BKFDD Document

Xuanxiang Huang
Department of Computer Science, Jinan University
Guangzhou, China
<cshxx@stu2016.jnu.edu.cn>

September 20, 2019

1 Introduction

This document shows the implementation of BKFDDs (Bi-Kronecker Functional Decision Diagrams [6]) based on CUDD-3.0 package [1]. To be compatible with CUDD's techniques, the corresponding theory of BKFDDs is modified. The theory of BKFDDs is shown in section 2, while BKFDD's group sifting is shown in section 3. Some comparisons of CUDD and BBDD are shown in section 4, and some details of BKFDD package are shown in section 5.

2 The Theory of BKFDDs

This section shows the theory of BKFDDs, which are generalization of some existing DDs (BDDs [3], FDDs [8, 7, 2], KFDDs [4], and BBDDs [1]).

2.1 Node Structures and Classical Logic Expansions

In CUDD, the only terminal node is the terminal of **1**, and the terminal of **0** is represented by the terminal of **1** with complemented edge. Figure 1 shows the BDD node structure in CUDD, where dash lines denote low-edges, solid lines represent high-edges, and dotted lines are complemented edges.

The Shannon expansion [14] is the mathematical formulation of BDD, in CUDD, it has the following form, where f_1 means $f_{x_i=1}$, and f_0 is $f_{x_i=0}$.

$$f = x_i \cdot f_1 + \bar{x}_i \cdot f_0 \text{ (S)}$$

[1] `http://vlsi.colorado.edu/~fabio/`

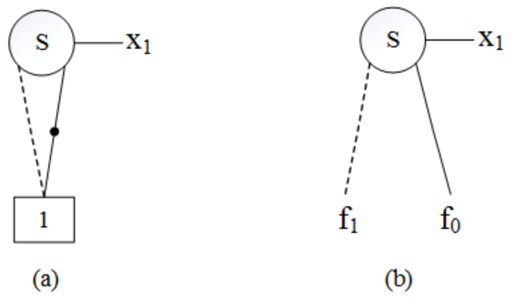

Figure 1: BDD node

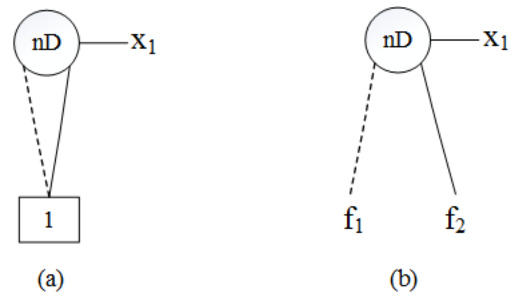

Figure 2: FDD node (decomposed by nD)

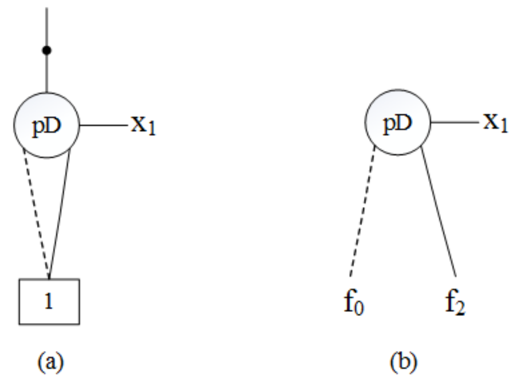

Figure 3: FDD node (decomposed by pD)

Then, it is easy to derive the negative and positive Davio expansions [2], and their form in CUDD are shown as follow:

$$f = f_1 \oplus \bar{x}_i \cdot f_2 \text{ (nD)}$$
$$f = f_0 \oplus x_i \cdot f_2 \text{ (pD)}$$

where f_2 means $f_1 \oplus f_0$. And the node structures of Davio expansions are shown in Figure 2 and 3. All three above expansions are called classical expansions in BKFDD's theory.

2.2 Biconditional Logic Expansions

Classical expansions are based on single variable decomposition, whose decomposition procedures are substituting each variable x by boolean constants (**1** or **0**). By contrast, biconditonal logic expansions are simple form of expansions based on multi-variable decomposition, and their decompositions are substituting each variable x by a boolean function of g or $\bar{g}$.

In BKFDD's theory, substituted variables are called decision variables, and substituting boolean functions are auxiliary functions that are restricted to boolean constants or variables different from decision variables. Moreover, all biconditional expansions can be reduced to three expansions[6], which are biconditional Shannon expansion, biconditional negative Davio expansion and biconditional positive Davio expansion.

$$f = (x_i \equiv x_j) \cdot f_{x_i = x_j} + (x_i \oplus x_j) \cdot f_{x_i = \bar{x}_j} \text{ (bS)}$$
$$f = f_{x_i = x_j} \oplus (x_i \oplus x_j) \cdot (f_{x_i = x_j} \oplus f_{x_i = \bar{x}_j}) \text{ (bnD)}$$
$$f = f_{x_i = \bar{x}_j} \oplus (x_i \equiv x_j) \cdot (f_{x_i = x_j} \oplus f_{x_i = \bar{x}_j}) \text{ (bpD)}$$

[2]The Davio expansion is also called the Reed-Muller expansion [10, 13]

From the perspective of linear transformation, bS-expansion can be treated as replacing x_i of S-expansion by $x_i \equiv x_j$ [5, 9].

2.3 ITE Operations

This subsection presents ITE (If-Then-Else) operation of BKFDDs, which can be used to transform expansions.

Note that $ITE(f, g, h) = f \cdot g + \bar{f} \cdot h$. Let $Z = ITE(f, g, h)$, if the top variable of Z is decomposed by S, we can derive

$$\begin{aligned} Z &= v \cdot Z_1 + \bar{v} \cdot Z_0 \\ &= v \cdot (f \cdot g + \bar{f} \cdot h)_1 + \bar{v} \cdot (f \cdot g + \bar{f} \cdot h)_0 \\ &= v \cdot (f_1 \cdot g_1 + \bar{f}_1 \cdot h_1) + \bar{v} \cdot (f_0 \cdot g_0 + \bar{f}_0 \cdot h_0) \\ &= ITE(v, ITE(f_1, g_1, h_1), ITE(f_0, g_0, h_0)) \end{aligned}$$

If the top variable of f, g, h is associated with negative Davio expansion, we have

$$\begin{aligned} Z &= Z_1 \oplus \bar{v} \cdot Z_2 \\ &= (f \cdot g + \bar{f} \cdot h)_1 \oplus \bar{v} \cdot (f \cdot g + \bar{f} \cdot h)_2 \\ &= (f \cdot g + \bar{f} \cdot h)_1 \oplus \bar{v} \cdot ((f \cdot g + \bar{f} \cdot h)_1 \oplus (f \cdot g + \bar{f} \cdot h)_0) \\ &= ITE(f_1, g_1, h_1) \oplus \bar{v} \cdot (f_1 \cdot g_1 \oplus \bar{f}_1 \cdot h_1 \oplus (f_1 \oplus f_2) \cdot (g_1 \oplus g_2) \oplus \overline{(f_1 \oplus f_2)} \cdot (h_1 \oplus h_2)) \\ &= ITE(f_1, g_1, h_1) \oplus \bar{v} \cdot (ITE(f_2, g_2, h_2) \oplus ITE(f_2, g_1, h_1) \oplus ITE(f_1, g_2, h_2) \oplus h_1 \oplus h_2) \end{aligned}$$

Replace f_1, g_1, h_1 by f_0, g_0, h_0, respectively, the formula for positive Davio expansion can be obtained.

It can be concluded that, for S, and bS expansions, $ITE(f, g, h) = ITE(\alpha, ITE(f_l, g_l, h_l), ITE(f_h, g_h, h_h))$. While for nD, bnD, pD, bpD expansions, $ITE(f, g, h) = ITE(f_l, g_l, h_l) \oplus \bar{\alpha} \cdot (ITE(f_h, g_h, h_h) \oplus ITE(f_h, g_l, h_l) \oplus ITE(f_l, g_h, h_h) \oplus h_l \oplus h_h)$, where f_l is the low successor of f, f_h is high successor of f, and $\alpha \in \{\mathbf{1}, x_i \equiv x_j\}$.

2.4 BKFDD Expansion Transformation

Transforming expansion types of DD can be used to reduce the size. There are two types of expansion modifications, the first is to change expansions between all classical ones $\{S, nD, pD\}$ or between all biconditional ones $\{bS, bnD, bpD\}$. The second is to change between classical expansions and biconditional ones.

1. For the first type:

   (a) Expansion transformation between S and nD (or bS and bnD):
   $$\begin{cases} f_{low'(v)} = f_{low(v)} \\ f_{high'(v)} = f_{low(v)} \oplus f_{high(v)} \end{cases}$$

   (b) Expansion transformation from S to pD (or bS to bpD):
   $$\begin{cases} f_{low'(v)} = f_{high(v)} \\ f_{high'(v)} = f_{low(v)} \oplus f_{high(v)} \end{cases}$$

   (c) Expansion transformation from pD to S (or bpD to bS):
   $$\begin{cases} f_{low'(v)} = f_{low(v)} \oplus f_{high(v)} \\ f_{high'(v)} = f_{low(v)} \end{cases}$$

   (d) Expansion transformation between nD and pD (or bnD and bpD):
   $$\begin{cases} f_{low'(v)} = f_{low(v)} \oplus f_{high(v)} \\ f_{high'(v)} = f_{high(v)} \end{cases}$$

2. For the second type:

(a) Expansion transformation between S and bS:

$$\begin{cases} f_{low'(v)} = ITE(x_j, f_{low(v)}, f_{high(v)}) \\ f_{high'(v)} = ITE(x_j, f_{high(v)}, f_{low(v)}) \end{cases}$$

(b) Expansion transformation from nD and bnD (or pD and bpD):

$$\begin{cases} f_{low'(v)} = ITE(x_j, f_{low(v)}, (f_{low(v)} \oplus f_{high(v)})) = f_{low(v)} \oplus \bar{x}_j \cdot f_{high(v)} \\ f_{high'(v)} = f_{high(v)} \end{cases}$$

2.5 BKFDD Symmetry

Positive and negative symmetry of BDDs can be used to group variables during symmetry sifting [12] and group sifting [11]. Symmetry of BDDs are shown in Definition 1 and Lemma 1. The first symmetric type are called positive symmetry, and the second negative symmetry.

Definition 1. *[12] A boolean function $f(x_1, ..., x_n)$ is symmetric in x_i and $x_j(\bar{x}_j)$ if the interchange of x_i and $x_j(\bar{x}_j)$ leaves the function i+dentically the same.*

Lemma 1. *[12] A boolean function f is symmetric in x_i and x_j, if and only if $f_{x_i,\bar{x}_j} = f_{\bar{x}_i,x_j}$; f is symmetric in x_i and $\bar{x}_j$ if and only if $f_{x_i,x_j} = f_{\bar{x}_i,\bar{x}_j}$.*

These symmetric types can be extended to BKFDDs:

1. $f_{\mathsf{high}(\mathsf{low}(v))} = f_{\mathsf{low}(\mathsf{high}(v))}$ or $f_{\mathsf{low}(\mathsf{low}(v))} = f_{\mathsf{high}(\mathsf{high}(v))}$
2. $f_{\mathsf{low}(\mathsf{low}(v))} \oplus f_{\mathsf{low}(\mathsf{high}(v))} = f_{\mathsf{high}(\mathsf{low}(v))}$ or $f_{\mathsf{low}(\mathsf{low}(v))} \oplus f_{\mathsf{low}(\mathsf{high}(v))} = f_{\mathsf{high}(\mathsf{high}(v))}$
3. $f_{\mathsf{low}(\mathsf{low}(v))} \oplus f_{\mathsf{high}(\mathsf{low}(v))} = f_{\mathsf{low}(\mathsf{high}(v))}$ or $f_{\mathsf{low}(\mathsf{low}(v))} \oplus f_{\mathsf{high}(\mathsf{low}(v))} = f_{\mathsf{high}(\mathsf{high}(v))}$
4. $f_{\mathsf{high}(\mathsf{low}(v))} = f_{\mathsf{low}(\mathsf{high}(v))}$ or $f_{\mathsf{high}(\mathsf{low}(v))} \oplus f_{\mathsf{low}(\mathsf{high}(v))} = f_{\mathsf{high}(\mathsf{high}(v))}$

The above are four symmetric types for BKFDD, and sequentially they are called *S-*S, *S-*D, *D-*S, and *D-*D type, respectively. The last three types are variants of the first. Equations before "or" are positive symmetry, and are negative one after "or" . Furthermore the first type can only be used to check symmetry of two adjacent S-S, bS-S, S-bS and bS-bS nodes.

2.6 Chain Reduction Rules (RC)

Figure 4 shows the chain reduction rules in [6], which can be used to further reduce DD size. Its functionality is to detect irrelevant variables (i.e., variables not in support set) and eliminate them. All these rules are originated from both positive and negative symmetry of BDDs.

Apply RC to the left DD of case (a) is the same as applying linear combination [5, 9], all other cases can be viewed as variants of case (a). Each case corresponds to a chain reduction detection type which can be derived from BKFDD's symmetric types. Replace 'or' by 'and' for all equations, and the resulting equations are called RC Detection Types.

1. $f_{\mathsf{high}(\mathsf{low}(v))} = f_{\mathsf{low}(\mathsf{high}(v))}$ and $f_{\mathsf{low}(\mathsf{low}(v))} = f_{\mathsf{high}(\mathsf{high}(v))}$
2. $f_{\mathsf{low}(\mathsf{low}(v))} \oplus f_{\mathsf{low}(\mathsf{high}(v))} = f_{\mathsf{high}(\mathsf{low}(v))}$ and $f_{\mathsf{high}(\mathsf{low}(v))} = f_{\mathsf{high}(\mathsf{high}(v))}$
3. $f_{\mathsf{low}(\mathsf{low}(v))} \oplus f_{\mathsf{high}(\mathsf{low}(v))} = f_{\mathsf{high}(v)}$
4. $f_{\mathsf{high}(\mathsf{low}(v))} = f_{\mathsf{high}(v)}$

Specifically, type 1 (bS-*S type) can be used to detect case (a), type 2 (bS-*D type) fits case (b) and (c), type 3 (bD-*S type) is for case (d), and type 4 (bD-*D type) applies for case (e) and (f).

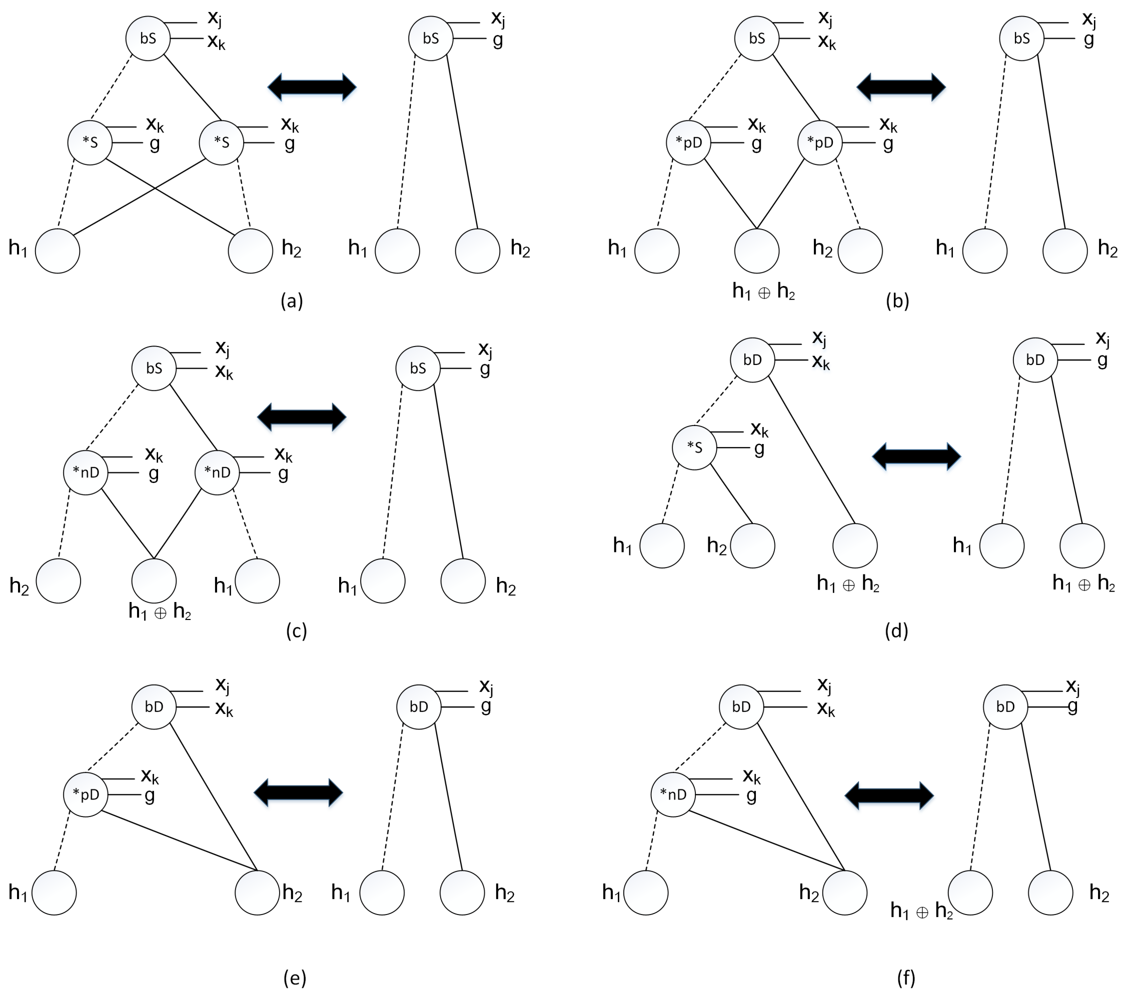

Figure 4: Chain Reduction Rules

2.7 Integrating Irrelevant Variables in DD nodes

The effect of RC is to eliminate irrelevant variables. However, the sources of introducing irrelevant variables are biconditional expansions and linear transformation. Decomposing functions by biconditional expansions will integrate two variables in DD nodes, and applying linear transformation to BDDs will put many variables in DD nodes, although the resulting DDs have smaller size, variables integrated in some DD nodes may turn out not in the support set of functions represented by these DD nodes.

2.7.1 Biconditional Expansions

For biconditional expansions, given variable order $x_1, ..., x_i, x_{i+1}, ..., x_j$ and a BBDD graph G representing function $(x_i \equiv x_j) \cdot h_1 + (x_i \oplus x_j) \cdot h_2$, then functions $(x_k \equiv x_j) \cdot h_1 + (x_k \oplus x_j) \cdot h_2$ and $(x_k \equiv x_j) \cdot h_2 + (x_k \oplus x_j) \cdot h_1$, where $i < k < j$, are contained in the subgraph of G.

In Figure 5, the functions are $(x_i \equiv x_j) \cdot h_1 + (x_i \oplus x_j) \cdot h_2$, although variables x_{i+1}, x_{i+2} are excluded in the support set of function $(x_i \equiv x_j) \cdot h_1 + (x_i \oplus x_j) \cdot h_2$, they are still contained in the left BBDD.

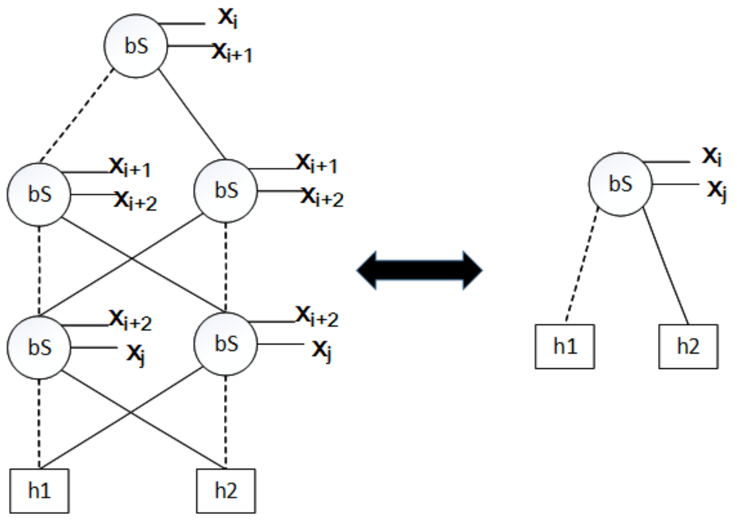

Figure 5: Integrating irrelevant variables in BBDD nodes, BBDD on the right is RC reduced

2.7.2 Linear Transformation

For linear transformation, in Figure 6, from (a) to (b), replacing x_{i+1} by $x_{i+1} \equiv x_j$ can reduce the size of the subgraph rooted at node 3, but increase the size of subgraph rooted at node 2. Reversely, from (b) to (a), apply linear transformation can reduce the size of the subgraph rooted at node2, but increase the size of the subgraph rooted at node 3. Moreover, applying linear transformation to x_{i+1} in both directions has no effect on the size of the graph rooted at node 1, because variable x_j is not in the support set of the function represented by the subgraph rooted at node 2, and $x_{i+1} \equiv x_j$ is in the support set of the function represented by the subgraph rooted at node 3.

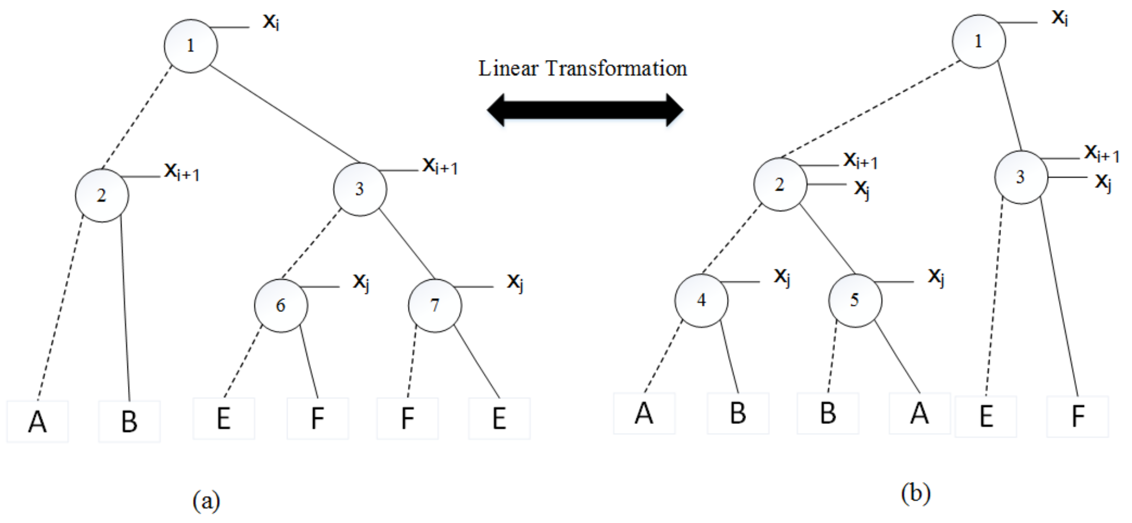

Figure 6: Limitation of linear transformation

If linear transformation is applied locally, for example, doing linear transformation only to the node 3, then one can get a new LTBDD, shown in Figure 7. So running linear transformation locally has the same functionality as the chain reduction.

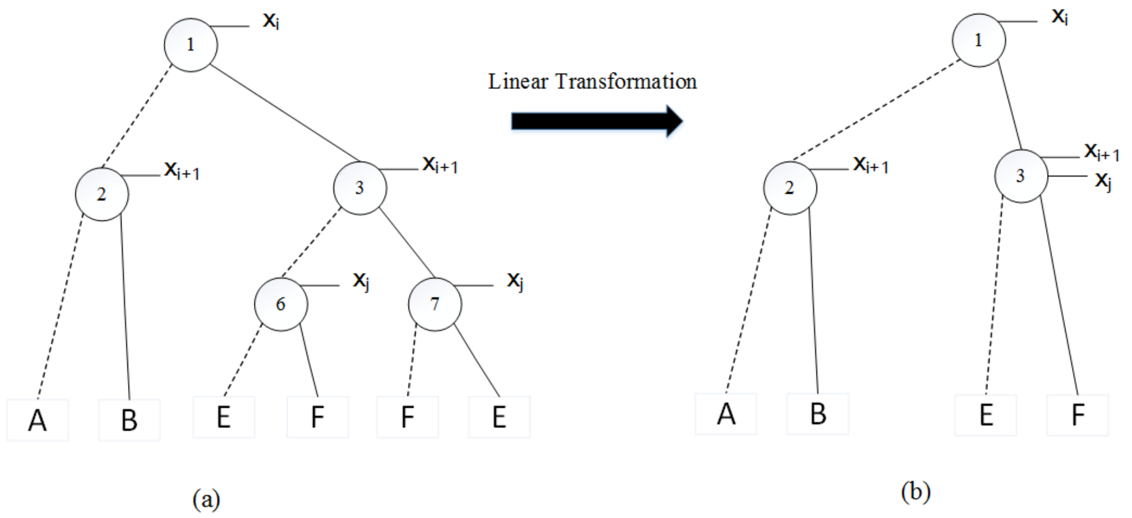

Figure 7: Applying linear transformation locally

2.8 Limitation of Chain Reduction Rules

RC can be used to eliminate variables not in the support set of functions. But this does not mean that RC always is effective. There can be an exception, namely, eliminating irrelevant variables has no effect on the size. For example, by applying RC, one can reduce Figure 8(a) to Figure 8(b). Note that the size of Figure 8(a) is the same as Figure 8(b). It can be concluded that if all internal nodes of a graph G_A referenced by nodes out of G_A, then RC has no effect on $|G_A|$.

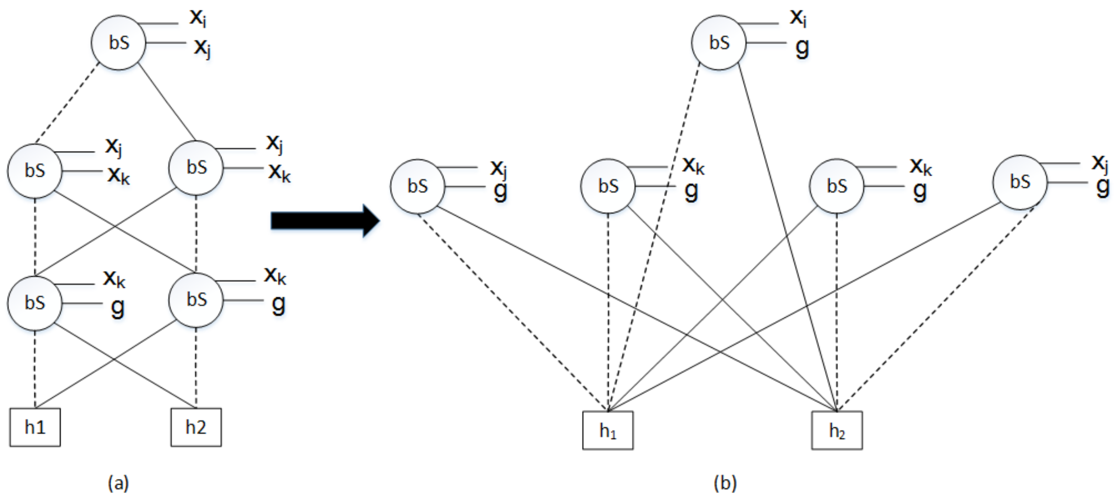

Figure 8: Limitation of Chain-Reduction Rules

Moreover, applying RC and reversely applying RC to non-Shannon nodes need extra computation overhead. In our BKFDD package, RC is carried out after sifting is done.

3 Group Sifting

This section shows BKFDD's group sifting algorithm, which is the extension of BDD's group sifting.

3.1 OET Decomposition

OET (variable order with expansion types) is a data structure in BKFDDs to store variable order and expansion associated with each variable.

Definition 2. A *variable order with expansion types (OET)* π is a sequence of n pairs (x, e) where $x \in \mathbf{X}$ and $e \in \{S, pD, nD, bS, bpD, bnD\}$ such that the variables of any two distinct pairs are different.

In BKFDD's group sifting, the first step is to divide OET into atoms, by means of expansion type.

Definition 3. *For each atom OET* $\psi = [\pi_i, ..., \pi_j]$, *which contains* m *elements, s.t.* $\forall i < m.\psi_i^e \in \{bS, bpD, bnD\} \wedge \psi_m^e \in \{S, pD, nD\}$, *or* $\psi_1^e \in \{S, pD, nD\} \wedge m = 1$.

The first kind of atom OETs are called biconditional group, denoted by $group_B$. The second are called single variable group, denoted by $group_V$. These two groups are atoms which cannot be decomposed any further, but can be merged with others to form a new group as follow.

$$OET_{atom} := group_B \mid group_V$$
$$OET_{sub} := OET_{atom} \mid OET_{sub} + OET_{sub}$$

A group is named as a classical group if all it contains are single variable group, denoted by $group_C$. All other groups are called mix group, denoted by $group_M$.

3.2 Strategy of Merging Groups

BKFDD symmetry property involves computation overhead, which is inefficient in BKFDD's group sifting. For practical reasons, all conditions should be modified, according to BKFDD's symmetric type and RC detection type, checking conditions are transformed to:

1. $f_{\mathsf{high}(\mathsf{low}(v))} = f_{\mathsf{low}(\mathsf{high}(v))}$ or $f_{\mathsf{low}(\mathsf{low}(v))} = f_{\mathsf{high}(\mathsf{high}(v))}$
2. $f_{\mathsf{high}(\mathsf{low}(v))} = f_{\mathsf{high}(\mathsf{high}(v))}$ or $f_{\mathsf{high}(\mathsf{low}(v))} = f_{\mathsf{low}(\mathsf{high}(v))}$
3. $f_{\mathsf{low}(\mathsf{high}(v))} = f_{\mathsf{high}(\mathsf{high}(v))}$ or $f_{\mathsf{high}(\mathsf{low}(v))} = f_{\mathsf{low}(\mathsf{high}(v))}$
4. $f_{\mathsf{high}(\mathsf{high}(v))} = \mathbf{0}$ or $f_{\mathsf{high}(\mathsf{low}(v))} = f_{\mathsf{low}(\mathsf{high}(v))}$

Note that condition $f_{\mathsf{high}(\mathsf{low}(v))} = f_{\mathsf{low}(\mathsf{high}(v))}$ is added to 2, 3, 4, this is because BDD's positive symmetry property can be applied in non-Shannon decomposed nodes, since they share the same variable swapping method.

There should be experiments to evaluate checking conditions with and without $f_{\mathsf{high}(\mathsf{low}(v))} = f_{\mathsf{low}(\mathsf{high}(v))}$. Besides, there are other methods to merge groups, in our BKFDD package, strategy of second difference [11] is used to merge two $group_V$ into a $group_C$.

3.3 Group Sifting Algorithm

There are two phases of BKFDD's group sifting algorithm. Given a BKFDD and its OET π, the first phase is to sift BKFDD and generate a new OET π', the second is to choose better expansions for each π' elements. Algorithm 1 shows the framework of BKFDD's group sifting. OET_{atom} of π and π' are the same, namely, variable order and expansion types of each OET atom are fixed during reordering.

```
Algorithm 1: BKFDDs Group Sifting
1 Sifting BKFDDs and generate a new OET $\pi'$;
2 Choose better expansions for each $\pi'_i$(except $\pi'_n$) to generate $\pi''$;
```

The algorithm for the first phase is shown in Algorithm 2. Line 1 is to detect OET_{atom} of given π and store OET_{atom}. Line 2 is to determine the sifting order of each OET_{atom}, in general, the number of nodes of each OET_{atom} is the key of sifting. After that(line 3-9), each OET_{atom} will be sifted to find a better position in OET. For $group_B$, they will be sifted only, while for $group_V$, they will be merged when sifting. In line 10-15, if OET_{atom} just sifted is $group_V$, and adjacent OET_{sub} are $group_V$, then method of second difference will be used to check whether they can be merged to form a $group_C$. If they can, then newly created $group_C$ will be sifted again. In line 17, all $group_M$ and $group_C$ will be dissolved, while $group_B$ and $group_V$ will be retained.

Algorithm 2: Sifting Procedure

```
1  Detect $OET_{atom}$ and stored;
2  Determine sifting order of $OET_{atom}$;
3  foreach $OET_{atom}$ do
4      if $OET_{atom}$ is $group_B$ then
5          Sift only
6      end
7      else
8          Merge adjacent $OET_{sub}$ to form a new group($group_V$ or $group_M$) while sifting
9      end
10     if $OET_{atom}$ and adjacent $OET_{sub}$ are groupv then
11         Use second difference method to check whether they can be merged to form a $group_C$
12     end
13     if a new $group_C$ formed then
14         Sifting newly formed $group_C$
15     end
16 end
17 Dissolve all $group_M$ and $group_C$;
```

Moreover, two algorithms (Algorithm 3 and 4) are provided for the second phase. The difference between them is that $\pi_i'^e \in \{S, nD, bS, bnD\}$ is in Algorithm 3, while $\pi_i'^e \in \{S, nD, bS, bnD, pD, bpD\}$ is in Algorithm 4. In both algorithms, $\pi_n'^e$ is always treated as Shannon expansion since $\pi_n'^e$ has no effect on the DD size.

The reason why we provide two algorithms for the second phase are:

- If all nodes decomposed by $expn$ ($expn \in \{S, bS, nD, bnD\}$) are in canonical form, then there is no guarantee that these nodes remain canonical after changing $expn$ to $expn'$($expn' \in \{pD, bpD\}$). But it can remain canonical after changing $expn$ to $expn'$ ($expn' \in \{S, bS, nD, bnD\}$).

- If all nodes decomposed by $expn$ ($expn \in \{pD, bpD\}$) are canonical, then the canonicity of these nodes cannot be guaranteed after changing $expn$ to $expn'$ ($expn' \in \{S, bS, nD, bnD, pD, bpD\}$).

In Algorithm 3, if $\pi_i'^x$ and $\pi_{i+1}'^x$ interacts [11, 15], better expansions will be chosen from $\{S, nD, bS, bnD\}$, otherwise, there is no need to combine these two adjacent variables, expansion types can be chosen from either $\{S, nD\}$ or $\{bS, bnD\}$.

Algorithm 3: Choose a Better BKFDD Procedure (without pD and bpD)

```
1  foreach $\pi_i'$ do
2      if $\pi_i'^x$ and $\pi_{i+1}'^x$ interacts then
3          Choose a better expansion for $\pi_i'^e$ from $\{S, nD, bS, bnD\}$
4      end
5      else
6          if $\pi_i'^e \in \{S, nD\}$ then
7              Choose a better expansion for $\pi_i'^e$ from $\{S, nD\}$
8          end
9          else
10             Choose a better expansion for $\pi_i'^e$ from $\{bS, bnD\}$
11         end
12     end
13 end
```

In Algorithm 4, changing expansion types will make nodes violate "low-edge regular" principle, and thus become uncanonical. So fixing canonicity of nodes is necessary, and needs extra runtime. To reduce runtime, it is better to determine better expansions for each π'_i in the top-down manner, and the reasons are as follow.

- Changing expansion types of node v only needs subgraphs rooted at $\mathsf{low}(v)$ and $\mathsf{high}(v)$, and does not affect the canonicity of nodes below v.
- All nodes above v will be involved in the canonicity fixing procedure.

Algorithm 4: Choose Better BKFDD Procedure

```
foreach $\pi'_i$ do
    if $\pi'^x_i$ and $\pi'^x_{i+1}$ interacts then
        Choose a better expansion for $\pi'^e_i$ from $\{S, nD, bS, bnD, pD, bpD\}$
    end
    else
        if $\pi'^e_i \in \{S, nD, pD\}$ then
            Choose a better expansion for $\pi'^e_i$ from $\{S, nD, pD\}$
        end
        else
            Choose a better expansion for $\pi'^e_i$ from $\{bS, bnD, bpD\}$
        end
    end
    Fix canonicity of nodes;
end
```

In the top-down manner, we only need to fix canonicity of nodes after changing expansion types for all π'_i, but in the bottom-up manner, we must fix canonicity of nodes after changing expansion types for every π'_i.

There are two modes, namely SND mode and SD mode, in our BKFDD package. Algorithm 3 is applied in SND mode, while Algorithm 4 is applied in SD mode. For practical reasons, we provide some arguments in both to reduce search space, which will be introduced later.

4 Remarks on CUDD and BBDD Package

1. Dynamic reordering in CUDD is triggered by node count, but in BBDD [3] (and Puma[4]), it is triggered after boolean operations. So CUDD's strategy can be more effective.
2. CUDD has interacting matrix[15] to detect unnecessary variable swap, i.e., if two adjacent variables are not interacting, swapping them costs constant time. But this interacting matrix technique cannot be applied in BBDD, because in each of BBDD's CVO swap, successors of three adjacent variables need to be adjusted. So all CVO swaps in BBDD have the same bound.
3. BBDD's CVO swap can be implemented by linear combination and variable swap, which is shown in Figure 9.
4. BBDD's R4 is a special case of BKFDD's RC. Applying RC only to single variable is the same as applying R4 to BBDDs. So if all variables are in the support set, R4 has no effect on BBDD's size.
5. From CUDD's perspective, BBDD's R4 is to eliminate isolated projection functions in CUDD. Note that in implementation of CUDD, the number of isolated projection functions are always excluded in the DD size, which actually has same functionality as BBDD's R4.

Some experimental results are stored in "cudd_bkfdd_kfdd_comp" directory.

[3]`https://lsi.epfl.ch/BBDD`
[4]`https://ira.informatik.uni-freiburg.de/software/puma/pumamain.html`

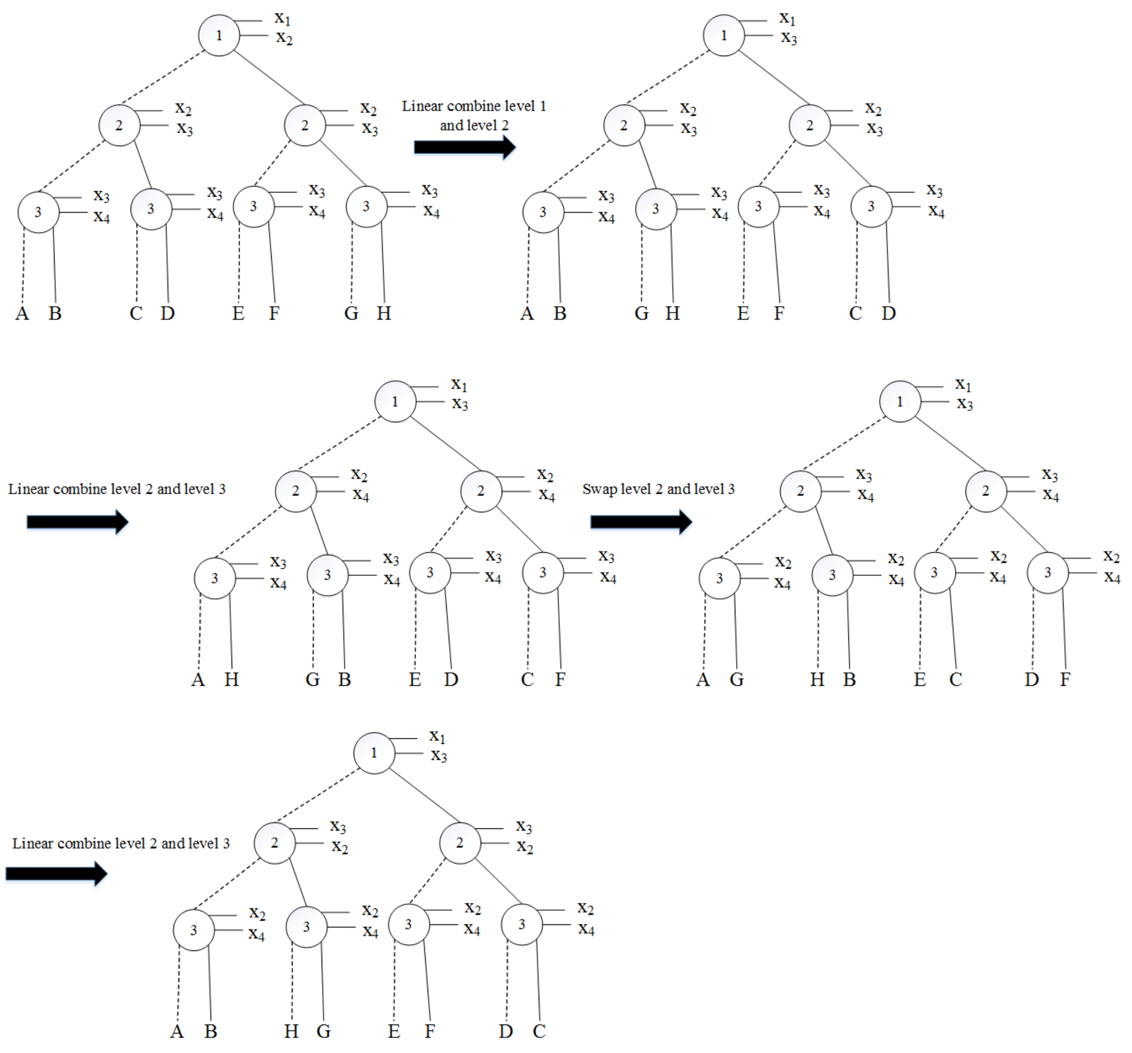
Linear combine level 1 and level 2
Linear combine level 2 and level 3
Swap level 2 and level 3
Linear combine level 2 and level 3

Figure 9: BBDD CVO swap (implemented by linear combination and variable swap)

5 BKFDD Package

5.1 Installing and Cleanning

BKFDD is integrated in CUDD, which can be installed and cleaned along with CUDD, see the README file of the CUDD package.

5.2 Source Files

The source files of BKFDD package are contained in "bkfdd" and "testbkfdd" directories.

5.3 Input and Output

BKFDD reads BLIF file and generates BLIF file as its output. Functions for dumping BKFDD in BLIF are stored in "bkfddDump.c".

5.4 Arguments

There are some arguments to control the procedure of choosing better expansions.

- Use “-sdmode” to active SD mode, the default mode of BKFDD is SND mode.
- Use “-davioexist x” to restrict the maximal number of $\{nD, bnD, pD, bpD\}$ contained in DD with $x \in [0, 100]$, if there are y variables, then there are at most $y \times x/100$ number of $\{nD, bnD, pD, bpD\}$ allowed to exist in DD. The default value of x is 15.
- Use “-chooselowbound x” to quickly end the procedure of choosing better expansions if the procedure has reduce (1-x) percent of DD nodes, and the default value of x is 70.
- Use “-choosenew x” to determine when to accept new expansions $expn'$. And $expn'$ will be accepted if changing expansion from $expn$ to $expn'$ makes the size of DD be from y_1 to y_2 s.t. $y_2 < (y_1 \times x/10000)$. The default value of x is 10000.
- Use “-choosedav x” to determine when to accept $expn'$ ($expn' \in \{nD, bnD, pD, bpD\}$). And $expn'$ will be accepted if changing expansions from $expn$ ($expn \in \{S, bS\}$) to $expn'$ makes the size of DD be from y_1 to y_2 s.t. $y_2 < (y_1 \times x/10000)$. The default value of x is 10000.
- If the current expansion of π'_i is the best after we check different expansion types, then we fail to choose better expansion for π'_i. Use “-choosefail x” to quickly end the procedure of choosing better expansion if we have failed enough times.

5.5 Benchmarks and Experimental Results

Benchmarks are contained in “testbkfdd” and “nanotrav”, and can be downloaded from `https://ddd.fit.cvut.cz/prj/Benchmarks/`.

Some experimental results are stored in “cudd_bkfdd_kfdd_comp” directory.

6 Feedback

If you experience any bug of BKFDD package, please contact us at cshxx@stu2016.jnu.edu.cn.

References

[1] Luca Amarú, Pierre-Emmanuel Gaillardon, and Giovanni De Micheli. Biconditional binary decision diagrams: A novel canonical logic representation form. *IEEE Journal on Emerging and Selected Topics in Circuits and Systems*, 4(4):487–500, 2014.

[2] Bernd Becker, Rolf Drechsler, and Michael Theobald. On the implementation of a package for efficient representation and manipulation of functional decision diagrams. In *Proc. of IFIP WG 10.5 Workshop Applications of the Reed-Muller Expansion in Circuit Design*, pages 162–169, 1993.

[3] Randal E. Bryant. Graph-Based Algorithms for Boolean Function Manipulation. *IEEE Trans. Comput.*, 100(8):677–691, 1986.

[4] Rolf Drechsler and Bernd Becker. Ordered kronecker functional decision diagrams-a data structure for representation and manipulation of boolean functions. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 17(10):965–973, 1998.

[5] W Gunther and Rolf Drechsler. Efficient minimization and manipulation of linearly transformed binary decision diagrams. *IEEE Transactions on Computers*, 52(9):1196–1209, 2003.

[6] Xuanxiang Huang, Kehang Fang, Liangda Fang, Qingliang Chen, Zhao-Rong Lai, and Linfeng Wei. Bi-kronecker functional decision diagrams: A novel canonical representation of boolean functions. In *The Thirty-Third AAAI Conference on Artificial Intelligence, AAAI 2019, Honolulu, Hawaii, USA, January 27 - February 1, 2019*, pages 2867–2875, 2019.

[7] U. Kebschull, E. Schubert, and W. Rosenstiel. Efficient Graph-Based Computation and Manipulation of Functional Decision Diagrams. In *EDAC*, pages 278–282, 1993.

[8] Udo Kebschull, Endric Schubert, and Wolfgang Rosenstiel. Multilevel logic synthesis based on functional decision diagrams. In *[1992] Proceedings The European Conference on Design Automation*, pages 43–47. IEEE, 1992.

[9] Christoph Meinel, Fabio Somenzi, and Thorsten Theobald. Linear sifting of decision diagrams. In *Proceedings of the 34th annual Design Automation Conference*, pages 202–207. ACM, 1997.

[10] David E. Muller. Application of Boolean Algebra to Switching Circuit Design and to Error Detection. *Trans. of the IRE Professional Group on Electronic Computers*, 3(3):6–12, 1954.

[11] Shipra Panda and Fabio Somenzi. Who are the variables in your neighborhood. In *Proceedings of the 1995 IEEE/ACM international conference on Computer-aided design*, pages 74–77. IEEE Computer Society, 1995.

[12] Shipra Panda, Fabio Somenzi, and Bernard F Plessier. Symmetry detection and dynamic variable ordering of decision diagrams. In *Proceedings of the 1994 IEEE/ACM international conference on Computer-aided design*, pages 628–631. IEEE Computer Society Press, 1994.

[13] Irving S. Reed. A class of multiple-error-correcting codes and their decoding scheme. *Trans. of the IRE Professional Group on Information Theory*, 4(4):38–49, 1954.

[14] Claude E. Shannon. A Symbolic Analysis of Relay and Switching Circuits. *Trans. of the American Institute of Electrical Engineers*, 57(12):713–723, 1938.

[15] Fabio Somenzi. Efficient manipulation of decision diagrams. *International Journal on Software Tools for Technology Transfer*, 3(2):171–181, 2001.