review some definitions used in this paper. In Section 3, manipulation algorithms for LT-BDDs are presented. Experimental results are given in Section 4. Finally, the paper is summarized.

2 Preliminaries

2.1 Binary Decision Diagrams

As is well-known, each Boolean function $f : \mathbf{B}^n \to \mathbf{B}$ can be represented by a *Binary Decision Diagram* (BDD) [2], i.e. a directed acyclic graph where a Shannon decomposition

$$f = \bar{x}_i f_{x_i=0} + x_i f_{x_i=1} \quad (1 \le i \le n)$$

is carried out in each node.

A BDD is called *ordered* if each variable is encountered at most once on each path from the root to a terminal node and if the variables are encountered in the same order on all such paths. A BDD is called *reduced* if it does neither contain

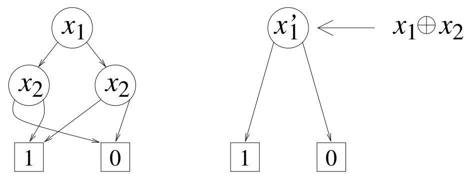

Figure 1: BDD and LT-BDD for function $f(x_1,x_2)=\overline{x}_1x_2+x_1\overline{x}_2$.

isomorphic sub-graphs nor vertices with both edges pointing to the same node. Reduced and ordered BDDs are canonical, i.e. for each Boolean function the BDD can be uniquely determined. Furthermore, for functions represented by reduced ordered BDDs efficient manipulations are possible [2, 1, 4]. In the following, only reduced, ordered BDDs are considered and for briefness these graphs are called BDDs.

2.2 Linear Transformations

Definition 1 A *Linear Transformation* (LT) of a set of input variables $X_n=\{x_1,\ldots,x_n\}$ is a bijective mapping $\tau:\mathbf{B}^n\to\mathbf{B}^n$ that maps each variable x_i to a linear combination of a set of variables V_i, i.e.

$$x_i' = \bigoplus_{x_j\in V_i} x_j.$$

This can also be seen as a re-encoding of the variables which can be represented by a non-singular $n\times n$-matrix over the Galois field $(\mathbf{B},\oplus,\cdot)$.

In other words, instead of labeling nodes with single variables, they are labeled with the parity of a set of variables.

The corresponding diagrams representing a linearly transformed BDD are called LT-BDDs in the following. LT-BDDs are a canonical representation for a fixed variable ordering and linear transformation. Therefore, also the satisfiability problem can easily be checked.

Example 1 For function $f(x_1,x_2)=\overline{x}_1x_2+x_1\overline{x}_2$, the mapping $\tau(x_1,x_2)=(x_1\oplus x_2,x_2)$ is a linear transformation. The BDD and the LT-BDD for f are given in Figure 1.

As shown in [11], the number of possible LTs is

$$\prod_{i=0}^{n-1}(2^n-2^i),$$

which is much larger than $n!$, the number of possible variable orderings, but much smaller than $2^n!$, the number of all possible automorphisms. Similar estimations hold for the space requirements: permutations require $O(n)$ space, linear transformations $O(n^2)$, and general automorphisms $O(2^n)$. Therefore, linear transformations are a good compromise, as they can be represented efficiently while enlarging the search space significantly, which gives more room for optimizations. In [8] it has been proven that for some functions even an exponential reduction in size can be observed.

In the following we denote the composition of two functions with $\circ$, i.e. $(f\circ g)(x)=f(g(x))$ for any two functions f and g.

3 Operations on LT-BDDs

The quality of a data structure largely depends on the efficiency of manipulation algorithms. In this section it is shown that for LT-BDDs common BDD operations can be carried out in polynomial time.

3.1 ITE

For BDDs it is well-known that all synthesis operations, like AND and OR, can be carried out efficiently by making use of *if-then-else* (*ite*) [1]. The following computation shows that these results can be transferred to LT-BDDs: if linear transformed functions $f\circ\tau$, $g\circ\tau$, and $h\circ\tau$ are used with the *ite*-operator $ite(f,g,h)=f\cdot g+\overline{f}\cdot h$ (the transformation has to be the same for all operators f, g, and h), then

$$\begin{aligned} & ite(f\circ\tau,g\circ\tau,h\circ\tau)\\ &\quad= (f\circ\tau)\cdot(g\circ\tau)+(\overline{f}\circ\tau)\cdot(h\circ\tau)\\ &\quad= ((f\cdot g)\circ\tau)+((\overline{f}\cdot h)\circ\tau)\\ &\quad= (f\cdot g+\overline{f}\cdot h)\circ\tau\\ &\quad= ite(f,g,h)\circ\tau. \end{aligned}$$

In other words, the *ite*-operator can be carried out on the BDD structure and the linear transformation can be ignored. The results of polynomial worst case complexity from [1] can directly be transferred.

3.2 Cofactor

The cofactor $f_{x_i=b}$ for a given Boolean function $f:\mathbf{B}^n\to\mathbf{B}$ and $b\in\mathbf{B}$ is defined as

$$f_{x_i=b}(x_1,\ldots,x_n)\equiv f(x_1,\ldots,x_{i-1},b,x_{i+1},\ldots,x_n),$$

i.e. variable x_i is substituted by a constant b. To simplify the following calculation, we restrict w.l.o.g. to case $b=0$. The results for $b=1$ follow analogously.

For BDDs, the cofactor of a function can be obtained by redirecting edges pointing to nodes labeled with x_i to their *low* successor nodes. In case of LT-BDDs, this algorithm cannot be used, since a variable can occur more than once in the transformation (see Example 1). However, it holds

$$\begin{aligned} & f_{x_i=0}(x_1,\ldots,x_n)\\ &\quad= f(x_1,\ldots,x_{i-1},0,x_{i+1},\ldots,x_n)\\ &\quad= x_i\cdot f(x_1,\ldots,x_{i-1},\overline{x}_i,x_{i+1},\ldots,x_n)\\ &\quad+ \overline{x}_i\cdot f(x_1,\ldots,x_{i-1},x_i,x_{i+1},\ldots,x_n). \end{aligned}$$

In the following, we define

$$f^{\overline{x}_i}(x_1,\ldots,x_n)\equiv f(x_1,\ldots,x_{i-1},\overline{x}_i,x_{i+1},\ldots,x_n),$$

i.e. the function obtained by complementing variable x_i. Using this notation, the previous equation can be written as

$$f_{x_i=0}=x_i\cdot f^{\overline{x}_i}+\overline{x}_i\cdot f=ite(x_i,f^{\overline{x}_i},f).$$

Note that x_i refers to the untransformed variable. To obtain a representation for x_i in the LT-BDD, it is possible to keep a

```
complement_variable(f, x_i) {
    if (f is constant)
        return f;
    if (computed table has entry (f, x_i))
        return entry;
    v = top variable of f;
    t = complement_variable(f_high, x_i);
    e = complement_variable(f_low, x_i);
    if (linear transformation for v contains x_i)
            result = node(v, e, t);
    else    result = node(v, t, e);
    insert (f, x_i, result) to computed table;
    return result;
}
```

Figure 2: Sketch of algorithm to complement a variable

BDD for each variable representing it and to perform transformations also on these "projection functions"[1]. Furthermore, $f^{\overline{x}_i}$ can be computed efficiently from f by exchanging *low*- and *high*-edges for all nodes which depend on x_i, since complementing a variable $x_i \in V_j$ in an expression like $x'_j = \bigoplus_{x_k \in V_j} x_k$ results in complementing variable x'_j. A sketch of this algorithm is given in Figure 2.

Theorem 1 Cofactor computation on LT-BDDs can be done in polynomial time.

Proof: This follows from the description of the algorithm above. LT-BDDs for the untransformed variables can have at most n nodes and variables can be complemented in linear time. Furthermore, the worst case for *ite* is the product of the BDD sizes of the arguments, i.e. in this case time and space is bounded by $O(n \cdot |f|^2)$. □

To speed up the implementation, the recursive calls of *ite* and variable complementation can be merged, i.e. in the *ite*-algorithm the second operand is complemented implicitly during recursive calls. Furthermore, if variable x_i does not occur below the top variable in a call of the modified *ite*-algorithm (which can be checked by a preprocessing), the original *ite*-algorithm can be used. A sketch of the modified algorithm is given in Figure 3.

```
ite_and_comple(f, g, h, x_i) {
    if (terminal case)
        return result;
    v = top variable of f, g and h;
    if (v is below x_i)
        return ite(f, g, h);
    if (computed table has entry (f, g, h, x_i))
        return entry;
    h_1 = h_high;
    h_0 = h_low;
    if (linear transformation for v contains x_i)
            exchange h_1 and h_0;
    t = ite_and_comple(f_high, g_high, h_1, x_i);
    e = ite_and_comple(f_low, g_low, h_0, x_i);
    result = node(v, t, e);
    insert (f, g, h, x_i, result) to computed table;
    return result;
}
```

Figure 3: Sketch of the modified *ite* algorithm

3.3 Quantification

Finally, we focus on quantification of variables, an operation frequently used in sequential verification.

Existential quantification of a variable is defined by $\exists x_i : f \equiv f_{x_i=0} + f_{x_i=1}$. A straightforward algorithm can be obtained using the algorithm for cofactor computation described in the previous section. However, the computation can be speeded up using the following equations:

$$\begin{aligned}\exists x_i : f(x_1, \ldots, x_n) &= f_{x_i=0} + f_{x_i=1} \\ &= x_i f^{\overline{x}_i} + \overline{x}_i f + x_i f + \overline{x}_i f^{\overline{x}_i} \\ &= f^{\overline{x}_i} + f,\end{aligned}$$

i.e. one call of the modified *ite*-algorithm of the previous section is sufficient. The algorithm is even faster, since only two arguments of *ite* are really used for the computation of OR. Therefore, the worst case time is $O(|f|^2)$. Also the cache can be used more efficiently, since only three arguments are necessary to identify a cache entry instead of four in case of cofactor computation.

A similar computation shows that universal quantification can be computed in the same way since $\forall x_i : f(x_1, \ldots, x_n) = f^{\overline{x}_i} \cdot f$.

4 Experimental Results

In this section we describe experimental results that have been carried out on a *SUN Ultra 1* with 128 MBytes. All runtimes are given in CPU seconds. The algorithms have been integrated in the CUDD package [14].

We compare BDDs and LT-BDDs for the application of existential quantification using the CUDD package. We measure the runtime to compute for all input variables x_i and for all outputs f_j the existential quantification $\exists x_i : f_j$, i.e. for a function with n inputs and m outputs, $n \cdot m$ computations are carried out. Results are given in Table 1. In Column "circuit", the name of the circuit is given. In the next column, the number of inputs and outputs is shown. For each circuit, the BDD is built using sifting dynamically with a "maxgrowth parameter" of 2.0. A final call of converging sifting or converging linear sifting, respectively, is carried out to obtain a good initial representation. The initial number of nodes and the runtime for BDDs and LT-BDDs are given in the next columns. For LT-BDDs also the number of entries in the transformation matrix is given: a value of zero means that no transformations have been carried out.

It can be seen that for functions for which large reductions can be obtained using LT-BDDs, also the runtime for

[1] This is done in the CUDD implementation of linear sifting [10].

		BDD		LT-BDD			difference	
circuit	in/ out	nodes	time	nodes	LT	time	nodes %	time %
c432	36/ 7	1244	0.161	1244	0	0.151	0.0	-6.3
c499	41/ 32	26440	8.374	3291	179	1.598	-87.6	-80.9
c880	60/ 26	8459	2.043	8471	3	2.030	0.1	-0.6
c1355	41/ 32	29592	9.588	13239	72	7.038	-55.3	-26.6
c2670	233/140	4145	3.760	2455	190	1.909	-40.8	-49.2
c3540	50/ 22	23846	31.190	23894	17	31.630	0.2	1.4
c5315	178/123	1898	1.171	1664	59	1.334	-12.3	13.9
c7552	207/108	8557	5.739	5534	259	4.623	-35.3	-19.4
dalu	75/ 16	905	0.172	855	3	0.164	-5.5	-5.0
des	256/245	3077	2.626	2120	53	2.403	-31.1	-8.5
frg2	143/139	1208	0.383	1098	5	0.489	-9.1	27.7
i10	257/224	52825	38.480	41015	69	33.410	-22.4	-13.2
k2	45/ 45	1305	0.182	1282	6	0.179	-1.8	-1.6
sum		163501	103.870	106162		86.958	-35.1	-16.3

Table 1: Results for the existential quantification

existential quantification can be reduced tremendously (see e.g. *c499*). However, there are also cases where the transformation matrix has many entries but it cannot reduce the size much (see e.g. *c5315*). In these cases, the more expensive algorithm has to be applied while the BDDs have nearly equal size. For functions where sizes cannot be reduced, runtime is in the same range as for BDDs (see e.g. *c432* or *c880*), as in that case, the algorithm is very similar to the quantification algorithm used in CUDD[2]. All in all, the average reduction in size and runtime is 35% and 16%, respectively.

5 Conclusions

Linear transformations are a powerful concept to minimize BDD sizes. In this paper polynomial time algorithms have been presented to manipulate LT-BDDs. It has been shown that LT-BDDs allow in many cases to significantly reduce the size of the representation compared to BDDs. But in contrast to other extensions of BDDs, like FDDs and IBDDs, efficient manipulation is still possible.

Experimental results for existential quantification have shown that a reduction in space and time can be observed. It is focus of current work to apply these techniques to sequential verification. The problem that arises here is the following: to compute the relational product efficiently, existential quantification over a *set* of variables has to be carried out *simultaneously to* computing the product. While simultaneous quantification and product computation can be solved using the same approach as proposed in this paper, quantification over a set of variables cannot be handled directly using these methods.

References

[1] K.S. Brace, R.L. Rudell, and R.E. Bryant. Efficient implementation of a BDD package. In *Design Automation Conf.*, pages 40–45, 1990.

[2] R.E. Bryant. Graph - based algorithms for Boolean function manipulation. *IEEE Trans. on Comp.*, 35(8):677–691, 1986.

[3] R.E. Bryant. On the complexity of VLSI implementations and graph representations of Boolean functions with application to integer multiplication. *IEEE Trans. on Comp.*, 40:205–213, 1991.

[4] R. Drechsler and B. Becker. *Binary Decision Diagrams - Theory and Implementation*. Kluwer Academic Publishers, 1998.

[5] R. Drechsler, A. Sarabi, M. Theobald, B. Becker, and M.A. Perkowski. Efficient representation and manipulation of switching functions based on ordered Kronecker functional decision diagrams. In *Design Automation Conf.*, pages 415–419, 1994.

[6] M. Fujita, Y. Matsunaga, and T. Kakuda. On variable ordering of binary decision diagrams for the application of multi-level synthesis. In *European Conf. on Design Automation*, pages 50–54, 1991.

[7] J. Gergov and C. Meinel. Efficient analysis and manipulation of OBDDs can be extended to FBDDs. *IEEE Trans. on Comp.*, 43:1197–1209, 1994.

[8] W. Günther and R. Drechsler. BDD minimization by linear transformations. In *Advanced Computer Systems*, pages 525–532, 1998.

[9] W. Günther and R. Drechsler. Minimization of BDDs using linear transformations based on evolutionary techniques. In *Int'l Symp. Circ. and Systems*, pages I:387–I:390, 1999.

[10] C. Meinel, F. Somenzi, and T. Theobald. Linear sifting of decision diagrams. In *Design Automation Conf.*, pages 202–207, 1997.

[11] C. Meinel and T. Theobald. Local encoding transformations for optimizing OBDD-representations of finite state machines. In *FMCAD*, volume 1166, pages 404–418, 1996.

[12] R. Rudell. Dynamic variable ordering for ordered binary decision diagrams. In *Int'l Conf. on CAD*, pages 42–47, 1993.

[13] D. Sieling and I. Wegener. Graph driven BDDs - a new data structure for Boolean functions. *Theoretical Computer Science*, 141:283–310, 1995.

[14] F. Somenzi. *CUDD: CU Decision Diagram Package Release 2.3.0.* University of Colorado at Boulder, 1998.

[2] The smaller runtime for *c432* is due to a slightly more efficient implementation than in CUDD.