Int J STTT (2001) 3: 112–136 / Digital Object Identifier (DOI) 10.1007/s100090100056

Special section on BDD

Binary decision diagrams in theory and practice

Rolf Drechsler[1], **Detlef Sieling**[2]

[1] Corporate Technology, Siemens, 81730 Munich, Germany; E-mail: rolf.drechsler@mchp.siemens.de
[2] FB Informatik, LS 2, University of Dortmund, 44221 Dortmund, Germany; E-mail: sieling@ls2.cs.uni-dortmund.de

Published online: 15 May 2001 –

Abstract. Decision diagrams (DDs) are the state-of-the-art data structure in VLSI CAD and have been successfully applied in many other fields. DDs are widely used and are also integrated in commercial tools. This special section comprises six contributed articles on various aspects of the theory and application of DDs. As preparation for these contributions, the present article reviews the basic definitions of binary decision diagrams (BDDs). We provide a brief overview and study theoretical and practical aspects. Basic properties of BDDs are discussed and manipulation algorithms are described. Extensions of BDDs are investigated and by this we give a deeper insight into the basic data structure. Finally we outline several applications of BDDs and their extensions and suggest a number of articles and books for those who wish to pursue the topic in more depth.

Key words: Binary decision diagram – Branching program – data structure – Boolean function – VLSI CAD

1 Introduction

Decision diagrams (DDs) are the most frequently used data structure for representation and manipulation of Boolean functions in the area of VLSI CAD. *Binary decision diagrams* (BDDs) as a data structure for representation of Boolean functions were first introduced by Lee [94] and further popularized by Akers [2] and Moret [110]. In the restricted form of reduced *ordered BDDs* (OBDDs) they gained widespread application because OBDDs are a canonical representation and allow efficient manipulations as proved by Bryant [23]. Some fields of application in VLSI CAD are verification, test generation, fault simulation, and logic synthesis. However, in other areas, such as SAT-solving, OBDDs have also been successfully used. OBDDs have been studied from a theoretical and practical point of view for several years. They have become a very popular data structure, since they allow efficient representation of many functions occurring in practice. Furthermore, the data structure can be efficiently implemented on modern computers using a programming language such as *C*. It is important to note that BDDs (also called *branching programs* (BPs) in this context) have been studied theoretically for a long time, but these results were (nearly) unknown in the "VLSI CAD community."

One major contribution of this article is to point out the close relation between the two "different" fields and it turns out that they are effectively two sides of the same coin. In this special section we introduce the basic concept of OBDDs and discuss several aspects of this data structure. Even though the contributed articles mainly cover the practical aspects, in this overview article several theoretical aspects are also covered. Before we give a detailed outline of this overview, we sketch the contents of the contributed articles (also see Sect. 6).

In the first article by Bryant and Chen, an extension of BDDs to represent integer-valued functions is proposed and the application in formal verification is studied. The paper by Minato introduces a DD extension that is well suited for representation of sets of combinations and in this context outperforms "classic" BDDs. The two papers by Somenzi and Höreth focus on implementation techniques of BDD and integer-valued DDs, respectively. Optimization techniques are described regarding how to speed up computations when using DDs in real-world applications. The article by Harlow and Brglez describes

Supported in part by DFG grant We 1066/9.

a first step towards experimental evaluation of experiments on the robustness of BDDs. Finally, an application of BDDs, i.e., Boolean matching, is discussed in the work of Mohnke, Molitor, and Malik.

This overview article is structured as follows: in Sect. 2 OBDDs are introduced and some basic notations and definitions are given. Examples are also provided to simplify understanding for non-experts. Basic properties, such as canonicity, are considered. Manipulation algorithms on OBDDs are discussed and their complexity is analyzed. Furthermore, we study the problem of variable ordering for OBDDs.

In Sect. 3 generalizations of the basic OBDD concept are discussed. We show that for some functions by changing the decomposition rule carried out in each node the size can be reduced by an exponential factor. Then we study the effect when the variable ordering is relaxed, and when additional transformations may be applied to the represented functions. Finally, in this section we consider DDs for integer-valued functions, i.e., so-called word-level DDs, that have shown to work very well in the area of arithmetic circuit verification.

Several different applications of BDDs and their generalizations are outlined in Sect. 4. We first consider applications from the field of VLSI CAD, such as verification, logic synthesis, and testing. However, we also show that DDs are very useful in other areas of combinatorial optimization. At the end of Sect. 4 we briefly discuss the use of BDDs in complexity theory as a model of sequential computation.

The main conclusions are summarized in Sect. 5 where we also give pointers to further literature. Finally, we give a more detailed overview of the contributed articles in Sect. 6.

2 Ordered binary decision diagrams

2.1 Definition and examples of OBDDs

An OBDD is a graphic description of an algorithm for the computation of a Boolean function. The following definition describes the syntax of OBDDs, i.e., the properties of the underlying graph. The semantics of OBDDs, i.e., the functions represented by OBDDs, is specified in definition 2.

Definition 1. An OBDD G representing the Boolean functions $f^1, \ldots, f^m$ over the variables $x_1, \ldots, x_n$ is a directed acyclic graph with the following properties:

1. For each function f^i there is a pointer to a node in G.
2. The nodes without outgoing edges, which are called *sinks* or *terminal nodes*, are labeled by 0 or 1.
3. All non-sink nodes of G, which are also called *internal nodes*, are labeled by a variable and have two outgoing edges, a 0-edge and a 1-edge.
4. On each directed path in the OBDD each variable occurs at most once as the label of a node.
5. There is a variable ordering π, i.e., a permutation of $x_1, \ldots, x_n$, and on each directed path the variables occur according to this ordering. This means, if x_i is arranged before x_j in the variable ordering, then it must not happen that on some path there is a node labeled by x_j before a node labeled by x_i.

In the figures we draw sink nodes as squares and internal nodes as circles. We always assume that edges are directed downwards. 0-edges are drawn as dashed lines while 1-edges are drawn as solid lines. Figure 1 shows an OBDD G_f with the variable ordering x_1, x_3, x_2 and an OBDD G_g with the variable ordering x_1, y_1, x_0, y_0.

Definition 2. Let G be an OBDD for the functions $f^1, \ldots, f^m$ over the variables $x_1, \ldots, x_n$, and let $a = (a_1, \ldots, a_n)$ be an input. The *computation path* for the node v of G and the input a is the path starting at v which is obtained by choosing at each internal node labeled by x_i the outgoing a_i-edge.

Each node v represents a function f_v, where $f_v(a)$ is defined as the value of the sink at the end of the computation path starting at v for the input a. Finally, f^j is defined as the function represented at the head of the pointer for f^j.

Definition 2 can be seen as the description of an algorithm to obtain for each function f^j and each input a the computation path and, therefore, the value $f^j(a)$.

In the OBDD G_f in Fig. 1 the computation path for the input $(x_1, x_2, x_3) = (1, 1, 0)$ is indicated by a dotted line. Furthermore, for each node v of G_f the function f_v represented at v is given. By definition 2 it is easy to verify that the OBDD G_f in Fig. 1 represents the function $f(x_1, x_2, x_3) = x_1 \oplus x_2 \oplus x_3$ and the OBDD G_g represents the function $g(x_1, y_1, x_0, y_0) = (s_2, s_1, s_0)$, where (s_2, s_1, s_0) is the sum of the two 2-bit numbers (y_1, y_0) and (x_1, x_0). It suffices to consider all possible inputs and to compute the values of the functions using definition 2. Of course, this is not feasible for functions with a larger number of input variables. Another possibility to obtain the functions represented by an OBDD is to consider the relation between the functions f_v represented at the nodes of the OBDD.

Obviously, the function represented at the sink labeled by $c \in \{0, 1\}$ is the constant function c. Now let v be an internal node which is labeled by x_i. Let v_0 be the 0-successor of v, i.e., the node reached via the 0-edge leaving v, and let v_1 be the 1-successor of v. We consider the computation of f_v for some input. If in the input the value of x_i is 0, then by definition 2 we may obtain f_v by evaluating f_{v_0} and, if the value of x_i is 1, by evaluating f_{v_1}. This can be expressed by the equation

$$f_v = \overline{x}_i f_{v_0} \vee x_i f_{v_1}. \tag{1}$$

Using equation (1) we may compute the functions represented at the nodes of an OBDD in a bottom-up fashion.

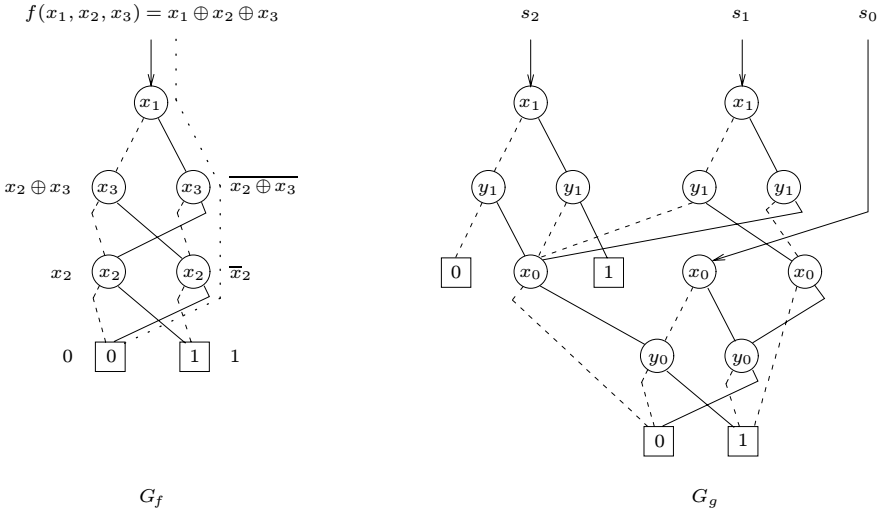

Fig. 1. Examples of OBDDs

However, the opposite is also true. If a node v labeled by x_i represents the function f_v, then the 0-successor of v represents the subfunction (sometimes called cofactor) $f_{v|x_i=0}$ and the 1-successor the subfunction $f_{v|x_i=1}$. In other words, at v the function f_v is decomposed using *Shannon's decomposition rule*

$$f_v = \overline{x}_i f_{v|x_i=0} \vee x_i f_{v|x_i=1}. \tag{2}$$

We point out that in Sect. 3 we shall consider variants of OBDDs where Shannon's decomposition rule is replaced by a different decomposition rule.

From equation (2) we see that we decompose the function f_v in different ways if we choose different variables x_i for the decomposition. Hence, we may get different OBDDs for the same function if we use different variable orderings. Later on, we shall see that the size of an OBDDs depends on the chosen variable ordering.

Finally, we note that in implementations a slightly different version of OBDDs, namely OBDDs with complemented edges (sometimes called output inverters), is used. Complemented edges were suggested by Akers [2], Minato, Ishiura and Yajima [108] and Brace, Rudell and Bryant [21]. On each edge and each pointer for a function f^j there may be a complement attribute. Then $f^j(a)$ is computed as described in definition 2 but the value obtained by the computation is complemented if the number of complement attributes encountered on the computation path is odd. The use of complemented edges allows us to obtain an OBDD for $\overline{f}$ from an OBDD for f in constant time. Furthermore, the number of nodes of OBDDs with complemented edges is usually smaller than the number of nodes of OBDDs without complemented edges. On the other hand, the description of properties of OBDDs and algorithms on OBDDs is more involved for OBDDs with complemented edges. Hence, in the following, we consider OBDDs without complemented edges unless stated explicitly.

2.2 The canonicity of reduced OBDDs

An important property of OBDDs is that they are a canonical representation of Boolean functions. Canonicity means that for all Boolean functions $f^1, \ldots, f^m$ and each variable ordering π there is a unique OBDD G which can be obtained from each OBDD for $f^1, \ldots, f^m$ and π without much effort. The computation of G from some OBDD for $f^1, \ldots, f^m$ and π is called *reduction*, since it is performed by applying *reduction rules*, and G is called *the reduced OBDD* for $f^1, \ldots, f^m$ and π. An obvious advantage of canonical representations is that two functions are identical iff their reduced OBDDs are identical. Furthermore, we shall see that reduced OBDDs for $f^1, \ldots, f^m$ and the variable ordering π are of minimum size among all OBDDs for $f^1, \ldots, f^m$ and π.

In the following, we always assume that the OBDDs only contain nodes that are reachable from the pointer of some function f^j. Each node not reachable in this way can be removed since it does not affect any computation path.

The reduction of OBDDs is based on only two reduction rules, the *S-deletion rule* and the *merging rule*. The main idea of the reduction rules is to remove redundancies from the OBDD, namely, superfluous tests of variables and tests that are represented more than once. Both reduction rules are sketched in Fig. 2.

The S-deletion rule can be applied to nodes v for which both outgoing edges lead to the same node w. It is obvious that we can redirect all edges leading to v to the node w and that we can delete v afterwards without changing the function represented by the OBDD. We use the term S-deletion rule (for Shannon deletion rule) instead of simply deletion rule in order to distinguish this rule from a different deletion rule for the variants of OBDDs which are not based on Shannon's decomposition rule.

The merging rule is applicable if there are nodes v and w with the same label, the same 0-successor, and the same 1-successor. Again, it is obvious that we can redirect all

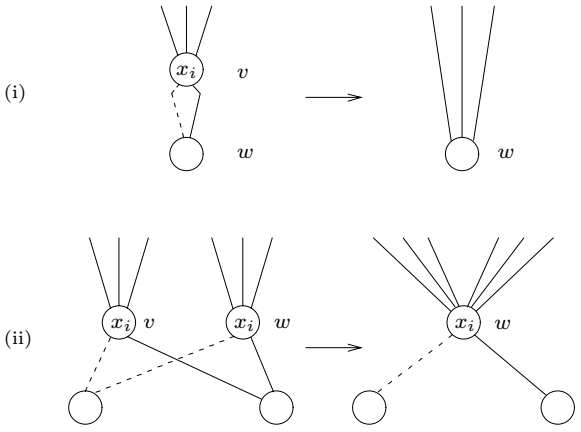

Fig. 2. The (i) S-deletion rule and the (ii) merging rule for OBDDs

edges leading to v to the node w and that we can delete v afterwards without changing the represented function. Furthermore, the same operation can be performed on sinks with the same label.

Bryant [23] proved that these reduction rules suffice to obtain the canonical representation for each function and each variable ordering.

Theorem 1. *For all functions $f^1, \ldots, f^m$ and for each variable ordering π there is a unique OBDD G which can be obtained from each OBDD for $f^1, \ldots, f^m$ and π by applying the S-deletion rule and the merging rule until neither of the rules is applicable.*

In particular, the reduction of an OBDD yields the same result even if the reduction rules are applied in a different order. Theorem 1 also implies that the functions f^i and f^j represented in the same reduced OBDD are identical iff the pointers for f^i and f^j lead to the same node.

An example of the application of the reduction rules is shown in Fig. 3. Bryant [23] constructed an efficient algorithm for the reduction of OBDDs. To make a reduction algorithm efficient it is essential that the reduction rules are applied levelwise bottom-up. The reason is that a reduction rule may become applicable to a node v if we apply some reduction rule to some successor of v. On the other hand, a reduction rule cannot become applicable to v if we apply some reduction rule to some predecessor of v (see Fig. 3). Hence, if during the bottom-up traversal on some node neither reduction rule is applicable, it cannot happen that later on some reduction rule is applicable. The algorithm of Bryant has on input G a run time of $O(|G| \log |G|)$ where $|G|$ denotes the number of nodes of G. An algorithm with linear run time $O(|G|)$ was presented by Sieling and Wegener [124].

We do not present a complete proof of theorem 1. In order to get a feeling why reduced OBDDs are unique we describe which functions are computed at the nodes of a reduced OBDD. In order to simplify the presentation we only consider OBDDs for a single function f and we fix the variable ordering $x_1, \ldots, x_n$. This is no restriction since we may rename the variables. Let v be the node at the head of the pointer for f.

Consider a partial assignment $x_1 = c_1, \ldots, x_{i-1} = c_{i-1}$. If we follow the computation path which starts at v and which is chosen for this partial assignment, we reach some node w which obviously represents the subfunction $g = f_{|x_1=c_1,\ldots,x_{i-1}=c_{i-1}}$ of f. We conclude that at the nodes of each OBDD for f only such subfunctions are represented. The following lemma (Sieling and Wegener [123] – a similar lemma is given in Friedman and Supowit [55]) – describes which of these subfunctions are represented at the nodes of a reduced OBDD for f. We say that a function g essentially depends on x_i iff $g_{|x_i=0} \not\equiv g_{|x_i=1}$.

Lemma 1. *Let G be the reduced OBDD for the single function f and the variable ordering $x_1, \ldots, x_n$. Let S_i be the set of different subfunctions $f_{|x_1=c_1,\ldots,x_i=c_{i-1}}$ where $c_1, \ldots, c_{i-1} \in \{0,1\}$ and which essentially depend on x_i. For each function $g \in S_i$, the OBDD G contains exactly one node labeled by x_i which represents g. The OBDD G does not contain further internal nodes.*

Let us examine the reduction rules again (see Fig. 2). By the S-deletion rule only nodes v labeled by x_i are removed which represent functions not essentially depending on x_i. Furthermore, it is easy to see that v and its successor w represent the same function. In addition, for

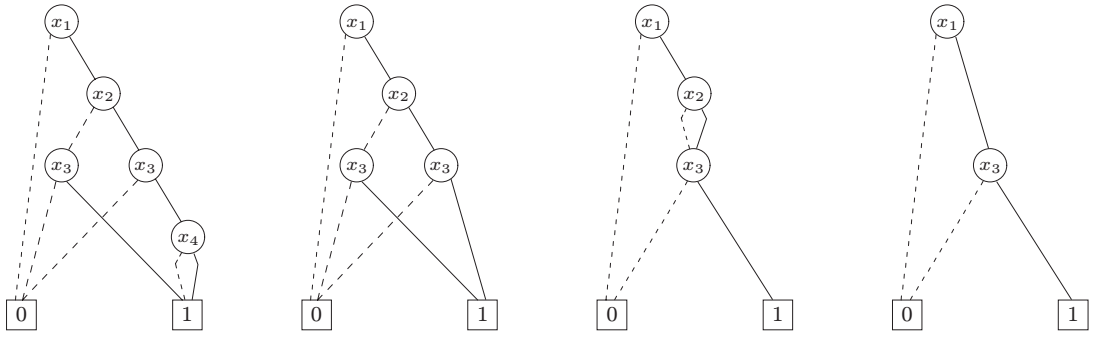

Fig. 3. An example for the reduction of an OBDD

the merging rule it is obvious that only nodes representing the same function are merged. Thus, the reduction rules make sure that for each function in S_i there is only one node representing this function. The canonicity of G can be proved by showing that there is only one possibility to combine the nodes for the functions described in lemma 1 to an OBDD for f.

Finally, we discuss the reduction of OBDDs with complemented edges. The merging rule can be adapted by requiring that v and w can only be merged if additionally the complement attributes on the outgoing edges of v and w coincide. Similarly, the deletion rule is applicable to nodes v for which both outgoing edges lead the same node if there are complement attributes on both outgoing edges or on neither outgoing edge. In order to ensure canonicity we have to restrict the use of complement attributes by allowing complement attributes only on 1-edges and on pointers for functions. Furthermore, there is no sink labeled by 1 since this sink can be simulated by a complemented edge to the sink labeled by 0. The number of complement attributes on each computation path does not change if we flip the complement attributes on all edges leading to some node v (including function pointers to v) and on all edges leaving v. It is clear that by applying this rule bottom-up we can make sure that complement attributes only occur on 1-edges. Examples of the application of this rule are shown in Fig. 4. In this figure the complement attributes are indicated by black dots on edges.

With the above restrictions on the complement attributes, OBDDs with complemented edges are also a canonical representation; this means, theorem 1 holds and there is also a slightly adapted version of lemma 1. We note that the choice to allow complemented edges only on the 1-edges and the choice to allow only the 0-sink are somewhat arbitrary; it is also possible to allow complement attributes on the 0-edges or to have only the 1-sink. Later on, we shall see that this symmetry does not hold for other variants of OBDDs.

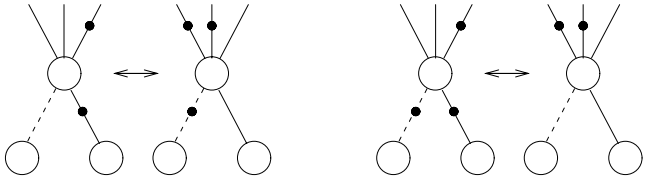

Fig. 4. Replacing complement attributes on the edges leading to and from some OBDD node

2.3 Algorithms on OBDDs

Many operations for the manipulation of Boolean functions can be performed efficiently for functions represented by OBDDs. The most important basic operations are the following ones:

1. Evaluation. For an OBDD G representing f and an input a compute the value $f(a)$.
2. Reduction. For an OBDD G compute the equivalent reduced OBDD.
3. Equivalence test. Test whether two functions represented by OBDDs are equal.
4. Satisfiability problems. These problems include:
   - Satisfiability. For an OBDD G representing f find an input a for which $f(a) = 1$ or output that no such input exists.
   - SAT-Count. For an OBDD G representing f compute the number of inputs a for which $f(a) = 1$.
5. Synthesis (also called Apply). For functions f and g represented by an OBDD G include into G a representation for $f \otimes g$ where $\otimes$ is a binary Boolean operation (e.g., $\wedge$).
6. Replacements (also called Substitution). There are two replacement operations:
   - Replacement by constants. For a function f represented by an OBDD, for a variable x_i and a constant $c \in \{0, 1\}$ compute an OBDD for $f_{|x_i=c}$.
   - Replacement by functions. For functions f and g represented by an OBDD and for a variable x_i compute an OBDD for $f_{|x_i=g}$.
7. Universal quantification and existential quantification. For a function f represented by an OBDD and for a variable x_i compute an OBDD for $(\forall x_i : f) := f_{|x_i=0} \wedge f_{|x_i=1}$ or $(\exists x_i : f) := f_{|x_i=0} \vee f_{|x_i=1}$, respectively.

We have already described algorithms for the operations evaluation and reduction. As we shall see in the following, in OBDD packages the operation reduction is usually integrated into the other operations such that only reduced OBDDs are represented. Before we present algorithms for the other operations we give a short motivation for these operations.

Many applications of OBDDs concern functions that are given as circuits. Hence, an important operation is the computation of an OBDD for a function given by a circuit, which is usually performed by the symbolic simulation of the circuit. This means the following. First, OBDDs for the functions representing the input variables are constructed. This is easy since an OBDD for the function x_i merely consists of a node labeled by x_i with the 0-sink as 0-successor and the 1-sink as 1-successor. Afterwards we run through the circuit in some topological order (each gate is considered after all its predecessors have been considered) and compute for each gate a representation of the function at its output by combining the OBDDs representing the functions at its input by the synthesis operation.

Another possibility is that a given circuit is built of larger blocks. Then we may compute OBDDs for the functions computed by each block and combine these OBDDs with the operation replacement by functions.

If we have computed OBDDs for the functions represented by two circuits we may apply the equivalence operation for OBDDs in order to test the circuits for equivalence.

The operation quantification is important for the application model checking (see, for example, Burch, Clarke, McMillan, Dill, and Hwang [30]).

In applications such as Boolean matching, signatures for the considered functions are computed. Roughly, a signature is a property of a function that can be computed efficiently and that is likely to be different for different functions. Hence, signatures can be used to detect that given functions are different. A very simple signature is the number of satisfying inputs of a function. Hence, we may apply the operation SAT-count in order to compute this (and also other) signatures (see, for example, Mohnke, Molitor, and Malik [109]).

In order to simplify the presentation we assume in the following that the variable ordering of the considered OBDDs is $x_1, \ldots, x_n$. We point out that in many applications it is a serious problem to select a suitable variable ordering for the OBDDs. In Sect. 2.4 we shall see how much the OBDD size may depend on the variable ordering. There we shall also discuss strategies to choose a suitable variable ordering. We note that some operations can be performed more efficiently if the considered OBDDs are reduced. Hence, in the following we sometimes distinguish between operations on reduced and on non-reduced OBDDs, and discuss for which operations the result is reduced.

2.3.1 Equivalence and satisfiability

We start with the equivalence test. In BDD-packages different functions are usually represented in the same reduced OBDD. Then the equivalence test of two functions f and g is easy. As mentioned above, it suffices to check whether the pointers for f and g lead to the same node, which can be done in constant time.

Another possibility is that f and g are represented in different reduced OBDDs with the same variable ordering. Let v_f and v_g be the nodes that are the heads of the pointers for f and g. By theorem 1 the OBDDs consisting of the nodes reachable from v_f and v_g are identical iff $f \equiv g$. Hence, it suffices to test these OBDDs for isomorphy, which can be done by a simultaneous depth-first search traversal through the OBDDs starting at v_f and v_g. The run time of this algorithm is $O(\min\{|G_f|, |G_g|\})$ where G_f denotes the OBDD for f, which consists of the nodes reachable from v_f, and G_g is defined similarly.

There is also a third version of the equivalence test, namely, for the case that the OBDDs for f and g have a different variable ordering. The approaches of Sect. 2.4 for solving the variable ordering problem imply that this situation may occur. Then the equivalence test is much more complicated. A polynomial time algorithm for this problem was first presented by Fortune, Hopcroft, and Schmidt [54]. Later on, Meinel and Slobodová [99] presented an algorithm for this problem and estimated its run time by $O(|G_f||G_g|)$.

Now we consider the satisfiability problem. In order to find an input a for which $f(a) = 1$, we start at the pointer for f and search for a path to the 1-sink. For this, a simple depth-first search approach is sufficient. We choose a such that this path is run through. If there is no such path, we conclude that $f \equiv 0$. If the OBDD is reduced, the run time of this algorithm is $O(n)$ since it suffices to consider a single path. We point out that for this algorithm it is essential that on each path in the OBDD each variable occurs at most once (property 4 of definition 1). Thus, there are no inconsistent paths, i.e., paths on which some x_i-node is left via the 0-edge and some other x_i-node via the 1-edge. If there were such a path, it would not be possible to define a as outlined before. There are extensions of OBDDs in which inconsistent paths may occur. For most such variants the satisfiability test is NP-complete and the equivalence test is coNP-complete (see, for example, Bollig, Sauerhoff, Sieling, and Wegener [19]). Hence, if we relax the read-once property (property 4 of definition 1) we do not only lose the canonicity of the representation but also the efficient algorithms for satisfiability and equivalence.

An operation closely related to satisfiability is the operation SAT-count. Let f be the function represented at some node v_f. We consider the 2^n computation paths starting at v_f which are chosen for the 2^n inputs. Note that for different inputs the computation paths need not to be different. We want to count the number of these paths that lead to the 1-sink. Let w be a node of the OBDD. The following fact is easy to see: If $N(w)$ paths run through w then these paths equally distribute among the 0-edge and the 1-edge leaving w. This directly leads to the following algorithm for SAT-count. We start with $N(v_f) = 2^n$ and $N(w) = 0$ for all nodes $w \neq v_f$. Then we run levelwise top-down through the OBDD. For each node w with the successors w_0 and w_1 we perform the assignments $N(w_0) := N(w_0) + N(w)/2$ and $N(w_1) := N(w_1) + N(w)/2$. Hence, we distribute the paths through w among the outgoing edges. Then the number of satisfying inputs is $N(1\text{-sink})$. The number of performed operations in $O(|G|)$.

2.3.2 Synthesis

Synthesis is probably the most important operation since it is used to construct OBDDs from circuits as outlined above and, hence, it is needed in almost all applications. In order to simplify the presentation we first present the idea of the synthesis algorithm and discuss implementation issues later on.

Let v_f and v_g be the nodes of an OBDD G which represent the functions f and g. The goal is to construct an OBDD for $f \otimes g$ where $\otimes$ is a binary Boolean operator, e.g., $\wedge$. We consider how to compute $(f \otimes g)(a)$ for some

input a. One may think of the evaluation of $(f \otimes g)(a)$ as a simultaneous traversal of the computation paths for a starting at v_f and v_g. In this simultaneous traversal we want to test the variables according to the variable ordering of G. We start at the nodes v_f and v_g. If v_f and v_g are labeled by the same variable x_i, we follow both computation paths through the a_i-edges leaving v_f and v_g, respectively. However, it may also happen that, for example, v_f is labeled by x_i and that v_g is a sink or that it is labeled by x_j where $i < j$. If we want to perform a simultaneous traversal, we have to wait at v_g while we follow the a_i-edge leaving v_f. Similarly, we may have to wait at v_f. In the same way we may perform all subsequent tests of variables until we reach the sinks on both computation paths. Then it is easy to compute the value of $(f \otimes g)(a)$ by combining the labels of the sinks by $\otimes$.

Our goal is to construct an OBDD G that "simulates" this simultaneous traversal. The situation that a node u is reached on the computation path for $f(a)$ and a node w on the computation path for $g(a)$ is simulated by reaching a node (u, w) in G. Hence, the node set for the representation of $f \otimes g$ is some subset of $V_f \times V_g$ where V_f is the set of nodes reachable from v_f, and V_g is defined similarly. We describe how to obtain the c-successor of (u, w). Let u_0 and u_1 be the 0- and 1-successor of u and let w_0 and w_1 be defined similarly. We distinguish the following cases:

Case 1: u and w are sinks with labels c_u and c_w. Then (u, w) is a sink with label $c_u \otimes c_w$.

Case 2: u and w are internal nodes and both are labeled by x_i. Then (u, w) is an internal node labeled by x_i and with the c-successors (u_c, w_c).

Case 3: u is an internal node with label x_i and w is a sink or an internal node with label x_j where $j > i$. Then (u, w) is an internal node with label x_i and the c-successors (u_c, w). This models the situation that we have to wait on the computation path for g during the simultaneous traversal.

Case 4: w is an internal node with label x_i and u is a sink or an internal node with label x_j where $j > i$. This situation is handled similarly to Case 3.

An example of two OBDDs G_f and G_g and the constructed OBDD G for $f \wedge g$ is shown in Fig. 5. It is easy to see that the node set of G is much smaller than $V_f \times V_g$. Hence, it is essential to construct only those nodes of G which are really needed. This is done by constructing G in a depth-first search manner starting at the node (v_f, v_g) where v_f and v_g are the nodes where the computations of f and g start.

In order to make sure that no part of G is constructed twice in the depth-first search construction, for each pair (u, w) it is stored which node is constructed for this pair. Usually, this is done using a hash table, which is called the *computed table*. Before we create a node for (u, w) we check using the computed table whether a node for (u, w) was already created. In this case we do not need to create this node for a second time.

In the example of Fig. 5 more than one 0-sink is created. In particular, the constructed OBDD is not reduced. In order to keep the number of created nodes as small as possible, usually the reduction is integrated into the synthesis algorithm; this means it is ensured that no two nodes represent the same function and that at each x_k-node a function essentially depending on x_k is represented. Usually this is done using a hash table, which is called the *unique table*. In the unique table for each triple (x_k, R_0, R_1) is stored whether there is a node in the OBDD which is labeled by x_k, has the 0-successor R_0, and the 1-successor R_1. In the positive case there is a pointer to this node. Before we create a node labeled by x_k and with the 0-successor R_0 and the 1-successor R_1, we check using the unique table whether such a node was already constructed. In this case, we may set a pointer to this node instead of creating a new node. Similarly, before creating the node we check whether R_0 and R_1 coincide. In this case, by the S-deletion rule it suffices to set a pointer to R_0 instead of creating a node labeled by x_k.

The integration of the reduction into the synthesis is also important because we want to represent $f \otimes g$ in the same OBDD as f and g. This can be done by storing the unique table for the whole OBDD and using the unique table as described in the last paragraph. Then the OBDD for $f \otimes g$ shares as many as possible nodes with the given OBDD for f and g.

A generic algorithm for the synthesis is given in Figure 6. The check for terminal cases saves the construction of nodes representing a constant function. For example, if we consider the pair of $u \in V_f$ and the 0-sink $w \in V_g$ during the construction of an OBDD for $f \wedge g$, we know that at (u, w) the constant function 0 is represented and

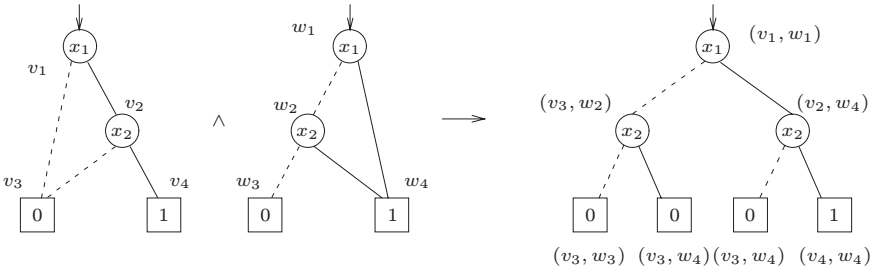

Fig. 5. An example of two OBDDs G_f and G_g and the constructed OBDD for $f \wedge g$

Input: A reduced OBDD G with its unique table that represents f and g at the nodes u and w, a binary operation $\otimes$.
Output: The reduced OBDD G still representing f and g, a pointer to the node representing $f \otimes g$, and the updated unique table for G.

Synthesis(u, w)
if (terminal case) **then return** result.
if (computed table entry(u, w) exists) **then return** result.
Let x_k be the top variable of u and w.
if u is labeled by x_k
  then let $\hat{u}_0$ be the 0-successor of u
  else let $\hat{u}_0 := u$
Let $\hat{u}_1$, $\hat{w}_0$ and $\hat{w}_1$ be defined similarly
$R_0 =$ Synthesis$(\hat{u}_0, \hat{w}_0)$
$R_1 =$ Synthesis$(\hat{u}_1, \hat{w}_1)$
if $R_0 = R_1$ **then return** R_0 /* apply the S-deletion rule */
if (unique table entry R for (x_k,R_0,R_1) does not exist)
  then /* the merging rule is not applicable */
    create node R with label x_k, the 0-successor R_0
    and the 1-successor R_1
    insert ((x_k,R_0,R_1),R) into unique table
insert ((u, w), R) into computed table
return R

Fig. 6. An algorithm for the synthesis of OBDDs

there is no need to construct successors of (u, w) as described above. Similarly, if we have to construct a node for the pair (u, u) and, for example, the operation $\wedge$, it suffices to return a pointer to u, which is already contained in the OBDD. The ideas of all other instructions in the algorithm in Fig. 6 have already been explained.

Since the set of nodes constructed for the representation of $f \otimes g$ is some subset of $V_f \times V_g$, the number of constructed nodes is at most quadratic in the size of the input. Since the algorithm uses hash tables, the expected time for the construction of each node is constant. Hence, the expected run time is $O(|V_f||V_g|)$. The worst case run time is only bounded by $O(|V_f|^2|V_g|^2)$ since hash tables are very slow in the worst case which, however, is quite unlikely to occur.

We remark that it is straightforward to generalize the synthesis algorithm to combine three functions f, g, h by a ternary operator $op : \{0,1\}^3 \to \{0,1\}$; this means computing a representation for $op(f, g, h)$. As suggested by Brace, Rudell, and Bryant [21], in OBDD packages usually the synthesis algorithm for the ternary operation *ite* ("if-then-else") is implemented where *ite* is defined by

$$ite(f, g, h) = fg \vee \overline{f}h.$$

All binary Boolean operations can be simulated by the *ite*-operator, e.g., $f \vee g = ite(f, 1, g)$, $f \wedge g = ite(f, g, 0)$ or $f \oplus g = ite(f, \overline{g}, g)$. The motivation for preferring synthesis with the operation *ite* is that this increases the hit-rate of the computed table. For more details and further implementation issues we refer to Somenzi [126].

Due to the importance of the synthesis operation, alternative implementations of the synthesis operation have also been suggested. Ochi, Ishiura, and Yajima [111] and Ochi, Yasuoka, and Yajima [112] use a breadth-first search approach instead of the depth-first search approach which we described above. Such an approach is more suitable for vector processing. Another advantage is the reduction of the number of page faults when working on large OBDDs. Hett, Drechsler, and Becker [70] use a totally different implementation of the synthesis operation. We only remark that they introduce OBDD nodes for Boolean operations and perform the synthesis by moving these nodes in the considered OBDDs. This approach also allows us to reorder OBDDs during synthesis. A combination of these alternative approaches was presented by Yang, Chen, Bryant, and O'Hallaron [138]. Höreth [71] suggests speeding up successive applications of the synthesis operation by also implementing synthesis for more complex operations than *ite*.

As already discussed for the equivalence test, we may consider OBDDs for different variable orderings. If f and g are given by OBDDs with different variable orderings, the above synthesis algorithm does not work since for the simultaneous traversal the variables have to be encountered in the same ordering in both OBDDs. We note that in this situation the synthesis problem is even harder: Fortune, Hopcroft, and Schmidt [54] proved that it is NP-hard to determine whether $f \wedge g = 0$. Hence, it is also NP-hard to determine whether $f \wedge g$ can be represented by an OBDD only consisting of the 0-sink.

2.3.3 Replacements and quantification

It is obvious how to construct an OBDD for $f_{|x_i=c}$ from an OBDD for f. It suffices to redirect all edges leading to nodes labeled by x_i to the c-successor of these nodes. Afterwards, the OBDD is not necessarily reduced such that we may apply a reduction algorithm to the result. If we consider an OBDD representing several functions this algorithm destroys the representation of the other functions. Thus, we have to create a copy of the representation of f before performing the replacement as described before.

Now the operation replacement by a function can be solved easily by combining the algorithms for replacement by a constant and synthesis for ternary operators because

$$f_{|x_i=g} = ite(g, f_{|x_i=1}, f_{|x_i=0}).$$

Similarly it follows from the definition of the quantification operations that they can be implemented by combining replacement by constants and synthesis.

2.4 The variable ordering problem for OBDDs

Bryant [23] already discovered the dependence of the OBDD size on the chosen variable ordering. In order to show this he presented the following example. Let n be an even number. Then $f_n : \{0,1\}^n \to \{0,1\}$ is defined by

$$f_n(x_1, \ldots, x_n) = x_1x_2 \vee x_3x_4 \vee \ldots \vee x_{n-1}x_n.$$

Figure 7 shows the OBDDs for f_6 for the variable orderings $x_1, x_2, \ldots, x_6$ and $x_1, x_3, x_5, x_2, x_4, x_6$. For arbitrary n it can be shown using lemma 1 that the number of internal nodes for f_n and the variable ordering $x_1, x_2, \ldots, x_n$ is n while the number of internal nodes for the variable ordering $x_1, x_3, \ldots, x_{n-1}, x_2, x_4, \ldots, x_n$ is $2^{n/2+1} - 2$. The reason for the exponential size is that for this variable ordering the OBDD has to "store" the values of the variables with odd indices in the $(n/2+1)$th level. We conclude that the choice of the variable ordering determines whether the number of nodes is linear or exponential. There are more examples with this property, see, for example, Wegener [135]. The examples make clear that the variable ordering must be chosen carefully.

Wegener [135] proves an even stronger result on the function f_n. He shows that the fraction of variable orderings π that lead to a polynomial number of nodes in an OBDD for f and π with respect to all variable orderings is exponentially small. The same holds, for example, for the function describing the most significant bit of the sum of two n-bit numbers. This result coincides with the observation that many practically important functions have only few variable orderings leading to small OBDD size. We conclude that variable orderings should not be chosen randomly.

In applications, whether a computation on OBDDs can be completed or fails because of lack of memory may depend on the chosen variable ordering. Thus, computing a good variable ordering is an important problem. Friedman and Supowit [55] present an algorithm for the computation of a variable ordering leading to minimum OBDD size. This algorithm is based on a dynamic programming approach. However, the run time is exponential and the algorithm can only be applied to functions with a small number of inputs. The algorithm was improved by Ishiura, Sawada, and Yajima [77], Jeong, Kim, and Somenzi [79], and by Drechsler, Drechsler, and Günther [47]. However, if the number of variables exceeds 30, even with all these improvements, the optimum variable ordering cannot be computed unless special properties of the considered function are exploited.

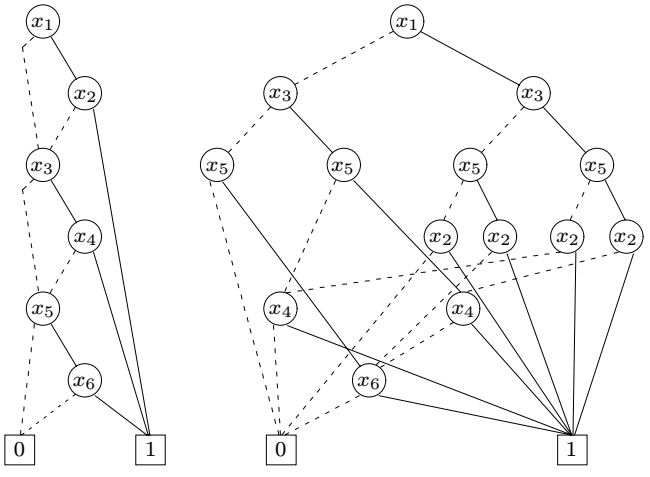

Fig. 7. (Bryant [23]) OBDDs for $f_6 = x_1x_2 \vee x_3x_4 \vee x_5x_6$ and the variable orderings $x_1, x_2, \ldots, x_6$ and $x_1, x_3, x_5, x_2, x_4, x_6$

In many other papers heuristics for the variable ordering problem are presented, which we discuss shortly in the following. We may distinguish two different situations where variable orderings have to be obtained. In the first one, the function to be represented is given by a circuit. Then the structure of the circuit might help to obtain dependencies between the variables which we can use to get a variable ordering. A rule of thumb to construct a variable ordering is "variables belonging together in some sense should be arranged together in the variable ordering." This rule is applied, for example, in the heuristic algorithm of Fujita, Fujisawa, and Kawato [57]. Their algorithm runs through the given circuit starting at the outputs and using a depth-first search approach. The variables are arranged in the ordering in which they are found by the depth-first search traversal. Hence, variables occurring in the same part of the circuit are likely to be arranged closely together in the variable ordering. Other papers presenting heuristics to obtain variable orderings from circuits are, for example, Malik, Wang, Brayton, and Sangiovanni-Vincentelli [97], Butler, Ross, Kapur, and Mercer [31], Fujita, Matsunaga, and Kakuda [58], Jeong, Plessier, Hachtel and Somenzi [80], Ross, Butler, Kapur, and Mercer [114], Touati, Savoj, and Lin [132], Mercer, Kapur, and Ross [103], and Fujii, Ootomo, and Hori [56].

In the second situation we already have an OBDD G for the functions to be represented and want to minimize the number of nodes by choosing the variable ordering. Fujita, Matsunaga, and Kakuda [58] use an algorithm for this problem in the following case. For the transformation of a circuit into an OBDD an initial variable ordering is chosen, then the OBDD is constructed and, finally, this OBDD is minimized by improving the variable ordering. Rudell [115] suggests improving the variable ordering also during the transformation of the circuit into the OBDD. This approach is called dynamic reordering. It is motivated by the fact that during the transformation the set of represented functions changes and that it is unlikely that the same variable ordering is suitable for all these sets of functions. Dynamic reordering can also be used in other applications where long sequences of computations are performed on OBDDs, i.e., for model checking. Bryant [25] reports that this dynamic reordering slows down the computations but it can make the difference between success and failure in completing an application. This leads to the following problems:

1. MinOBDD. For an OBDD G for $f^1, \ldots, f^m$ with the variable ordering π compute a variable ordering minimizing the OBDD size for $f^1, \ldots, f^m$.
2. Reordering of OBDDs. For a variable ordering π and an OBDD G with the variable ordering π' compute an OBDD with the variable ordering π that represents the same functions as G.

For the problem MinOBDD several heuristics have been suggested. Most of them are based on local search approaches. The main observation is that it is easy to implement algorithms for the following local changes of the variable ordering:

1. Swap two variables which are adjacent in the variable ordering. Figure 8 shows that this operation essentially consists of redirecting pointers. We only remark that swapping of two adjacent variables can be done in a similar way for OBDDs with complemented edges.
2. Let the variable x_i jump at the jth position in the variable ordering without changing the relative ordering of the other variables. The implementation of this operation using the synthesis algorithm is described by Bollig, Löbbing, and Wegener [18].

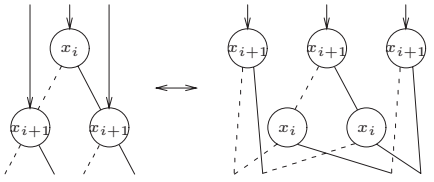

Fig. 8. Swapping two adjacent variables

Local search and simulated annealing approaches based on these operations are presented, for example, in Mercer, Kapur, and Ross [103], Fujita, Matsunaga, and Kakuda [58], Ishiura, Sawada, and Yajima [77], and Bollig, Löbbing, and Wegener [17]. One of the most successful algorithms is the Rudell's sifting algorithm [115]. Roughly, the sifting algorithm selects a variable and searches for the position of this variables which minimizes the OBDD size. This process is iterated until it does not lead to further improvements. The reason for the success of the sifting algorithm is that it provides a good trade-off between the run time and the quality of its results. In [48] Drechsler and Günther present an approach to speed up the sifting algorithm by lower bound computations. The use of genetic algorithms for the computation of good variable orderings is suggested by Drechsler, Becker, and Göckel [43].

The fact that no efficient algorithm to solve the problem MinOBDD exactly on all instances is known leads to the question of whether there are any such algorithms at all. The following theorem implies that such algorithms are quite unlikely to exist.

Theorem 2. *MinOBDD is NP-hard.*

The theorem was proved for OBDDs representing a large number of functions by Tani, Hamaguchi, and Yajima [129], and for OBDDs representing single functions by Bollig and Wegener [20]. However, the NP-hardness results do not exclude the existence of efficient algorithms for approximately solving the problem MinOBDD. For example, an algorithm that for each OBDD efficiently computes a variable ordering such that the OBDD size is at most 10% larger than the minimum size, would be helpful in practice and is not excluded by theorem 2. Such algorithms are known for many other combinatorial optimization problems. For example, for the Knapsack Problem, the polynomial time algorithm of Ibarra and Kim [76] guarantees a solution that is larger than the optimum size by a factor of at most $1+\varepsilon$ where each constant $\varepsilon > 0$ can be chosen. However, it is unlikely that an approximate solution for the problem MinOBDD can be computed by a polynomial time algorithm. For a function f and for $\alpha \geq 1$ we call a variable ordering π an α-approximation, if the OBDD-size for f and π is larger than the minimum OBDD size for f by a factor of at most α. The following theorem (Sieling [119]) implies that it is hard to approximate the problem MinOBDD.

Theorem 3. *For each constant $\alpha \geq 1$ the computation of α-approximations is NP-hard.*

Hence, heuristics are the only possibility to obtain good variable orderings.

Now let us consider the reordering of OBDDs. From the example of the function f_n defined at the beginning of this section, it follows that the output size for the reordering problem of OBDDs may be exponential in the input size. Hence, we can only hope for algorithms which are efficient with respect to the size of the input G *and* the output H. Such algorithms are presented by Meinel and Slobodová [99], Tani and Imai [130], and Savický and Wegener [116]. The most efficient algorithm is presented in the latter paper and has a run time of $O(|G||H| \log |H|)$.

3 Generalizations of OBDDs

In the last section we have seen efficient algorithms for the manipulation of functions represented by OBDDs. However, there are many functions which occur in applications and for which OBDDs are much too large to be stored in a computer memory. For example, Bryant [23, 24] proved that OBDDs for the multiplication of two n-bit numbers have at least $2^{n/8}$ nodes. Experiments show that OBDDs for multipliers are very large even for small values of n. For example Yang et al. [138] constructed an OBDD for a 16-bit multiplier with more than 40 million nodes. There are other functions with the same property. This is the reason why quite a large number of extensions of the basic OBDD concept have been suggested. Most of these extensions allow a more succinct representation

of functions than OBDDs. On the other hand, these extensions have the disadvantage that for some of the basic operations listed in Sect. 2 only less efficient algorithms are known. In the following list we give a rough classification of the extensions of OBDDs and examples of such extensions:

1. Extensions obtained by replacing Shannon's decomposition rule by a different rule. Examples for such extensions are ordered functional decision diagrams (OFDDs), ordered Kronecker functional decision diagrams (OKFDDs) and parity OBDDs.
2. Extensions obtained by relaxing the variable ordering condition (5) of definition 1. Then we obtain free BDDs.
3. Extensions obtained by relaxing the read-once condition (4) in definition 1. Examples are kOBDDs and IBDDs.
4. Extensions obtained by introducing a transformation τ and representing a function g such that $f = g \circ \tau$. This variant is called transformed BDD (TBDD).
5. Extensions that represent functions $f : \{0,1\}^n \to \mathbf{Z}$ instead of Boolean functions. In order to represent such functions, we may extend OBDDs and OFDDs by allowing sinks labeled by numbers from $\mathbf{Z}$ or by introducing edge labels. Examples of the former variant are multi-terminal BDDs (MTBDDs), which are also called algebraic decision diagrams (ADDs), and binary moment diagrams (BMDs). Examples for the latter variant are edge-valued BDDs (EVBDDs), multiplicative binary moment diagrams (*BMDs), hybrid decision diagrams (HDDs), Kronecker multiplicative BMDs (K*BMDs), and multiplicative power hybrid decision diagrams (*PHDDs).

This list is not complete at all. Due to the large number of extensions of OBDDs we can only discuss some properties of these extensions in the following sections. We remark that another extension of OBDDs, namely zero-suppressed BDDs (ZBDDs), are considered in more detail in the article of Minato [107] in this special section. BMDs are also considered in the article of Bryant and Chen [28].

3.1 Decision diagrams based on different decomposition rules

Ordered functional decision diagrams (OFDDs) were proposed by Kebschull, Schubert, and Rosenstiel [81, 82]. They are syntactically defined as OBDDs (see definition 1) but the Shannon decomposition is replaced by the (positive) Davio decomposition

$$f = f_{|x_i=0} \oplus x_i (f_{|x_i=0} \oplus f_{|x_i=1}). \tag{3}$$

Let w be an x_i-node of an OFDD with the 0-successor w_0 and the 1-successor w_1. Then at w_0 the function $f_{w_0} = f_{w|x_i=0}$ and at w_1 the function $f_{w_1} = f_{w|x_i=0} \oplus f_{w|x_i=1}$ is represented.

The function represented by an OFDD can be evaluated by a bottom-up traversal. The c-sink represents the constant function c. At each internal node w the value $f_w(a)$ is computed by $f_w(a) = f_{w_0}(a) \oplus x_i f_{w_1}(a)$.

It is also possible to describe the represented function using computation paths. Different from OBDDs for each input a there may be several computation paths. Let v be a node representing some function f. Each computation path for $a = (a_1, \ldots, a_n)$ and f starts at v. At an internal node labeled by x_i each computation path has to follow the 0-edge, if $a_i = 0$. Each computation path may follow the 0-edge *or* the 1-edge, if $a_i = 1$. Then $f(a) = 1$ iff the number of computation paths from v to the 1-sink is odd.

An example of an OFDD for $f(x_1, x_2, x_3) = x_1 \oplus x_2 \oplus x_3$ is shown in Fig. 9. The computation paths for the input $(1, 0, 1)$ are indicated by dotted lines. Since the number of such paths leading to the 1-sink is even, the function takes the value 0 for this input.

We consider the reduction rules for OFDDs. The merging rule for OBDDs can also be applied to OFDDs, while we need a different deletion rule which we call pD-deletion rule. This has the following reason. For OBDDs the deletion rule is applicable to nodes v for which both outgoing edges lead to the same node w. However, for OFDDs the nodes v and w do not represent the same function. The pD-deletion rule is applicable to nodes v whose 1-successor is the 0-sink (Drechsler, Theobald, and Becker [52]). Let v_0 be the 0-successor of v. By the positive Davio decomposition rule $f_v = f_{v_0}$. Hence, we can delete v and redirect all edges leading to v to v_0. The reduction algorithm for OBDDs can easily be modified to work on OFDDs by replacing the deletion rule. It can also be shown that reduced OFDDs are a canonical representation.

Since OFDDs are syntactically equivalent to OBDDs one may ask for the relation between the function f_{OBDD} which we obtain by considering a decision diagram G as an OBDD and the function f_{OFDD} which we obtain by considering G as an OFDD. Since OBDDs and OFDDs have different deletion rules this question can only be answered for the case that neither of the deletion rules has been applied. Hence, we consider *com-*

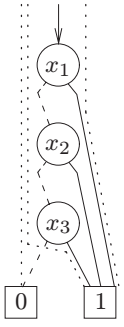

Fig. 9. An OFDD for the function $f(x_1, x_2, x_3) = x_1 \oplus x_2 \oplus x_3$ where the computation paths for the input $(1, 0, 1)$ are indicated by dotted lines

plete decision diagrams, i.e., decision diagrams where on each path from a function pointer to a sink all variables are tested. Becker, Drechsler, and Werchner [12] observed that in the case of a complete decision diagram G it holds that $f_{OBDD} = \tau(f_{OFDD})$ and $f_{OFDD} = \tau(f_{OBDD})$, where $\tau(f(a)) = \bigoplus_{x \leq a} f(x)$. Here, $\leq$ denotes the componentwise comparison, i.e., $x \leq a$ iff $\forall i : x_i \leq a_i$. The relation $f_{OFDD} = \tau(f_{OBDD})$ can be used to obtain size bounds for OFDDs for some function f by considering OBDDs for $\tau(f)$. Becker, Drechsler, and Werchner [12] present functions for which OBDDs are exponentially larger than OFDDs and vice versa.

Becker, Drechsler, and Theobald [10] implemented an OFDD-package in which OFDDs with complemented edges are used. As for OBDDs the use of complemented edges may halve the size of the representation and allows the negation of a represented function in constant time. In order to obtain a canonical representation we allow complement attributes only on 1-edges and on function pointers. Furthermore, there is only a 0-sink and edges to the 1-sink are replaced by complemented edges to the 0-sink. In Sect. 2.2 we discussed how to obtain the canonical form of OBDDs with complemented edges. Here we need a slightly different rule to modify the complemented edges at a node. Let v be a node labeled by x_i and representing a function f. If we flip the complement attributes on all edges leading to v, the node has to represent $\overline{f}$ afterwards. Let g and h be the functions represented at the 0- and 1-successor of v, respectively. Then $f = g \oplus x_i h$ and $\overline{f} = \overline{g} \oplus x_i h$. We conclude that we have to flip the complement attribute on the 0-edge leaving v while the complement attribute on the 1-edge remains unchanged. This rule is depicted in Fig. 10. By this rule it is possible to remove all complement attributes on 0-edges but it is not possible to remove complement attributes on 1-edges. Hence, unlike OBDDs, we do not have the choice to allow complement attributes on 0-edges instead of 1-edges if we want to obtain a canonical representation.

Algorithms for the manipulation of OFDDs are presented by Becker, Drechsler, and Werchner [12] and Werchner, Harich, Drechsler, and Becker [136]. We only consider the main differences to OBDDs. For the binary operation $\oplus$ there is a synthesis algorithm similar to that for OBDDs. However, for the operation $\wedge$, synthesis may cause an exponential blow-up (Becker, Drechsler, and Werchner [12]). However, it can be tested in polynomial time whether the result of an $\wedge$-synthesis is different from the constant function 0 (Drechsler, Sauerhoff, and Sieling [51]). This can be shown by exploiting the relation between OFDDs and $\oplus$-OBDDs, which we describe below. The replacement of a variable x_i by the constant 0 can be done as for OBDDs by redirecting the edges to x_i-nodes to their 0-successors. The replacement of x_i by 1 is more complicated since an edge leading to an x_i-node v has to be redirected to a node representing $f_{v|x_i=1}$ which need not be in the OFDD. Then, such a node has to be computed which can be done by applying the synthesis for the operation $\oplus$ to the successors of v. Hence, the replacement of a variable by the constant 1 may square the size of the OFDD. An example where a logarithmic number of replacements of variables by the constant 1 causes an exponential blow-up was presented by Bollig, Löbbing, Sauerhoff, and Wegener [16].

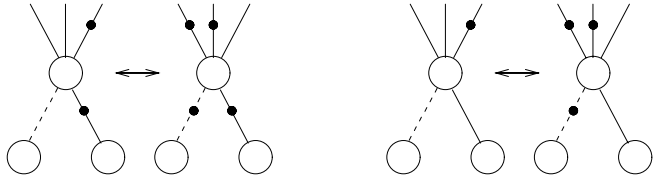

Fig. 10. Moving complement attributes in OFDDs

Since OFDDs are a canonical representation, the test of whether a function is satisfiable and the computation of a satisfying input can be done efficiently. However, the operation SAT-count is NP-hard (Werchner, Harich, Drechsler, and Becker [136]). The variable ordering influences the size of OFDDs similarly to OBDDs. The problem to compute an optimal variable ordering for a given OFDD was shown to be NP-hard by Bollig, Löbbing, Sauerhoff, and Wegener [16].

Besides Shannon's decomposition rule and the positive Davio decomposition rule, there is a third decomposition rule, namely the negative Davio decomposition rule $f = f_{|x_i=1} \oplus \overline{x}_i(f_{|x_i=0} \oplus f_{|x_i=1})$. There is also a variant of OFDDs based on the negative Davio decomposition. Drechsler, Sarabi, Theobald, Becker, and Perkowski [50] combined OBDDs, positive Davio OFDDs, and negative Davio OFDDs to a new variant called ordered Kronecker functional decision diagrams (OKFDDs). The main idea is that at different nodes different decompositions may be chosen. However, in order to obtain a canonical representation, for each variable x_i the decomposition chosen at all x_i-nodes needs to be the same. Hence, in OKFDDs, there is a decomposition type list which describes for each variable the decomposition rule applied at nodes labeled by this variable. Obviously, each OBDD, each positive Davio OFDD, and each negative Davio OFDD can be transformed into an OKFDD without increasing the size. Hence, for each function, the size of the best OKFDD is not larger than the minimum of the sizes of OBDDs, positive Davio OFDDs, and negative Davio OFDDs for this function. Drechsler, Becker, and Jahnke [44] presented heuristics for the choice of the decomposition type list. Becker, Drechsler, and Theobald [11] adapted the definition of the τ-operator to describe the relation between the functions that we obtain when considering a decision diagram as an OBDD or as an OKFDD, respectively. They also showed that when considering canonical representations with complemented edges there is no further decomposition type besides Shannon, positive Davio, and negative Davio that can reduce the size of ordered DDs.

A representation that combines OBDDs, OFDDs, and OKFDDs as well and that avoids the exponential blow-up for the $\wedge$-synthesis and the replacement of variables by 1 are parity OBDDs ($\oplus$-OBDDs) which were introduced by Gergov and Meinel [62] and by Waack [133]. In the following definition we use the notation of Waack [133].

Definition 3. A $\oplus$-OBDD G representing the Boolean functions $f^1, \ldots, f^m$ over the variables $x_1, \ldots, x_n$ is a directed acyclic graph with the following properties:

1. For each function f^j there is a *set of pointers* to nodes in G.
2. There is one node without outgoing edges, called the 1-sink.
3. All internal nodes of G are labeled by a variable x_i, and have two (possibly empty) *sets of successors*, a set of 0-successors and a set of 1-successors.
4. On each directed path in G each variable occurs at most once as the label of a node.
5. On each directed path in G the variables are tested according to a fixed variable ordering.

For each input a and each node v there is a set of computation paths which can be obtained by starting at v and choosing the a_i-edges leaving each x_i-node. The function f_v represented at the node v takes the value 1 on the input a iff the number of computation paths for a from v to the 1-sink is odd.

Figure 11 shows examples of $\oplus$-OBDDs for the function $(x_1 \wedge x_2) \oplus x_3$. Similar to OBDDs, the relation between the functions represented at some node v and the function represented at its successors can be described; however, now v may have several 0-successors $u_1, \ldots, u_r$ and several 1-successors $w_1, \ldots, w_s$. Let v be labeled by x_i. It is not hard to show that

$$f_v = \bar{x}_i(f_{u_1} \oplus \cdots \oplus f_{u_r}) \vee x_i(f_{w_1} \oplus \cdots \oplus f_{w_s}).$$

In the case $r = s = 1$ we obtain Shannon's decomposition rule. Hence, a node of a $\oplus$-OBDD can simulate a node that decomposes the represented function by Shannon's decomposition. Figure 12 shows that a node of a $\oplus$-OBDD can also simulate a node for the positive and negative Davio decomposition. Hence, OKFDDs can be transformed into $\oplus$-OBDDs without increasing the number of nodes.

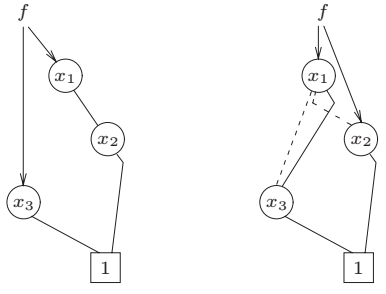

Fig. 11. Examples of $\oplus$-OBDDs for the function $(x_1 \wedge x_2) \oplus x_3$

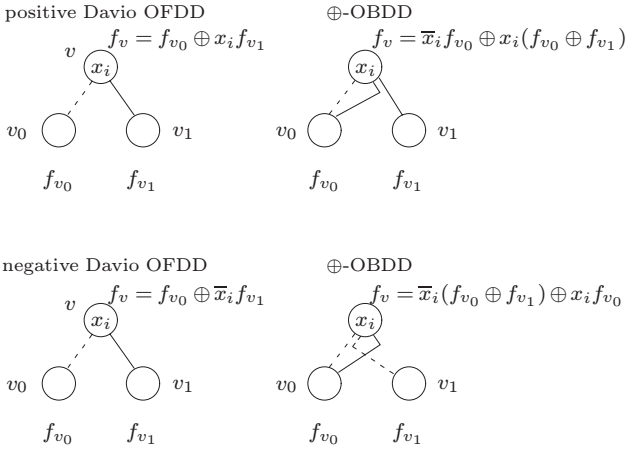

Fig. 12. The simulation of nodes for positive and negative Davio decomposition by nodes of $\oplus$-OBDDs

Waack [133] applied methods from linear algebra to show that the minimal number of x_i-nodes of a $\oplus$-OBDD only depends on the variable ordering and the represented function, i.e., it is not possible to reduce, for example, the number of x_i-nodes by increasing the number of x_j-nodes. Although $\oplus$-OBDDs with a minimum number of nodes are not obtained by reduction rules, they are called reduced $\oplus$-OBDDs. However, reduced $\oplus$-OBDDs are not a canonical representation. There may be different reduced $\oplus$-OBDDs for the same function and the same variable ordering (see Fig. 11). This is similar to the fact in linear algebra that a basis of a vector space is not uniquely determined though different bases have the same size. We mention that Král [83] obtained a canonical variant of $\oplus$-OBDDs by imposing further restrictions on $\oplus$-OBDDs. Waack [133] also presented a reduction algorithm based on Gaussian elimination for $\oplus$-OBDDs. Löbbing, Sieling, and Wegener [95] showed that Gaussian elimination cannot be avoided when computing reduced $\oplus$-OBDDs such that a linear time reduction algorithm is unlikely to exist. Efficient algorithms for the other basic operations listed in Sect. 2 except SAT-count are presented by Gergov and Meinel [62] and Waack [133]. The synthesis algorithm for $\oplus$-OBDDs is a generalization of the synthesis algorithm for OBDDs. In particular, the size of a $\oplus$-OBDD for $f \wedge g$ is bounded by $O(|V_f||V_g|)$ where V_f and V_g are the sizes of the given $\oplus$-OBDDs for f and g, respectively. Since $\oplus$-OBDDs are not a canonical representation, the equivalence test of functions f and g is performed by computing a $\oplus$-OBDD for $f \oplus g$ and testing this $\oplus$-OBDD for nonsatisfiability. SAT-count for $\oplus$-OBDDs is NP-hard since it is already NP-hard for OFDDs.

$\oplus$-OBDDs seem to be quite difficult to implement because the number of successors of each node is not fixed. Furthermore, $\oplus$-OBDDs are not useful for the representation of multipliers, since Gergov [60] proved an exponential lower bound on the size of $\oplus$-OBDDs for integer multiplication.

3.2 BDDs with generalized variable orderings

Free BDDs (FBDDs) are the extension of OBDDs that is obtained by relaxing the variable ordering condition (5) of definition 1. On each path in an FBDD each variable may be tested at most once, but different from OBDDs the variables may be tested in different orderings on different paths. Even in the early paper of Fortune, Hopcroft, and Schmidt [54] an example of a function is presented for which OBDDs are of exponential size while FBDDs are of polynomial size. As an example of such a function we consider the hidden weighted bit function HWB. This function was introduced by Bryant [24] as an example of a simple function without polynomial size OBDDs. It is defined as

$$HWB(x_1, \dots, x_n) = \begin{cases} x_s \text{ if } s \geq 1, \\ 0 \quad \text{if } s = 0. \end{cases}$$

On the right of Fig. 13 an FBDD for HWB_6 is shown. We only remark that this construction can be generalized to show that there are FBDDs for HWB of quadratic size (Sieling and Wegener [125]; for Bryant, see [61]).
However, in the description of the synthesis algorithm for OBDDs we saw that it is essential to encounter the variables in the given OBDDs in the same ordering. Hence, the synthesis algorithm cannot be generalized to (unrestricted) FBDDs. In order to get a similar synthesis algorithm we restrict FBDDs in such a way that in different FBDDs for the same input the variables are found in the same ordering. On the other hand, for different inputs the orderings may be different. Hence, we obtain more general variable orderings than for OBDDs. Such generalized variable orderings can be represented by a BDD-like graph called a *graph ordering*. We point out that graph orderings do not represent functions. The following definition describes the properties of graph orderings as well as the requirements on FBDDs respecting a graph ordering G. Such FBDDs are also called graph-driven BDDs or G-FBDDs. They were introduced by Sieling and Wegener [125] and Gergov and Meinel [61].

Definition 4. A *graph ordering* G over the variables $x_1, \dots, x_n$ is a graph with the following properties:

1. There is one node, called the *source* without incoming edges.
2. There is one sink.
3. All non-sink nodes are labeled by a variable and have an outgoing 0-edge and an outgoing 1-edge.
4. On each directed path from the source to the sink each variable occurs exactly once.

A G-FBDD G' is a graph that fulfills the properties 1–4 of definition 1 and the following relaxed ordering condition: for each input a on the computation path for a in G' the variables are found in the same ordering as on the computation path for a in G (where in G' variables may be omitted).

An example of a graph ordering is shown on the left of Fig. 13, and the FBDD for HWB given on the right is driven by this graph ordering.

We note that the same reduction rules can be applied to FBDDs as to OBDDs. For each graph ordering G, reduced G-FBDDs are a canonical representation. Hence, the satisfiability test and the equivalence test can be done as for OBDDs. In addition, the synthesis operation can be done in similar way to OBDDs. However, when computing a G-FBDD for $f \otimes g$ from G-FBDDs G_f and G_g for f and g, we also have to take into account the graph ordering G such that the size of the output is only bounded by $|G||G_f||G_g|$.

The main difference to OBDDs is the complexity of the replacement operations (see Sieling and Wegener [125]). Let us consider the replacement of x_i by 0. In OBDDs, we have the situation that the OBDDs starting at the successors of each x_i-node have the same variable ordering. Hence, we can replace the pointers to each x_i-node v by pointers to the 0-successor of v. In G-FBDDs

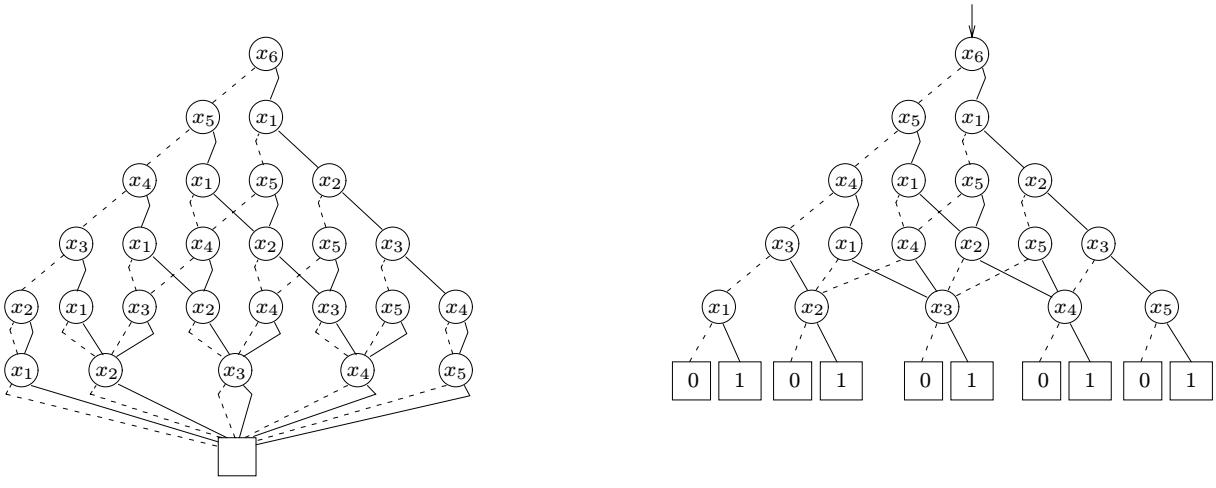

Fig. 13. A graph ordering G and a G-FBDD for HWB_6

the graph ordering starting at the 0-successor of an x_i-node v may be different from the graph ordering starting at the 1-successor of v. Hence, for each x_i-node v we have to represent the function computed at the 0-successor of v using the graph ordering starting at the 1-successor of v. This may cause on exponential blow-up. Therefore, we have to choose the graph ordering in such a way that for variables x_i that have to be replaced by a constant the graph orderings at both successors of x_i-nodes coincide. In other words, we lose the advantages of FBDDs if we have to perform replacement operations. The same holds for the quantification operations.

Similar to the variable ordering problem for OBDDs we have the problem to choose a suitable graph ordering. Bern, Meinel, and Slobodová [14] present heuristics to create graph orderings which have a tree-like shape. An exact algorithm for FBDD-minimization with exponential run time and a heuristic are suggested by Günther and Drechsler [64]. The problem to minimize FBDDs and, therefore, the problem of computing an optimal graph ordering can be shown to be NP-hard (Sieling [120]).

3.3 Decision diagrams with repeated tests

As examples of decision diagrams where the read-once property (condition 4 in definition 1) is relaxed we consider indexed BDDs (IBDDs) and kOBDDs.

Definition 5. An IBDD is a decision diagram fulfilling properties 1–3 of definition 1 and the following extra condition: the set of nodes can be partitioned into k layers of nodes such that for each layer there is a variable ordering. A kOBDD fulfills the further condition that the variable orderings of all k layers are equal. The function represented by an IBDD and a kOBDDs is defined as for OBDDs (definition 2).

IBDDs were suggested by Jain, Bitner, Abadir, Abraham, and Fussell [78] and kOBDDs by Bollig, Sauerhoff, Sieling, and Wegener [19]. Both extensions of OBDDs are considered because the set of functions with polynomial size representations is larger than for OBDDs. An example for a function without polynomial size OBDDs but with IBDDs and kOBDDs of quadratic size is the hidden weighted bit function which we considered in the last section. Remember that OBDDs for HWB have exponential size. Figure 14 shows a 2OBDD (which is also an IBDD) for HWB_4. In the figure the border between the top layer and the bottom layer is indicated by a dotted line. In the top layer of the 2OBDD the number s is computed and the second layer is used to perform a second test of x_s. This construction can be generalized to show that $O(n^2)$ nodes suffice to represent HWB_n in a 2OBDD or an IBDD.

The size of IBDDs and kOBDDs for multiplication depends on the number of layers. Burch [29] constructed BDDs of size $O(n^3)$ for multiplication where in the BDDs

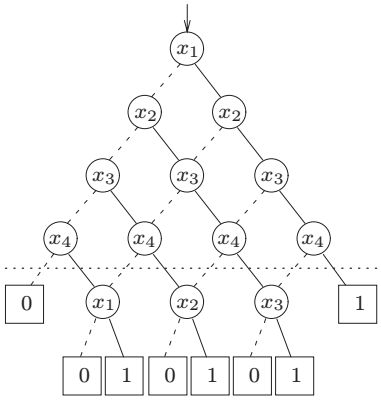

Fig. 14. An IBDD and 2OBDD for HWB_4

variables are tested repeatedly. These BDDs can be seen as IBDDs and kOBDDs with a large number of layers. On the other hand, the results of Gergov [60] imply an exponential lower bound on the size of IBDDs and kOBDDs for multiplication if the number of layers is constant. We see that the size of IBDDs and kOBDDs depends on the number of layers. Bollig, Sauerhoff, Sieling, and Wegener [19] proved that the set of functions representable by polynomial size IBDDs and kOBDDs becomes strictly larger when increasing the number k of layers only by 1.

The definition of IBDDs and kOBDDs implies that we have more possibilities to choose a variable ordering for IBDDs than for kOBDDs. Hence, we may expect that IBDDs with k layers are more powerful than kOBDDs. In fact, Krause [85] (for a complete proof see [84]) proved an exponential lower bound on the size of kOBDDs for the permutation matrix test function. This function gets as input a Boolean matrix and has to test whether each row and each column contains exactly one 1. It is easy to construct an IBDD of linear size for this function. This IBDD only consists of two layers, where in the first layer the variables are arranged in a rowwise ordering and in the second layer in a columnwise ordering.

However, IBDDs and kOBDDs are not a canonical representation. Figure 15 shows an example of a BDD which can be seen as an IBDD or kOBDD consisting of two layers. It is easy to see that neither the S-deletion rule nor the merging rule is applicable to this BDD and that it represents the constant function 0. However, the constant function 0 also has a second representation, namely, the BDD only consisting of the 0-sink. We see that the equiv-

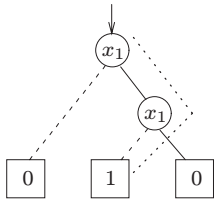

Fig. 15. An IBDD and 2OBDD representing the constant function 0

alence test cannot be performed by a simple isomorphy test as for reduced OBDDs. The figure also shows that IBDDs and kOBDDs may contain inconsistent paths. In the figure such a path is indicated by a dotted line. As already mentioned in Sect. 2.3.1, inconsistent paths make the satisfiability test more difficult since there may be paths from some node to the 1-sink not corresponding to any input. For the satisfiability test of IBDDs Jain, Bitner, Abadir, Abraham, and Fussell [78] present a heuristic algorithm. In fact, the satisfiability test for IBDDs is NP-complete and the equivalence test is coNP-complete even if there are only two layers (Bollig, Sauerhoff, Sieling, and Wegener [19]). For kOBDDs the situation is different. In [19], algorithms for the satisfiability test with a run time of $O(|G|^{2k-1})$ and for the equivalence test with a run time of $O(|G_1|^{2k-1}|G_2|^{2k-1})$ are given. The run time is polynomial if the number k of layers is constant. Hence, we should keep the number of layers as small as possible. The main idea used for the construction of the algorithms is the fact that different layers of a kOBDD have the same variable ordering and, hence, can be combined by the synthesis algorithm for OBDDs. Recently, Günther and Drechsler [66] presented an efficient way how to implement decision diagrams with repeated tests based on OBDD packages.

Finally, we remark that the synthesis algorithm for OBDDs works for IBDDs and kOBDDs as well. One may rename the variables such that the variables in different layers have different names. Then we have an OBDD and may apply the synthesis algorithm for OBDDs. Afterwards, we undo the renaming of the variables and we get an IBDD or kOBDD, respectively. As in the case of OBDDs the size of the result is bounded by $O(|G_f||G_g|)$ where $|G_f|$ and $|G_g|$ are the sizes of representations of the functions f and g. However, there is a second algorithm for synthesis where the size of the output is bounded by $O(|G_f|+|G_g|)$. For example, we may get an IBDD (or kOBDD) for $f \wedge g$ by replacing the 1-sink of G_f by a copy of G_g. In this synthesis algorithm the number of layer increases which makes satisfiability and equivalence more difficult. Thus, it is reasonable to apply the second synthesis algorithm only if the result of the first synthesis algorithm is too large.

3.4 Transformed BDDs

Transformed BDDs (TBDDs) were suggested by Bern, Meinel, and Slobodová [13, 100]. The main of idea of TBDDs is to represent a function f by an OBDD for a function g where $f = g \circ \tau$ for a one-to-one function $\tau : \{0,1\}^n \to \{0,1\}^m$. This is a quite general concept because there are many possibilities to choose τ. In particular, IBDDs, kOBDDs, and graph-driven BDDs can be implemented by a suitable choice of τ. Before we discuss possible choices of τ we mention the advantages of this concept. First, besides the transformation τ, we only have to represent OBDDs. For this we can use the available OBDD-packages. In addition, the synthesis algorithm of the OBDD-package can be used. In fact, we implicitly used the concept of TBDDs when describing the synthesis algorithm for IBDDs and kOBDDs by a renaming of the variables and the application of the synthesis of OBDDs. Consider, for example, a 2OBDD G with the variable ordering $x_1, \ldots, x_n$ in both layers. We may choose $\tau(x_1, \ldots, x_n) = (x_1, \ldots, x_n, x_1, \ldots, x_n)$ and may consider G as an OBDD with $2n$ distinct variables. The second advantage of TBDDs is that the transformation can help to reduce the size of the representation.

However, of course, we encounter the new problem of how to choose a suitable transformation and how to represent the transformation. For this reason only quite restricted transformations have been considered so far.

The first one is the use of graph orderings as transformation. We saw in Sect. 3.2 that a graph ordering G can be used to compute for each input $a_1, \ldots, a_n$ the permutation π such that in each G-FBDD the variable ordering $x_{\pi(1)}, \ldots, x_{\pi(n)}$ is chosen. Let $\tau(a_1, \ldots, a_n) = (a_{\pi(1)}, \ldots, a_{\pi(n)})$. It can easily be shown that this mapping is one-to-one. We note that $a_{\pi(1)}, \ldots, a_{\pi(n)}$ is the ordering in which the variables are tested in each G-FBDD such that one may see τ as a renaming of the variables so that G-FBDDs become OBDDs. Hence, we can implement graph-driven BDDs as TBDDs where the graph ordering G is a description of the transformation τ. However, the application of the merging rule leads to slightly different representations when implementing graph-driven BDDs as described in Sect. 3.2 or when using TBDDs. These differences are considered in Sieling and Wegener [122]. For example, as mentioned above there are graph-driven BDDs for the hidden weighted bit function HWB of quadratic size. It can be shown that using the transformation function defined by the graph ordering described above leads to a TBDD for HWB that only consists of three nodes independent of the number of variables. However, this does not mean that the representation size of HWB is constant, since the representation of the graph ordering also has to be taken into account.

The second type of transformations that have been considered are linear transformations. Then a transformation function $\tau(x_1, \ldots, x_n) = (l_1(x), \ldots, l_n(x))$ has to be chosen where $l_1(x), \ldots, l_n(x)$ are linear combinations of the variables $x_1, \ldots, x_n$. In order to ensure that τ is one-to-one the linear combinations have to be linearly independent. One may understand this variant of TBDDs as OBDDs where at the nodes linear combinations of variables may be tested. A similar idea was already used by Aborhey [1]. An example of an OBDD with linear tests is shown in Fig. 16. The semantics of this generalization of OBDDs is defined in the obvious way. At a node labeled by $l_i(x)$ the computation path follows the outgoing c-edge iff $l_i(a) = c$ for the given input a.

Günther and Drechsler [65] presented an example of a function which can be represented in polynomial size using OBDDs with linear transformations, but only in

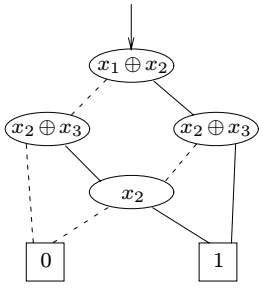

Fig. 16. An example of an OBDD with linear tests

exponential size using ordinary OBDDs. The results of Bern, Meinel, and Slobodová [13] imply that the synthesis algorithm for OBDDs can be applied for OBDDs with linear tests as well, and that OBDDs with linear transformations are a canonical representation. Lower bound results for OBDDs with linear transformations and for some of their generalization are presented in Sieling [121].

There remains the problem of choosing a suitable linear transformation. Günther and Drechsler [63] presented an exact algorithm to determine an optimal linear transformation such that the OBDD size is minimized, but due to its exponential runtime it is only applicable to functions depending on a small number of variables. In Sect. 2.4 we showed how a swap of two adjacent variables can be performed efficiently by redirecting pointers. In a similar way, it is possible to combine linear combinations l_i and l_{i+1} of adjacent levels to the new linear combinations l_i and $l_i \oplus l_{i+1}$ on these levels and to compute the corresponding OBDD with linear tests. Based on this observation, Meinel, Somenzi, and Theobald [101] extended the sifting algorithm of Rudell [115], which we discussed in Sect. 2.4, to optimize linear transformations.

3.5 Word-level decision diagrams

All DD types presented so far can only represent Boolean functions, i.e., functions $f : \{0,1\}^n \to \{0,1\}^m$. Recently, (especially in the area of verification) DDs have also been used to represent integer-valued functions, i.e., functions of the form $f : \{0,1\}^n \to \mathbf{Z}$. Many functions are very complex, when considered at the bit-level, while they become much simpler when described at word-level (see, for example, integer multiplication). For these functions the size becomes exponential for OBDDs independent of the variable ordering, but WLDDs can represent them using a polynomial number of nodes.

Before we describe WLDDs in more detail, we generalize the following three decompositions for word-level functions:

$$f_v = (1 - x_i) f_{v|x_i=0} + x_i f_{v|x_i=1}$$
$$f_v = f_{v|x_i=0} + x_i (f_{v|x_i=1} - f_{v|x_i=0})$$
$$f_v = f_{v|x_i=1} + (1 - x_i)(f_{v|x_i=0} - f_{v|x_i=1})$$

The notation S, pD and nD is used analogously to the bit-level. x_i still denotes a Boolean variable, but the values of the functions are integer numbers and they are combined with the usual operations (addition, subtraction, and multiplication) in the ring $\mathbf{Z}$ of integers.

The simplest extension of OBDDs is to introduce non-Boolean terminals, i.e., to allow more than two terminals in reduced graphs. The resulting DDs are called *multi-terminal BDDs* (MTBDDs) and have been introduced by Clarke et al. [33] or *algebraic decision diagrams* (ADDs) presented by Bahar et al. [6] where in each node an (integer-valued) Shannon decomposition is carried out. Notice that the variables are still Boolean. In Fig. 17 an MTBDD for the unsigned integer encoding is given. The reduction rules are analogous to those applied to OBDDs. The synthesis algorithms known for OBDDs can be directly transferred and have the same worst-case complexity. However, the major drawback of this data structure is that the number of nodes increases very fast, if the number of different terminals becomes large.

As an alternative, edge values are introduced to increase the amount of subgraph sharing when using integer-valued terminal nodes. Lai and Sastry [93] have presented *edge-valued binary decision diagrams* (EVBDDs), which are MTBDDs where an edge weight a is *added* to the function being represented. Thus, in the EVBDD, an edge with weight a to a node v labeled with variable x_i represents the function

$$\langle a, f_v \rangle = a + (1 - x_i) f_{v|x_i=0} + x_i f_{v|x_i=1}.$$

If, additionally, a multiplicative edge-weight is allowed the DDs are called *factored EVBDDs* (FEVBDDs) as presented by Tafertshofer and Pedram [128].

Bryant and Chen [27, 28] introduced *(multiplicative) binary moment diagrams* ((*)BMDs). BMDs make use of the (integer-valued) *positive Davio decomposition* (pD) and allow terminal nodes labeled with integer values (analogous to MTBDDs), i.e., they are the integer-valued generalization of OFDDs. *BMDs are a generalization of BMDs in the sense that they allow multiplicative edge weights: the values at the edges are multiplied with the functions represented. Thus, an edge with weight m to

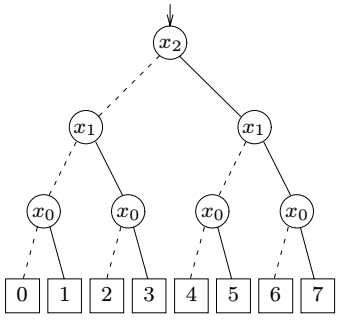

Fig. 17. MTBDD for unsigned integer encoding

a node v in a *BMD represents the function

$$\langle m, f_v \rangle = m(f_{v|x_i=0} + x_i(f_{v|x_i=1} - f_{v|x_i=0})).$$

Using this data structure, it was possible for the first time to verify multipliers of large bit length (see Sect. 4 below).

Next, the idea from OKFDDs to use more than one decomposition in a graph has been considered. *Kronecker binary moment diagrams* (KBMDs) proposed by Drechsler, Becker, and Ruppertz [45] (or *hybrid decision diagrams* (HDDs) as they are called by Clarke, Fujita, and Zhao [34]) try to combine the advantages of MTBDDs and BMDs. Analogous to OKFDDs at the bit-level, different decomposition types per variable can be used. Since we consider integer-valued functions a lot of different decomposition types are possible. They can be defined by the set $\mathbf{Z}_{2,2}$ of non-singular 2×2 matrices over $\mathbf{Z}$ [34]. As for OKFDDs decomposition types are associated to the n Boolean variables with the help of a *decomposition type list* (DTL) $d := (d_1, \ldots, d_n)$ where $d_i \in \mathbf{Z}_{2,2}$, i.e., for each variable one fixed decomposition is chosen.

K*BMDs [45] differ from KBMDs in the fact that they allow the use of integer weights, additive and multiplicative weights in parallel (as it has been considered in FEVBDDs). K*BMDs (and FEVBDDs) make use of the following type of representation:

$$\langle (a, m), f_v \rangle := a + m f_v$$

In contrast to FEVBDDs, which are based on Shannon decomposition, K*BMDs allow different decomposition types per variable. In the case of Shannon decomposition and positive and negative Davio decomposition the function represented at an edge that has the weight (a, m) and leads to the node v is then given by one of the following equations, respectively:

$$\begin{aligned}
\langle (a, m), f_v \rangle &= a + m((1 - x_i) f_{v|x_i=0} + x_i f_{v|x_i=1}) \\
\langle (a, m), f_v \rangle &= a + m(f_{v|x_i=0} + x_i(f_{v|x_i=1} - f_{v|x_i=0}) \\
\langle (a, m), f_v \rangle &= a + m(f_{v|x_i=1} \\
&\qquad + (1 - x_i)(f_{v|x_i=0} - f_{v|x_i=1}))
\end{aligned}$$

To make DDs with edge values a canonical representation, some further restrictions on the graph with respect to weights are required. For simplicity, here we only comment on these restrictions for the case of K*BMDs. (For other DD types the restrictions are similar.) Basically the following is required: there exists only one terminal and this terminal is labeled 0, the 0-edge of a node has additive weight 0 and the remaining weights have *greatest common divisor* 1. A K*BMD for the unsigned integer encoding is given in Fig. 18. (The *BMD for the same function is given in [27].) At the edges the additive and multiplicative values are displayed by (a, m). The decomposition type of each node is also given. For efficient implementation of word-level DDs see [72].

Recently, Chen and Bryant [32] proposed a data structure, called *multiplicative power hybrid decision diagrams*

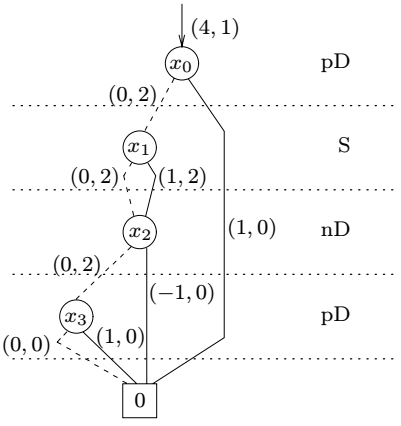

Fig. 18. K*BMD for unsigned integer encoding

(*PHDDs). The data structure is in principle similar to K*BMDs, but the edge values are interpreted as powers of a given basis. *PHDDs (with basis two) have been successfully used for verification of circuits computing floating point values.

Analogous to bit-level DDs, recursive synthesis algorithms can be described for word-level DDs. Again, exponential blow-ups can occur as in the case of Davio based bit-level DDs [9]. (For a detailed description of synthesis operations for word-level DDs see [28].) Depending on the type of operation and the function being represented, Drechsler and Höreth [49] showed that in some cases polynomial upper bounds can also be proven. Furthermore, several common algorithms known from OBDDs can be transferred, for example, such as dynamic variable reordering, since the variable ordering influences the size of the DD significantly, analogously to OBDDs. If DDs without edge values are considered, the reordering algorithms for OBDDs can be used. Höreth and Drechsler [73] proposed two methods on how to perform variable reordering in the presence of edge values, i.e., modifying the reduction algorithm or slightly modifying the standard sifting algorithm from Rudell [115]. These methods can be further sped up (analogously to OBDDs) using lower bound computations as proposed by Günther, Drechsler, and Höreth [67]

We conclude this section with some comments on theoretical results that give some further insight into the relation between the different data structures. As described above for bit-level DDs, it turns out that there exist functions that can only be represented by OBDDs efficiently but not by OFDDs, and vice versa. By definition OKFDDs are a superset of OBDDs and OFDDs, and thus combine the advantages both with respect to representation size; it even can be shown, that they represent functions efficiently where OBDDs and OFDDs fail. A theoretical background for word-level DDs has been provided by Becker, Drechsler, and Enders [9, 53]. Here, exponential trade-offs have been proven for DDs with and without edge-weights and for the three different decomposition types, i.e., Shannon, and positive and

negative Davio. The diagram in Fig. 19 displays the relation between the most commonly used types of DDs. The solid lines show the inclusion relation, e.g., OBDDs are a subset of OKFDDs. On the left- (right-) hand side the Shannon-based (Davio-based) DD types are given. The DDs in the middle column are hybrid in the sense that they allow different decomposition types within the graph.

For more details on word-level DDs see [41].

4 Applications

In this section, we outline some applications of OBDDs and also briefly mention fields where the extensions introduced above have been used. The list is not complete in the sense that "all" applications are covered and not all applications have equal importance. Furthermore, not all approaches presented in the literature can be reviewed. Instead, some representative ones are selected. However, the different sections in the following outline various perspectives regarding what should be considered when using DDs. (Further discussion on experiences when using BDDs can be found in [69].) Here, only the main ideas of the applications are proposed, while the papers in this special issue will give more details for some selected applications.

4.1 Verification

Nowadays modern circuit design can contain several million transistors. Verification of such large designs is becoming more and more difficult, since pure simulation cannot guarantee correct behavior and exhaustive simulation is too time consuming. Verification (in its "simplest" form) consists of checking two function representations, e.g., two circuits, for equivalence. If an OBDD can be constructed this problem becomes easy, since OBDDs are a canonical data structures. After OBDDs were introduced in the mid 1980s they were used in compiled simulators by Bryant et al. [26]. First approaches by Malik et al. [97] and Fujita et al. [57] using OBDDs for equivalence checking mainly focus on finding good variable orderings from the circuit description (also see Sect. 2.4). OBDDs are nowadays the state-of-the-art data structure in verification and have been integrated in many commercial tools. An example of a verification procedure has been described by Appenzeller and Kuehlmann [3]. If the OBDD construction cannot be completed due to memory limitations, combinations of OBDDs and structural approaches can also be used. A successful approach using "cuts and heaps" has been described by Kuehlmann and Krohm [89].

Nevertheless, for some "important" functions, such as multiplication, OBDDs fail due to their exponential size. In these cases, especially when dealing with arithmetic circuits, word-level DDs as introduced in the previous section can be applied. Using *BMDs, Bryant and Chen [27] showed that multipliers up to 256 bits could be verified. Unfortunately, *BMDs fail for the representation of circuits that can easily be represented using OBDDs [46]. For this purpose, hybrid DDs, such as HDDs and K*BMDs, have been applied. By definition K*BMDs are a superset of *BMDs and allow in all cases a representation at least as efficient as OBDDs. In particular, in applications where datapath and control logic have to be verified, a hybrid data structure should be chosen. Recent studies by Cyrluk, Möller, and Rueß [39] and Höreth and Drechsler [74] have presented "real world" examples where

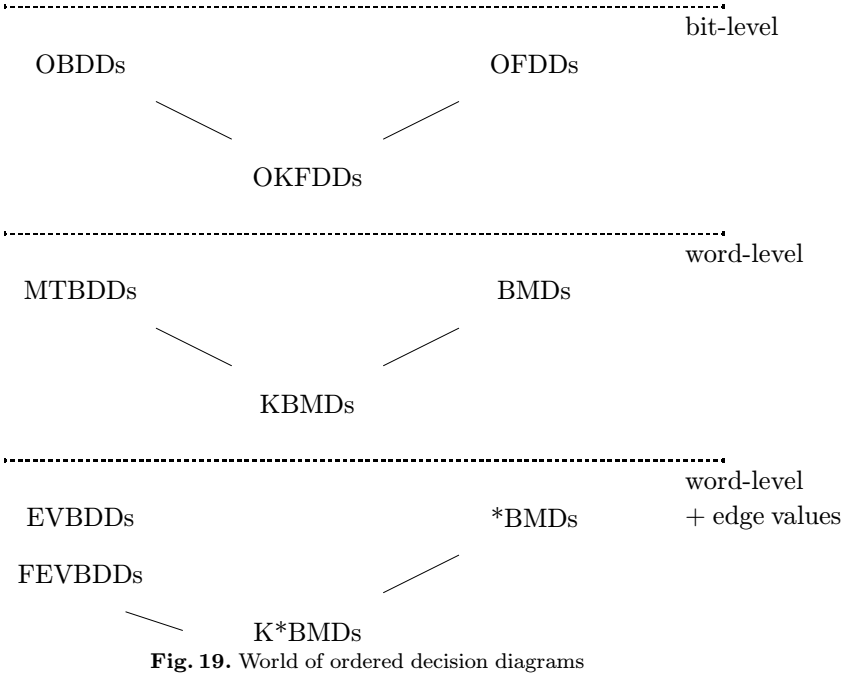

Fig. 19. World of ordered decision diagrams

only mixed representations can be applied, while pure approaches based on OBDDs and *BMDs, respectively, fail.

In addition, in the field of sequential circuits OBDDs have shown to work very well as demonstrated by Coudert, Berthet, and Madre [37]. Using OBDDs, reachability analysis and image computation for finite state machines with more than several million states could be verified. These techniques have been applied in the field of symbolic model checking by Burch et al. [30] and McMillan [98]. Based on OBDDs, industrial designs have also been successfully checked. The integration of WLDDs in symbolic model checking has been proposed by Clarke and Zhao [35].

4.2 Logic synthesis

The classic 2-level minimizers, such as ESPRESSO [22], used an enumeration of the terms in an array structure for representing the prime implicants. Coudert, Fraisse, and Madre [38] showed that using DDs tremendously improved the performance of the algorithms. For more details see [36].

An interesting approach for FPGA design is based on decompositions of functions as proposed by Ashenhurst [5]. The corresponding decomposition can easily be expressed by OBDDs and several researchers have studied the corresponding algorithms intensively (see, for example, [91, 118, 137]). The interesting property from the DD point of view is that a cut through the OBDD has to be found that crosses a minimal number of edges. Thus, minimization algorithms for OBDDs, e.g., based on sifting, are obviously not directly applicable and should be tailored to this context.

The construction of circuits resulting from a direct mapping of OBDDs has gained great interest, since the graph directly corresponds to a multiplexor network. One argument is that the mapping process can be simplified due to the simple structure. Additionally, the resulting circuits have good testability properties [4, 7, 8]. In this area, more general types than OBDDs, such as OKFDDs, have several advantages, since in the area of circuit design a small reduction in the graph size may also have a large influence on the size of the resulting circuit.

4.3 Testing

One possibility of using DDs in the area of testing has just been mentioned, i.e., in the form of design for testability. Alternatively, the symbolic representation of a circuit by OBDDs can be used for many problems in testing, such as test pattern generation [15, 40, 127] and fault simulation [87]. Using OBDDs, it was possible for the first time to compute exact signal probabilities and fault detection probabilities (see Krieger et al. [86, 88]). These probabilities are used in different areas in testing, such as weight optimization in built-in self-testing.

4.4 Combinatorial optimization and integer programming

DDs have been used in many other combinatorial optimization problems. For example, in many tasks during logic synthesis sets of elements have to be represented. ZBDDs have been introduced by Minato [104] (also see [107] in this special section) especially for this problem. Even though from the theoretical point of view they only differ by a linear factor from OBDDs with respect to size [12] they are frequently used in applications, such as 2-level minimization [36]. In addition, for multilevel synthesis they can be used, for example, for factorization in logic synthesis [106].

ZBDDs have also been used in "completely" different areas, e.g., counting the number of knight's tours on a chessboard (Löbbing and Wegener [96]). The main idea is to construct ZBDDs for which each satisfying input corresponds to a knight's tour. Thus, the operation SAT-count can be used to obtain the number of knight's tours. We only remark that several refinements are necessary since a ZBDD representing all knight's tours is much too large.

Lai, Pedram, and Vrudhula [92] use edge-valued BDDs for the implementation of an algorithm solving 0-1-integer linear programs. Such an optimization problem consists of a linear goal function

$$c_1 x_1 + \cdots + c_n x_n \to \min,$$

where $c_i \in \mathbf{Z}$, the m linear constraints

$$a_{i,1} x_1 + \cdots + a_{i,n} x_n \leq b_i,$$

where $i \in \{1, \ldots, m\}$ and $a_{i,j}, b_i \in \mathbf{Z}$, and the integrality constraints

$$x_1, \ldots, x_n \in \{0, 1\}.$$

It is well-known that solving 0-1-integer linear programs is NP-hard such that we can only hope for heuristic approaches. One can easily see that the goal function and the left-hand sides of the linear constraints can be represented by EVBDDs of linear size. The main ideas of the approach of Lai, Pedram, and Vrudhula are roughly the following ones. In a first step from the EVBDD for each linear constraint an OBDD is constructed that computes 1 on those inputs for which the constraint is fulfilled. Then the constructed OBDDs are combined by the logical AND. Finally, by a procedure similar to synthesis, the EVBDD for the goal function and the OBDD representing all linear constraints are combined in order to obtain an assignment to the x-variables that fulfills all constraints and for which the goal function takes the minimum. Since each of these steps may cause an exponential blow-up of the sizes of the BDDs, several refinements are used; in particular, all these ideas are integrated into a branch-and-bound algorithm in order to obtain a solver for 0-1 integer

linear programs. In addition, this example shows that OBDDs and their variants may be useful in quite unrelated areas.

4.5 Binary decision diagrams in complexity theory

As already mentioned in the introduction, BDDs have also been investigated in complexity theory. For a long time researchers in applications and in complexity theory did not know of each other since the terminology was different, e.g., BDDs are called branching programs in complexity theory. However, the motivation for considering BDDs in complexity theory is quite different. There BDDs are explored as a model describing the space complexity of computations. First, BDDs without the read-once property and without variable orderings were considered. It can be proved that polynomial size BDDs represent exactly those functions that can be computed by any reasonable model of sequential computation with a logarithmic amount of memory, see, for example, Wegener [134]. Thus, the problem to prove superlogarithmic space bounds for explicitly defined functions reduces to proving superpolynomial size bounds for BDDs. However, to obtain such a proof is considered to be as difficult as obtaining a solution of the $P \neq NP$ question. For this reason restricted BDDs have been considered in order to develop methods for the proof of lower bounds. Among others the variants of BDDs used in applications are also investigated and powerful methods for proving lower bounds are known. As the most important method we mention the theory of communication complexity. Roughly, the part of the input tested in the upper part of an OBDD has to "communicate" with the part of the input tested in the lower part of the OBDD. If much information has to be exchanged between the parts of the OBDD in order to compute the function, its width has to be large. For an introduction to communication complexity theory we refer to the monographs of Hromkovič [75] and Kushilevitz and Nisan [90]. We already mentioned some lower bound results in Sect. 3. An overview over lower bound results for several variants of BDDs, in particular, variants that are not used in applications, is given by Razborov [113]. We would like to point out that communication complexity is the same measure that is minimized when searching for good decompositions in logic synthesis as described above.

After recognizing the importance of BDD-like representations as a data structure for Boolean functions, researchers in complexity theory also investigated the complexity of operations on BDDs. The most important classification of problems in complexity theory is the distinction between problems solvable in polynomial time and NP-complete problems (see Garey and Johnson [59]). The advantage of this classification is that it is independent of the model of computation. However, in some applications exponential time algorithms may be helpful for small instances, while in other applications even polynomial time algorithms may be too slow. Nevertheless, the classification helps to determine whether it makes sense to search for an efficient algorithm for solving the considered problem exactly or whether we have to be satisfied with approximate solutions or heuristic solutions. Examples of negative results from complexity theory that solve problems raised in applications are the nonapproximability result for the variable ordering problem (theorem 3), or the result of Scholl, Becker, and Weis [117] and Thathachar [131] that word-level decision diagrams for division have exponential size and, therefore, cannot be used to verify division circuits. An example of a positive result is the introduction of $\oplus$-OBDDs by Waack [133]. Although they seem to be difficult to implement, they explain and show how to avoid the exponential blow-up happening for some operations on OFDDs. Several other results from complexity theory that concern the complexity of operations on OBDDs and their generalizations are mentioned in Sects. 2 and 3.

5 Conclusion

We have presented a brief overview on OBDDs and their generalizations. Theoretical concepts and applications have been discussed as an introduction to the articles of this special section. The six contributions in the following discuss several issues on application in more detail. Practical aspects of implementation of DD packages are given and an evaluation approach for OBDDs is proposed.

We conclude with a list of books that have been published on OBDDs and related topics. The interested reader can find some more information about OBDDs and their generalizations in these books. The books in alphabetic order of the first author's name are: Drechsler and Becker [42], Hachtel and Somenzi [68], Meinel and Theobald [102], Minato [105], and Wegener [135].

6 Contributions to this special section

Finally, after having given this (brief) overview on BDDs, we revisit the articles contributed to this special section and discuss their contents in the context of the previous sections. While mainly basic concepts and several theoretical results are considered in this introductory article, it was the goal to cover many aspects of BDDs that are driven by applications in the contributed articles. We roughly distinguish four main categories:

Concepts: starting from BDDs as introduced in Sect. 2 several extensions of the basic concept have been proposed (see Sect. 3).

Implementations: since the data structures should be applied to real-world problems it turned out to be very important to find clever implementations.

Properties: besides the theoretical aspects as discussed in previous sections, experimental studies have been carried out that provide an in-depth understanding of the behavior of algorithms on BDDs.

Applications: the wide use of BDDs and their extensions is mainly due to their successful application to many problems that were known to be infeasible before.

The articles in the following were selected in such a way that all of these aspects are touched.

We start with an article by Bryant and Chen, where an extension of BDDs to represent so-called pseudo-Boolean functions, i.e., functions with an integer range and a Boolean domain, is introduced. The resulting data structure, called BMDs (see above), especially allows us to represent arithmetic functions very efficiently. Based on BMDs, it was for the first time possible to represent multipliers of several hundred bits input-length very compact. Besides the representation, synthesis algorithms are also presented and it is shown how the algorithms known from BDDs can be extended to this more general case.

The second paper by Minato describes another extension of BDDs, where the focus is not arithmetic functions, but the efficient handling of sets of combinations. Again, first the data structure, called ZBDDs, is described and then algorithms for efficient manipulation are introduced. The efficiency of the algorithms known from BDDs, i.e., polynomial worst-case behavior, can be transferred to this case, too.

The next two papers focus on the implementation of BDDs and their extensions, respectively.

Somenzi describes an efficient implementation of BDDs. Besides the "pure" BDD structure, assistant data structures, such as hash-based unique and computed table, are introduced and their influence is discussed. Furthermore, aspects of the memory management during BDD construction and variable reordering are described. Statistical experiments are given to give an impression of the influence of the various aspects.

The paper by Höreth discusses the problems that result from implementation of extensions of BDDs and presents hybrid algorithms, i.e., algorithms mixing different DD-types. One section is devoted to modulo arithmetic, since this allows us to describe an elegant method for verification of high-level descriptions. Several experiments and a case study are given to provide an impression of the efficiency of the techniques proposed.

While BDDs have been studied intensively from a theoretical point of view, most experimental evaluations have been driven by applications. In the paper by Harlow and Brglez robustness properties are experimentally studied. After introducing the benchmark generation principles, variable ordering heuristics are studied. The behavior of BDDs and algorithms working on them is studied under various aspects of randomization.

Finally, the paper of Mohnke, Molitor, and Malik presents an application of BDDs. BDDs are used for Boolean matching that is one of the underlying problems in combinational equivalence checking. BDDs are used for the computation of signatures of a given function to determine matching candidates. Experiments on benchmark functions show the effectiveness of the approach.

The papers classified according to the four categories introduced above are shown in Fig. 20. The papers have been selected in such a way that all perspectives are considered, even though a complete coverage cannot be given. The papers are ordered starting from the basic data structures moving towards the application.

Acknowledgements. We would like to thank the reviewers for their comments and suggestions for improvement for all papers of this special section.

References

1. Aborhey, S.: Binary decision tree test functions. IEEE Trans Comp 27(11): 1461–1465, 1988
2. Akers, S.B.: Binary decision diagrams. IEEE Trans Comp 27: 509–516, 1978
3. Appenzeller, D.P., Kuehlmann, A.: Formal verification of a PowerPC microprocessor. In: Int Conf Comp Design, pp. 79–84, 1995
4. Ashar, P., Devadas, S., Keutzer, K.: Path-delay-fault testability properties of multiplexor-based networks. Integration, VLSI J 15(1): 1–23, 1993

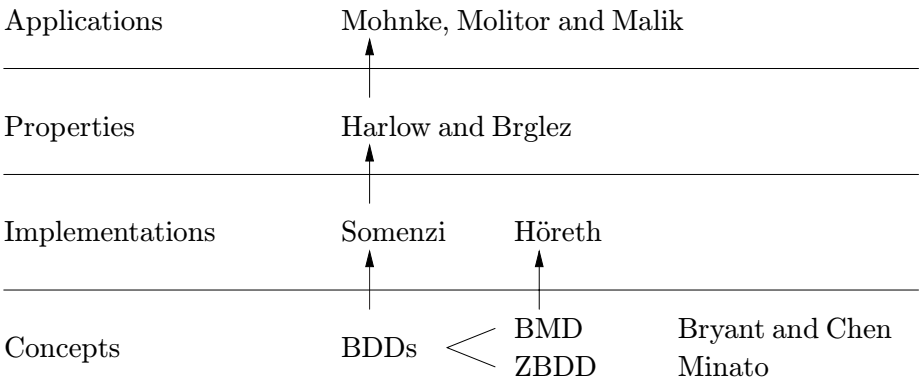

Fig. 20. Classification of contributed papers

5. Ashenhurst, R.L.: The decomposition of switching functions. In: Int Symp Theor Switching Funct, pp. 74–116, 1959
6. Bahar, R.I., Frohm, E.A., Gaona, C.M., Hachtel, G.D., Macii, E., Pardo, A., Somenzi, F.: Algebraic decision diagrams and their applications. In: Int Conf CAD, pp. 188–191, 1993
7. Becker, B.: Synthesis for testability: Binary decision diagrams. In: Symp Theor Aspects Comp Sci. LNCS 577. Berlin, Heidelberg, New York: Springer-Verlag, 1992, pp. 501–512
8. Becker, B., Drechsler, R.: Synthesis for testability: circuits derived from ordered Kronecker functional decision diagrams. In: Eur Des Test Conf, p. 592, 1995
9. Becker, B., Drechsler, R., Enders, R.: On the computational power of bit-level and word-level decision diagrams. In: ASP Des Autom Conf, pp. 461–467, 1997
10. Becker, B., Drechsler, R., Theobald, M.: On the implementation of a package for efficient representation and manipulation of functional decision diagrams. IFIP WG 10.5 Workshop on Applications of the Reed-Muller Expansion in Circuit Design, pp. 162–169, 1993
11. Becker, B., Drechsler, R., Theobald, M.: On the expressive power of OKFDDs. Formal Methods Syst Des 11(1): 5–21, 1997
12. Becker, B., Drechsler, R., Werchner, R.: On the relation between BDDs and FDDs. Inf Comput 123(2): 185–197, 1995
13. Bern, J., Meinel, C., Slobodová, A.: Efficient OBDD-based Boolean manipulation in CAD beyond current limits. In: Des Autom Conf, pp. 408–413, 1995
14. Bern, J., Meinel, C., Slobodová, A.: Some heuristics for generating tree-like FBDD types. IEEE Trans CAD 15: 127–130, 1996
15. Bhattacharya, D., Agrawal, P., Agrawal, V.D.: Test generation for path delay faults using binary decision diagrams. IEEE Trans Comp 44(3): 434–447, 1995
16. Bollig, B., Löbbing, M., Sauerhoff, M., Wegener, I.: Complexity theoretical aspects of OFDDs. In: Sasao, T., Fujita, M. (eds.): Representations of discrete functions, pp. 249–268. Kluwer Academic, Boston, Mass., USA, 1996
17. Bollig, B., Löbbing, M., Wegener, I.: Simulated annealing to improve variable orderings for OBDDs. In: Int Workshop Logic Synth, pp. 5.1–5.10, 1995
18. Bollig, B., Löbbing, M., Wegener, I.: On the effect of local changes in the variable ordering of ordered decision diagrams. Info Process Lett 59: 233–239, 1996
19. Bollig, B., Sauerhoff, M., Sieling, D., Wegener, I.: Hierarchy theorems for *k*OBDDs and *k*IBDDs. Theor Comput Sci 205: 45–60, 1998
20. Bollig, B., Wegener, I.: Improving the variable ordering of OBDDs is NP-complete. IEEE Trans Comp 45(9): 993–1002, 1996
21. Brace, K.S., Rudell, R.L., Bryant, R.E.: Efficient implementation of a BDD package. In: Des Automat Conf, pp. 40–45, 1990
22. Brayton, R.K., Hachtel, G.D., McMullen, C., Sangiovanni-Vincentelli, A.L..: Logic minimization algorithms for VLSI synthesis. Kluwer Academic, Boston, Mass., USA, 1984
23. Bryant, R.E.: Graph-based algorithms for Boolean function manipulation. IEEE Trans Comp 35(8): 677–691, 1986
24. Bryant, R.E.: On the complexity of VLSI implementations and graph representations of Boolean functions with application to integer multiplication. IEEE Trans Comp 40: 205–213, 1991
25. Bryant, R.E.: Binary decision diagrams and beyond: enabling techniques for formal verification. In: Int Conf CAD, pp. 236–243, 1995
26. Bryant, R.E., Beatty, D., Brace, K., Cho, K., Sheffler, T.: COSMOS: a compiled simulator for MOS circuits. In: Des Autom Conf, pp. 9–16, 1987
27. Bryant, R.E., Chen, Y.-A..: Verification of arithmetic functions with binary moment diagrams. In: Des Autom Conf, pp. 535–541, 1995
28. Bryant, R.E., Chen, Y.-A..: Verification of arithmetic circuits using binary moment diagrams. Software Tools Technol Transfer (this issue)
29. Burch, J.R.: Using BDDs to verify multipliers. In: Des Automat Conf, pp. 408–412, 1991
30. Burch, J.R., Clarke, E.M., McMillan, K.L., Dill, D.L., Hwang, L.J.: Symbolic model checking: 10^{20} states and beyond. Inf Comput 98(2): 142–170, 1992
31. Butler, K.M., Ross, D.E., Kapur, R.K., Mercer, M.R.: Heuristics to compute variable orderings for efficient manipulation of ordered binary decision diagrams. In: Des Autom Conf, pp. 417–420, 1991
32. Chen, Y.-A., Bryant, R.E.: *PHDD: an efficient graph representation for floating point circuit verification. In: Int Conf CAD, pp. 2–7, 1997
33. Clarke, E., Fujita, M., McGeer, P., McMillan, K.L., Yang, J., Zhao, X.: Multi terminal binary decision diagrams: An efficient data structure for matrix representation. In: Int Workshop Logic Synth, pp. P6a: 1–15, 1993
34. Clarke, E.M., Fujita, M., Zhao, X.: Hybrid decision diagrams – overcoming the limitations of MTBDDs and BMDs. In: Int Conf CAD, pp. 159–163, 1995
35. Clarke, E.M., Zhao, X.: Word level symbolic model checking - a new approach for verifying arithmetic circuits. Technical Report CMU-CS-95-161, 1995
36. Coudert, O.: Two-level logic minimization: an overview. Integration, VLSI J 17(2): 97–140, 1994
37. Coudert, O., Berthet, C., Madre, J.C.: Verification of sequential machines using Boolean functional vectors. In: Proc IFIP Int Workshop Applied Formal Methods Correct VLSI Des, pp. 111–128, 1989
38. Coudert, O., Fraisse, H., Madre, J.C.: A breakthrough in two-level logic minimization. In: Int Workshop Logic Synth, p. P2b, 1993
39. Cyrluk, D., Möller, O., Rueß, H..: An efficient decision procedure for the theory of fixed-sized bitvectors. In: Comput Aided Verification. LNCS 1254. Berlin, Heidelberg, New York: Springer-Verlag, 1997
40. Drechsler, R.: BiTeS: a BDD based test pattern generator for strong robust path delay faults. In: Eur Des Autom Conf, pp. 322–327, 1994
41. Drechsler, R.: Formal verification of circuits. Kluwer Academic, Boston, Mass., USA, 2000
42. Drechsler, R., Becker, B.: Binary decision diagrams - theory and implementation. Kluwer Academic, Boston, Mass., USA,1998
43. Drechsler, R., Becker, B., Göckel, N.: A genetic algorithm for variable ordering of OBDDs. IEE Proc 143(6): 364–368, 1996
44. Drechsler, R., Becker, B., Jahnke, A.: On variable ordering and decomposition type choice in OKFDDs. In: IEEE Trans Comp 47(12), December 1998
45. Drechsler, R., Becker, B., Ruppertz, S.: K*BMDs: a new data structure for verification. In: Eur Des Test Conf, pp. 2–8, 1996
46. Drechsler, R., Becker, B., Ruppertz, S.: The K*BMD: a verification data structure. IEEE Des Test Comp, pp. 51–59, 1997
47. Drechsler, R., Drechsler, N., Günther, W.: Fast exact minimization of BDDs. In: Des Autom Conf, pp. 200–205, 1998
48. Drechsler, R., Günther, W.: Using lower bounds during dynamic BDD minimization. In: Des Autom Conf, pp. 29–32, 1999
49. Drechsler, R., Höreth, S.: Manipulation of *BMDs. In: ASP Des Autom Conf, pp. 433–438, 1998
50. Drechsler, R., Sarabi, A., Theobald, M., Becker, B., Perkowski, M.A.: Efficient representation and manipulation of switching functions based on ordered Kronecker functional decision diagrams. In: Des Autom Conf, pp. 415–419, 1994
51. Drechsler, R., Sauerhoff, M., Sieling, D.: The complexity of the inclusion operation on OFDDs. IEEE Trans CAD 17(5): 457–459, 1998
52. Drechsler, R., Theobald, M., Becker, B.: Fast OFDD based minimization of fixed polarity Reed-Muller expressions. In: European Des Autom Conf, pp. 2–7, 1994
53. Enders, R.: Note on the complexity of binary moment diagram representations. IFIP WG 10.5 Workshop Appl Reed-Muller Expansion Circuit Des, pp. 191–197, 1995
54. Fortune, S., Hopcroft, J., Schmidt, E.M.: The complexity of equivalence and containment for free single variable program schemes. In: Int Colloquium Automat Lang Comput. LNCS 62. Berlin, Heidelberg, New York: Springer-Verlag, 1997, pp. 227–240

55. Friedman, S.J., Supowit, K.J.: Finding the optimal variable ordering for binary decision diagrams. IEEE Trans Comp 39(5): 710–713, 1990
56. Fujii, H., Ootomo, G., Hori, C.: Interleaving based variable ordering methods for ordered binary decision diagrams. In: Int Conf CAD, pp. 38–41, 1993
57. Fujita, M., Fujisawa, H., Kawato, N.: Evaluation and improvements of Boolean comparison method based on binary decision diagrams. In: Int Conf CAD, pp. 2–5, 1988
58. Fujita, M., Matsunaga, Y., Kakuda, T.: On variable ordering of binary decision diagrams for the application of multi-level logic synthesis. In: Eur Conf Des Autom, pp. 50–54, 1991
59. Garey, M.R., Johnson, D.S.: Computers and intractability – a guide to the theory of NP-completeness. Freemann, San Francisco, 1979
60. Gergov, J.: Time-space tradeoffs for integer multiplication on various types of input oblivious sequential machines. Inf Process Lett 51: 265–269, 1994
61. Gergov, J., Meinel, C.: Efficient analysis and manipulation of OBDD's can be extended to FBDD's. IEEE Trans Comp 43: 1197–1209, 1994
62. Gergov, J., Meinel, C.: Mod-2-OBDD's – a data structure that generalizes EXOR-sum-of-products and ordered binary decision diagrams. Formal Methods Syst Des 8: 273–282, 1996
63. Günther, W., Drechsler, R.: Linear transformations and exact minimization of BDDs. In: Great Lakes Symp VLSI, pp. 325–330, 1998
64. Günther, W., Drechsler, R.: Minimization of free BDDs. In: ASP Des Autom Conf, 1999
65. Günther, W., Drechsler, R.: On the computational power of linearly transformed BDDs. Inf Process Lett 119-125(75), 2000
66. Günther, W., Drechsler, R.: Implementation of read-k-times BDDs on top of standard BDD packages. In: VLSI Design Conf, pp. 173–178, 2001
67. Günther, W., Drechsler, R., Höreth, S.: Efficient dynamic minimization of word-level DDs based on lower bound computation. In: Int Conf Comp Des, pp. 383–388, 2000
68. Hachtel, G., Somenzi, F.: Logic synthesis and verification algorithms. Kluwer Academic, Boston, Mass., USA 1996
69. Harlow, J., Brglez, F.: Design of experiments and evaluation of BDD ordering heuristics. Software Tools Technol Transfer, (this issue)
70. Hett, A., Drechsler, R., Becker, B.: MORE: Alternative implementation of BDD packages by multi-operand synthesis. In: Eur Des Autom Conf, pp. 164–169, 1996
71. Höreth, S.: Compilation of optimized OBDD-algorithms. In: Eur Des Autom Conf, pp. 152–157, 1996
72. Höreth, S.: A word-level graph manipulation package. Software Tools Technol Transfer, (this issue)
73. Höreth, S., Drechsler, R.: Dynamic minimization of word-level decision diagrams. In: Des Autom Test Eur, pp. 612–617, 1998
74. Höreth, S., Drechsler, R.: Formal verification of word-level specifications. In: Des Autom Test Eur, pp. 52–58, 1999
75. Hromkovič, J..: Communication complexity and parallel computing. Berlin, Heidelberg, New York: Springer-Verlag, 1997
76. Ibarra, O.H., Kim, C.E.: Fast approximation algorithms for the knapsack and sum of subset problems. J ACM 22: 463–468, 1975
77. Ishiura, N., Sawada, H., Yajima, S.: Minimization of binary decision diagrams based on exchanges of variables. In: Int Conf CAD, pp. 472–475, 1991
78. Jain, J., Bitner, J.B., Abadir, M.S., Abraham, J.A., Fussell, D.: Indexed BDDs: algorithmic advances in techniques to represent and verify Boolean functions. IEEE Trans Comp 46: 1230–1245, 1997
79. Jeong, S.-W., Kim, T.-S., Somenzi, F.: An efficient method for optimal BDD ordering computation. In: Int Conf VLSI CAD, pp. 252–256, 1993
80. Jeong, S.-W., Plessier, B.F., Hachtel, G.D., Somenzi, F.: Variable ordering and selection for FSM traversal. In: Int Conf CAD, pp. 476–479, 1991
81. Kebschull, U., Rosenstiel, W.: Efficient graph-based computation and manipulation of functional decision diagrams. In: Eur Conf Des Autom, pp. 278–282, 1993
82. Kebschull, U., Schubert, E., Rosenstiel, W.: Multilevel logic synthesis based on functional decision diagrams. In: Eur Conf Des Autom, pp. 43–47, 1992
83. Král, D.: Algebraic and uniqueness properties of parity ordered binary decision diagrams and their generalization. In: Symp Math Found Comp Sci. LNCS 1873. Berlin, Heidelberg, New York: Springer-Verlag, 2000, pp. 477–487
84. Krause, M.: Untere Schranken für Berechnungen durch Verzweigungsprogramme. PhD thesis, Humboldt Universität Berlin, 1988. (in German)
85. Krause, M.: Lower bounds for depth-restricted branching programs. Inf Comput 91: 1–14, 1991
86. Krieger, R.: PLATO: a tool for computation of exact signal probabilities. In: VLSI Des Conf, pp. 65–68, 1993
87. Krieger, R., Becker, B., Keim, M.: A hybrid fault simulator for synchronous sequential circuits. In: Int Test Conf, pp. 614–623, 1994
88. Krieger, R., Becker, B., Sinković, R.: A BDD-based algorithm for computation of exact fault detection probabilities. In: Int Symp Fault-Tolerant Comp, pp. 186–195, 1993
89. Kuehlmann, A., Krohm, F.: Equivalence checking using cuts and heaps. In: Des Autom Conf, pp. 263–268, June 1997
90. Kushilevitz, E., Nisan, N.: Communication complexity. Cambridge University, Cambridge, UK, 1997
91. Lai, Y.-T., Pedram, M., Vrudhula, S.B.K.: BDD based decomposition of logic functions with application to FPGA synthesis. In: Des Autom Conf, pp. 642–647, 1993
92. Lai, Y.-T., Pedram, M., Vrudhula, S.B.K.: EVBDD-based algorithms for integer linear programming, spectral transformation, and function decomposition. IEEE Trans CAD 13(8): 959–975, 1994
93. Lai, Y.-T., Sastry, S.: Edge-valued binary decision diagrams for multi-level hierarchical verification. In: Des Autom Conf, pp. 608–613, 1992
94. Lee, C.Y.: Representation of switching circuits by binary decision diagrams. Bell Syst Tech J 38: 985–999, 1959
95. Löbbing, M., Sieling, D., Wegener, I.: Parity OBDDs cannot be handled efficiently enough. Inf Process Lett 67: 163–168, 1998
96. Löbbing, M., Wegener, I.: The number of knight's tours equals 33,439,123,484,294 – counting with binary decision diagrams. Electron J Combin 3: R5, 1996
97. Malik, S., Wang, A.R., Brayton, R.K., Sangiovanni-Vincentelli, A.L.: Logic verification using binary decision diagrams in a logic synthesis environment. In: Int Conf CAD, pp. 6–9, 1988
98. McMillan, K.L.: Symbolic Model Checking. Kluwer Academic, Boston, Mass., USA 1993
99. Meinel, C., Slobodová, A.: On the complexity of constructing optimal ordered binary decision diagrams. In: Symp Math Found Comp Sci. LNCS 841. Berlin, Heidelberg, New York: Springer-Verlag, 1994, pp. 515–524
100. Meinel, C., Slobodová, A.: A unifying theoretical background for some BDD-based data structures. Formal Meth Syst Des 11: 223–237, 1997
101. Meinel, C., Somenzi, F., Theobald, T.: Linear sifting of decision diagrams. In: Des Autom Conf, pp. 202–207, 1997
102. Meinel, C., Theobald, T.: Algorithms and data structures in VLSI design: OBDD - foundations and applications. Berlin, Heidelberg, New York: Springer-Verlag
103. Mercer, M.R., Kapur, R., Ross, D.E.: Functional approaches to generating orderings for efficient symbolic representations. In: Des Autom Conf, pp. 624–627, 1992
104. Minato, S.: Zero-suppressed BDDs for set manipulation in combinatorial problems. In: Des Autom Conf, pp. 272–277, 1993
105. Minato, S.: Binary decision diagrams and applications for VLSI CAD. Kluwer, Boston, Mass., USA, 1996
106. Minato, S.: Fast factorization for implicit cube set representation. IEEE Trans CAD 15: 377–384, 1996
107. Minato, S.: Zero-suppressed BDDs and their applications. Software Tools Technol Transfer, (this issue)
108. Minato, S., Ishiura, N., Yajima, S.: Shared binary decision diagram with attributed edges for efficient Boolean function manipulation. In: Des Autom Conf, pp. 52–57, 1990

109. Mohnke, J., Molitor, P., Malik, S.: Application of BDDs in Boolean matching techniques for formal logic combinatorial verification. Software Tools Technol Transfer, (this issue)
110. Moret, B.M.E.: Decision trees and diagrams. Comput Surv 14: 593–623, 1982
111. Ochi, H., Ishiura, N., Yajima, S.: Breadth-first manipulation of SBDD of Boolean functions for vector processing. In: Des Autom Conf, pp. 413–416, 1991
112. Ochi, H., Yasuoka, K., Yajima, S.: Breadth-first manipulation of very large binary decision diagrams. In: Int Conf CAD, pp. 48–55, 1993
113. Razborov, A.A.: Lower bounds for deterministic and nondeterministic branching programs. In: Fundam Comput Theor. LNCS 529. Berlin, Heidelberg, New York: Springer-Verlag, 1991, pp. 47–60
114. Ross, D.E., Butler, K.M., Kapur, R., Mercer, M.R.: Fast functional evaluation of candidate OBDD variable orderings. In: Eur Conf Des Autom, pp. 4–10, 1991
115. Rudell, R.: Dynamic variable ordering for ordered binary decision diagrams. In: Int Conf CAD, pp. 42–47, 1993
116. Savický, P., Wegener, I.: Efficient algorithms for the transformation between different types of binary decision diagrams. Acta Inform 34: 245–256, 1997
117. Scholl, C., Becker, B., Weis, T.M.: Word-level decision diagrams, WLCDs and division. Technical Report 102, Albert-Ludwigs-University, Freiburg, In: Int. Conf. on Computer Aided Design, pp. 672–677, 1998
118. Scholl, C., Molitor, P.: Efficient ROBDD based computation of common decomposition functions of multioutput Boolean functions. In: Saucier, G., Mignotte, A. (eds.): Novel approaches in logic and architecture synthesis, pp. 57–63. Chapman & Hall, London, 1995
119. Sieling, D.: The nonapproximability of OBDD minimization. Inf Comput (to appear)
120. Sieling, D.: The complexity of minimizing FBDDs. In: Symp Math Found Comp Sci. LNCS 1672. Berlin, Heidelberg, New York: Springer-Verlag, 1999, pp. 251–261
121. Sieling, D.: Lower bounds for linear transformed OBDDs and FBDDs. In: Conf Found Software Technol Theor Comput Sci. LNCS 1738. Berlin, Heidelberg, New York: Springer-Verlag, 1999, pp. 356–368
122. Sieling, D., Wegener, I.: A comparison of free BDDs and transformed BDDs. Formal Meth Syst Des (to appear)
123. Sieling, D., Wegener, I.: *NC*-algorithms for operations on binary decision diagrams. Parallel Process Lett 3(1): 3–12, 1993
124. Sieling, D., Wegener, I.: Reduction of OBDDs in linear time. Inf Process Lett 48(3): 139–144, 1993
125. Sieling, D., Wegener, I.: Graph driven BDDs – a new data structure for Boolean functions Theor Comput Sci 141: 283–310, 1995
126. Somenzi, F.: Efficient manipulation of decision diagrams. Software Tools Technol Transfer, (this issue)
127. Stanion, T., Bhattacharya, D.: TSUNAMI: a path oriented scheme for algebraic test generation. In: Int Symp Fault-Tolerant Comp, pp. 36–43, 1991
128. Tafertshofer, P., Pedram, M.: Factored edge-valued binary decision diagrams. Formal Meth Syst Des 10(2): 243–270, 1997
129. Tani, S., Hamaguchi, K., Yajima, S.: The complexity of the optimal variable ordering problems of shared binary decision diagrams. In: 4th Int Symp Algorithms Comput. LNCS 762. Berlin, Heidelberg, New York: Springer-Verlag, 1993, pp. 389–398
130. Tani, S., Imai, H.: A reordering operation for an ordered binary decision diagram and an extended framework for combinatorics of graphs. In: 5th Int Symp Algorithms Comput. LNCS 834. Berlin, Heidelberg, New York: Springer-Verlag, 1994, pp. 575–583
131. Thathachar, J.S.: On the limitations of ordered representations of functions. In: Comput Aided Verif. LNCS 1427. Berlin, Heidelberg, New York: Springer-Verlag, 1998, pp. 232–243
132. Touati, H.J., Savoj, H., Lin, B.: Implicit state enumeration of finite state machines using BDDs. In: Int Conf CAD, pp. 130–133, 1991
133. Waack, S.: On the descriptive and algorithmic power of parity ordered binary decision diagrams. In: 14th Symp Theor Aspects Comp Sci. LNCS 1200. Berlin, Heidelberg, New York: Springer-Verlag, 1997, pp. 201–212
134. Wegener, I.: The complexity of Boolean functions. Wiley, New York, and B.G. Teubner, Stuttgart, 1987
135. Wegener, I.: Branching programs and binary decision diagrams – theory and applications. SIAM Monogr Discrete Math Appl, 2000
136. Werchner, R., Harich, T., Drechsler, R., Becker, B.: Satisfiability problems for ordered functional decision diagrams. In: Sasao, T., Fujita, M. (eds.): Representations of discrete functions, pp. 233–248. Kluwer Academic, Boston, Mass., USA 1996
137. Wurth, B., Eckl, K., Antreich, K.: Functional multiple-output decomposition: Theory and implicit algorithm. In: Des Autom Conf, pp. 54–59, 1995
138. Yang, B., Chen, Y., Bryant, R., O'Hallaron, D.: Space and time efficient BDD construction via working set control. In: ASP Des Autom Conf, pp. 423–432, 1998