Application of Linearly Transformed BDDs in Sequential Verification

Wolfgang Günther  Andreas Hett  Bernd Becker

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

Abstract—The computation of the set of reachable states is the key problem of many applications in sequential verification. Binary Decision Diagrams (BDDs) are extensively used in this domain, but tend to blow up for larger instances. To increase the computational power of BDDs, linearly transformed BDDs (LTBDDs) have been proposed. In this paper we show how this concept can be incorporated into the sequential verification domain by restricting dynamic reordering in a way that the relational product can still be carried out efficiently. Experimental results are given to show the efficiency of our approach.

I. INTRODUCTION

Formal verification of sequential circuits is becoming more and more important. To compute the set of reachable states in a given Finite State Machine (FSM) is the underlying problem in most cases. To handle designs with a huge number of states symbolic techniques based on Binary Decision Diagrams (BDDs) [1, 8] are the state-of-the-art approach, avoiding an explicit representation of state sets [15, 5, 18].

Basically, there are two methods to compute the set of reachable states: one uses Transition Functions [7], the other uses Transition Relations [2]. Recently, a method to combine both approaches has been presented [18]. In this paper, we restrict to the latter approach.

There are two common ways to improve reachability analysis: the first one is based on partitioned transition relations [23, 3, 6, 4, 19], the second one uses partial and focused traversals [21, 5, 13]. Most of these approaches benefit from compact state representations.

One method to increase the computational power of BDDs (i.e. to allow that more Boolean functions can be compactly represented) is to apply Linear Transformations (LTs) on BDDs [17, 16]. The resulting Linearly Transformed BDDs (LTBDDs) often allow smaller representations of Boolean functions than standard BDDs, in some cases even an exponential reduction is possible [10]. Furthermore, many synthesis operations, e.g. the Boolean AND operation as well as existential quantifications, can be carried out in polynomial time [9].

In this paper we investigate the use of LTBDDs to compute the set of reachable states. In [9] an algorithm to compute existential quantification with respect to one variable is presented. However, to compute the relational product, which is the core operation of reachability analysis, it is necessary to carry out an AND-operation and existential quantification of a set of variables simultaneously.

\[ \exists x_1 \exists x_2 \cdots \exists x_n (f \wedge g) \]

Obviously it is possible to compute the AND-operation first and afterwards iteratively quantify each of the variables, but this takes much more time and the intermediate LTBDDs tend to blow up in size. Therefore, we restrict the underlying dynamic minimization algorithm, linear sifting [16], in a way that allows to carry out the relational product efficiently, while keeping the advantages of linear transformations.

The paper is structured as follows: first, basic definitions and notations are introduced. In Section III, the restrictions necessary for the linear transformation are described and the resulting modifications to linear sifting are presented. Finally, experimental results and a summary are given.

II. PRELIMINARIES

A. The Transition Relation Method

Definition 1 Let \( x = \{ x_1, \ldots, x_n \} \) denote a set of present state variables, \( y = \{ y_1, \ldots, y_n \} \) a set of next state variables, and \( w = \{ w_1, \ldots, w_m \} \) a set of input variables. Let \( f : \mathcal{B}^{n+m} \rightarrow \mathcal{B}^n \) be the Boolean function of
Reachability Analysis \( (S_0(x), \text{TR}_f(x,y)) \) \{ 
  \text{Reach}(x) = \text{From}(x) = S_0(x) 
  \text{do} \{ 
    \text{To}(y) = \exists x. (\text{TR}_f(x,y) \cdot \text{From}(x)) 
    \text{To}(x) = \text{To}(y)|_{y-x} 
    \text{New}(x) = \text{To}(x) \setminus \text{Reached}(x) 
    \text{if} (\text{New}(x) \neq \emptyset) \{ 
      \text{Reached}(x) = \text{Reached}(x) \cup \text{To}(x) 
      \text{From}(x) = \text{BestBdd}(\text{Reached}(x), \text{New}(x)) 
    \} 
  \} \text{while} (\text{New}(x) \neq \emptyset) 
\}

Fig. 1. Basic state traversal algorithm

the sequential circuit\(^1\). The Transition Relation (TR) of 
\( f \) is defined as \( \text{TR}_f : B^n \rightarrow B \), 
\[
\text{TR}_f(x,y) = 1 \iff \exists w. f(x,w) = y, 
\]
i.e. \( \text{TR}_f(x,y) = 1 \) iff state \( y \) is reachable from state \( x \) via 
one transition.

Computing the set of reachable states is based on a fix-
point iteration of image computations that terminates as 
soon as no new states can be reached, i.e. \( \text{New}(x) = \emptyset \) 
(see Figure 1 for a sketch). To obtain the image of a 
set \( S \subseteq B^n \) of states, the relational product has to be computed, 
\[
f(S)(y) = \exists x. (\text{TR}_f(x,y) \cdot S(x)), 
\]
which can be carried out in one pass on BDDs [2], allowing 
significant reductions in runtime and space requirements.

Replacing the variables \( y \) by \( x \) to compute \( \text{To}(y)|_{y-x} \) 
can be done by calling a substitution algorithm for all 
variables.

B. Linearly Transformed BDDs

Definition 2 A Linear Transformation of a set of (input) 
variables \( z = \{z_1, \ldots, z_n\} \) is a bijective mapping \( \tau : B^n \rightarrow B^n \) that maps each variable \( z_i \) to a linear combination of 
a set of variables \( V_i \), i.e.
\[
z'_i = \bigoplus_{z_j \in V_i} z_j. 
\]
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 \( (B, \oplus, \cdot) \).

\(^1\)Outputs of the circuit are not considered here; if necessary, they 
can be modeled as additional registers of which the circuit does not 
depend.

\[ \text{Fig. 2. BDD and LTBBDD of the function } f = (z_1 \oplus z_2) + (z_1 \oplus z_3) \]

Instead of representing a function \( f \) directly by a BDD, 
a transformed function \( f' \) is represented such that \( f(z) = f'(\tau(z)) \). The corresponding decision diagram is called 
Linearly Transformed BDD (LTBDD) in the following. In 
other words, instead of labeling nodes with one single 
variable, they are labeled with the parity of a set of variables. 
Note that the read-once property does not hold in 
LTBDDs, i.e. one (untransformed) variable may appear 
more than once on a path from the root to a terminal node.

Example 1 Consider the function \( f = (z_1 \oplus z_2) + (z_1 \oplus z_3) \). The transformation \( \tau(z_1, z_2, z_3) = (z_1 \oplus z_2, z_1 \oplus z_3, z_1) \) 
is an automorphism. The BDD for \( f(z) \) and the LTBDD 
for \( f'(z) \) are shown in Figure 2, respectively.

LTBDDs are a canonical representation for a fixed var-
iable ordering and linear transformation. Therefore, also 
the satisfiability problem can be checked easily: a function 
is satisfiable if its LTBDD is non-zero. Also Boolean 
operations (e.g. the Boolean AND-operation) can be car-
bied out efficiently [9].

C. Linear Sifting

Linear sifting [16] can be seen as an extension of the 
original sifting algorithm [22] by a linear operator\(^2\): It 
successively considers all variables of a given LTBDD. 
When a variable is chosen, the goal is to find the best 
position of the variable, assuming that the relative order 
of all other variables remains the same. At the same time, 
the transformation is improved.

In a first step, the order in which the variables are con-
sidered is determined. This is done by prioritizing all 
levels according to their size (the largest level goes first). 
To find the best position, variable \( z_i \) is moved across all 
LTBDD levels. This is done in three steps (see Figure 3):

\(^2\)Note that the linear transformation of two neighbor-
ed variables is a local operation that can be carried out efficiently.
Explore upwards: Variable $z_i$ is exchanged with its predecessor $z_j$ until it reaches the top border. After each exchange, the variables are linearly transformed, i.e. $z_i$ is replaced by $z_i \oplus z_j$. If, as a result of this transformation, the size does not decrease, the transformation is immediately undone.

Explore downwards: Variable $z_i$ is exchanged with its successor variable $z_j$ until it reaches the bottom border. After each exchange, the variables are also linearly transformed, i.e. $z_j$ is replaced by $z_j \oplus z_i$. Again, if the size does not decrease, the transformation is immediately undone.

Adjust best: Variable $z_i$ is moved to the best encountered position with respect to a minimal size of the LTBDD.

III. Restricting Linear Sifting during Reachability Analysis

Obviously using LTBDDs instead of BDDs can lead to smaller representations of the transition relation, by simply applying linear sifting instead of “standard” sifting. However, less efficient implementations for some operations of the key steps in Reachability Analysis (see Figure 1) have to be used, if the linear transformation is used without applying some restrictions. The most important operations that have to be carried out on LTBDDs in the algorithm of Figure 1 are

1. Image computation $\exists x.(\mathbf{TR}_f(x,y) \cdot \mathbf{From}(x))$
2. Variable substitution $\mathbf{To}(x) = \mathbf{To}(y)|_{y_i=x}$.

All remaining operations on sets can be carried out on LTBDDs in the same way as on BDDs [9].

A. Restrictions for the Relational Product

To efficiently compute the relational product for image computation, it is necessary to carry out an AND-operation and existential quantification of a set of variables simultaneously. Since the algorithm for existential quantification of [9] only deals with single variables, it is again necessary to restrict the type of transformation such that a fast algorithm can be applied.

In the following we show that under a certain condition, existential quantification can be carried out on LTBDDs in the same way as on BDDs, which allows to use the standard algorithm for the relational product.

Theorem 1 Let $f: \mathbb{B}^{2^n} \to \mathbb{B}$ be a Boolean function and $\tau: \mathbb{B}^n \to \mathbb{B}^n$ be a bijective mapping. Then for each $y \in \mathbb{B}^n$

$$\exists x \in \mathbb{B}^n. f(\tau(x), y) = \exists x \in \mathbb{B}^n. f(x, y).$$

This directly follows from the fact that $\tau$ is a bijection.

In our application this means that if the linear transformation is restricted such that current state variables are mapped to current state variables (i.e. $x'_i = \bigoplus x_j$ with $x_j \in V_i$, $V_i \subset x$ (i.e. $V_i$ only consists of current state variables), the linear transformation can be ignored when computing the relational product.

B. Restrictions for Variable Substitution

Generally, pairwise grouping of present state variables $x_i$ with their corresponding next state variables $y_i$ is a good heuristic for dynamic variable reordering of BDDs in sequential verification [24]. For dynamic minimization, group sifting [20] is usually used instead of standard sifting.

Grouping also allows to carry out variable substitution easily: to construct the BDD for $\mathbf{To}(x)$, the BDD for $\mathbf{To}(y)$ only has to be “moved” by one level.

If LTBDDs are used, a restriction to the linear transformation is necessary to use the same fast algorithm for variable substitution: corresponding current and next state variables need to have the same transformation, i.e. whenever two current state variables $x_i$ and $x_j$ are combined, the corresponding next state variables $y_i$ and $y_j$ also have to be combined.

\[3\text{In [14] it has been shown that this is not true in all cases. Instead, this grouping is relaxed to further reduce the BDD sizes. For sake of simplicity, we do not make use of this concept here. However, it could be integrated similarly.}\]
Linear Group Sifting Up \((x_i, y_i)\) : group to sift} {
while \((x_i, y_i)\) is not the topmost group) \{ 
    exchange group with its predecessor \((x_j, y_j)\)
    \(size_{\text{notrans}} = \text{BDD\_size()}\)
    linearly combine \((x_i, y_i)\) with \((x_j, y_j)\)
    \(size_{\text{trans}} = \text{BDD\_size()}\)
    if \((size_{\text{notrans}} < size_{\text{trans}})\)
        linearly combine \((x_i, y_i)\) with \((x_j, y_j)\) again (i.e. undo linear transformation)
\}

Fig. 4. Linear Group Sifting: Sift a Group Upwards

C. Linear Group Sifting

In summary, the linear transformation has to be restricted such that

1. current state variables are combined only with current state variables, i.e. for \(\tau_x(x, y) = \bigoplus_{u \in V_x} u\) it holds that \(V_x \subseteq x\). Thus, we can write \(\tau_x(x, y) = \tau_{x_i}(x)\).
2. the transformation of current state variables (in terms of current state variables) equals the transformation of next state variables (in terms of next state variables), i.e. \(\tau_x(x) = \tau_{y_i}(y)\).

An algorithm very similar to group sifting can be used for dynamical LTBDD minimization: current and next state variables are grouped together, and these pairs of variables can be exchanged and linearly combined, i.e. if the variable ordering is given by

\[
(\ldots, x_i, y_i, x_j, y_j, \ldots),
\]
the order after exchanging the pairs is

\[
(\ldots, x_j, y_j, x_i, y_i, \ldots)
\]
and after a linear combination, it is

\[
(\ldots, x_i \oplus x_j, y_i \oplus y_j, x_j, y_j, \ldots).
\]

A sketch of the linear group sifting algorithm when exploring upwards is given in Figure 4.

IV. EXPERIMENTAL RESULTS

In this section we give experimental results derived by an own implementation of a standard ITE-based BDD package (similar to [11, 12]) running on a SUN Ultra 2 workstation with 1 GByte of main memory.

As scenario for measurements we incorporated our methods into one of the key algorithms in sequential verification, i.e. Reachability analysis of sequential circuits. The models were taken from the benchmark set of SMV-model checking traces of Yang [24].

We compared group sifting to our LTBDD approach of linear group sifting. As good compromise to trade off (LT)BDD size reduction and runtime of the minimization processes both methods were applied by using sifting for half-overlapping windows of up to 20 current-next state variable pairs per window.

Sifting was initialized whenever the number of total (LT)BDD nodes crossed a dynamic node boundary limit. The initial value for this limit equals 2 times the number of (LT)BDD nodes needed for the initial representation of the transition relation. Whenever the boundary limit was crossed its value was increased by 20 %.

All (LT)BDD node peak sizes in Table I are given in units of 1,000 nodes, runtimes (denoted by \(\text{Time}\)) are given in CPU seconds. The peak size ratio of linear group sifting/standard group sifting is abbreviated by \(LG/G\).

Another point of interest is the ratio of accepted (due to size improvements) to denied linear transformations during linear sifting (denoted by \#Acc/Den) as well as the amount of levels bearing linearly transformed variables at the end of the reachability analysis (\#Final).

Obviously linear group sifting turned out to be superior in peak size reduction due to the higher capabilities of linearly transformed BDDs.

On the other hand, the advantage in memory efficiency has a price to be paid for the runtime needed to explore the larger search space during linear group sifting. However, although it seems to be straightforward to assume that the runtime overhead for linear group sifting could as well be spent for additional standard group sifting runs
TABLE I
Comparison of Sifting and Linear Sifting

<table>
<thead>
<tr>
<th>Circuit</th>
<th>#Latches</th>
<th>Group</th>
<th>Sifting</th>
<th>Linear</th>
<th>Group</th>
<th>Sifting</th>
<th>LG/G</th>
<th>#Acc/Den</th>
<th>#Final</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>Peak</td>
<td>Size</td>
<td>Peak</td>
<td>Size</td>
<td>Time [s]</td>
<td>Time [s]</td>
<td></td>
<td></td>
</tr>
<tr>
<td>furnace17</td>
<td>92</td>
<td>97</td>
<td>5,640</td>
<td>96</td>
<td>5,650</td>
<td>98 %</td>
<td>4 %</td>
<td>38</td>
<td></td>
</tr>
<tr>
<td>over12</td>
<td>87</td>
<td>451</td>
<td>91</td>
<td>313</td>
<td>193</td>
<td>69 %</td>
<td>4 %</td>
<td>10</td>
<td></td>
</tr>
<tr>
<td>mmgt20</td>
<td>132</td>
<td>983</td>
<td>6,280</td>
<td>846</td>
<td>10,200</td>
<td>86 %</td>
<td>8 %</td>
<td>34</td>
<td></td>
</tr>
<tr>
<td>dme2-16</td>
<td>293</td>
<td>310</td>
<td>6,940</td>
<td>178</td>
<td>7,060</td>
<td>57 %</td>
<td>6 %</td>
<td>30</td>
<td></td>
</tr>
<tr>
<td>dpd75</td>
<td>300</td>
<td>453</td>
<td>5,530</td>
<td>354</td>
<td>7,650</td>
<td>78 %</td>
<td>6 %</td>
<td>32</td>
<td></td>
</tr>
<tr>
<td>ftp3</td>
<td>50</td>
<td>375</td>
<td>1,210</td>
<td>348</td>
<td>1,280</td>
<td>92 %</td>
<td>8 %</td>
<td>28</td>
<td></td>
</tr>
</tbody>
</table>

(e.g. by carrying out additional sifting runs in sequence) and achieve similarly good results for the peak sizes, according to our experience this assumption does not hold.

The ratio of accepted transformations during linear sifting (as given in Column #Acc/Den) is rather low. One might expect that the proposed method only works well in case that only few variables are transformed. Therefore, we slightly modified linear sifting: instead of accepting transformations only if the resulting LTBD is smaller, they are accepted even if its size does not change. The results are given in Table II.

It can be seen that the number of entries in the transformation matrix increases by this modification. Furthermore, both peak size and runtime do not vary significantly. The small differences of these values can be explained as random effects due to the greedy nature of (linear) sifting.

To summarize our experiments, we propose that linear group sifting on LTBDs is more profitable with respect to the cost-efficiency-tradeoff than comparable approaches for dynamic BDD size minimization based on level exchanges during sequential verification.

V. CONCLUSIONS

In this paper we introduced a concept how to benefit from the advantages of linearly transformed BDDs (LTBDs) by means of their incorporation into the sequential domain.

Although both cost aspects, runtime and memory efficiency, are of interest, the importance of “small” peak sizes in sequential verification nowadays bears highest priority, since in most cases, the controlling factor whether sequential verification processes can successfully terminate or must be cancelled is memory consumption, especially during blow-ups of the BDD peak sizes.

We have shown that efficient manipulation for LTBDs is possible in the sequential domain.

We demonstrated (for the scenario of reachability analysis) that as well as for the combinatorial domain the incorporation of LTBDs into the sequential domain yields a significant BDD node reduction potential. It was possible to reduce peak sizes by more than 15% on average when existing methods were exhausted.

Furthermore, we showed that linear group sifting on LTBDs is more profitable than comparable approaches for dynamic BDD size minimization based on level exchanges during sequential verification, offering new potentials for key problems in this domain.

REFERENCES


<table>
<thead>
<tr>
<th>Circuit</th>
<th>#Latches</th>
<th>Group Sifting Peak size</th>
<th>Linear Group Sifting Peak size</th>
<th>LG/G</th>
<th>#Acc/Den</th>
<th>#Final</th>
</tr>
</thead>
<tbody>
<tr>
<td>furnace17</td>
<td>92</td>
<td>97</td>
<td>95</td>
<td>5,904</td>
<td>97 %</td>
<td>28 %</td>
</tr>
<tr>
<td>over12</td>
<td>87</td>
<td>451</td>
<td>351</td>
<td>231</td>
<td>77 %</td>
<td>35 %</td>
</tr>
<tr>
<td>mmgt20</td>
<td>132</td>
<td>983</td>
<td>846</td>
<td>9,643</td>
<td>86 %</td>
<td>25 %</td>
</tr>
<tr>
<td>dme2-16</td>
<td>293</td>
<td>310</td>
<td>177</td>
<td>7,211</td>
<td>57 %</td>
<td>12 %</td>
</tr>
<tr>
<td>dpt75</td>
<td>300</td>
<td>453</td>
<td>389</td>
<td>8,415</td>
<td>83 %</td>
<td>12 %</td>
</tr>
<tr>
<td>ftp3</td>
<td>50</td>
<td>375</td>
<td>360</td>
<td>1,008</td>
<td>96 %</td>
<td>29 %</td>
</tr>
</tbody>
</table>


