The Theory of Zero-Suppressed BDDs and the Number of Knight's Tours

Martin Löbbing*
FB Informatik, LS II
Univ. Dortmund
44221 Dortmund, Germany
Tel: +49-231-755-2469
Fax: +49-231-755-2047

Olaf Schröer
Philips Dialogue Systems
Weißhausstraße 2
52066 Aachen, Germany
Tel: +49-241-6003-664
Fax: +49-241-6003-601

Ingo Wegener*
FB Informatik, LS II
Univ. Dortmund
44221 Dortmund, Germany
Tel: +49-231-755-2777
Fax: +49-231-755-2047

email: {loebbing,olaf,wegener}@ls2.informatik.uni-dortmund.de

Abstract— Zero–suppressed binary decision diagrams (ZBDDs) have been introduced by Minato in [14]–[17]. Here the structural properties of ZBDDs are worked out and a generic synthesis algorithm is presented and analyzed. It is proved that ZBDDs can be at most by a factor $n+1$ smaller or larger than ordered BDDs (OBDDs) for the same function on n variables. Using ZBDDs the best known bounds on the number of knight's tours on an 8×8 chessboard are improved significantly.

I. Introduction

Bryant [4] has introduced ordered binary decision diagrams (OBDDs) for the representation and manipulation of Boolean functions. OBDDs allow better algorithms in many areas of application like verification, model checking, fault simulation, test pattern generation, timing analysis and many more. Several OBDD variants have been introduced and applied, among them ordered functional decision diagrams (OFDDs) (Kebschull, Schubert and Rosenstiel [10]), graph driven BDDs (Gergov and Meinel [7] and Sieling and Wegener [20]), indexed BDDs (IBDDs) (Jain, Abadir, Bitner, Fussell and Abraham [8]), edge–valued BDDs (EVBDDs) (Lai and Sastry [12]), algebraic decision diagrams (ADDs) (Bahar, Frohm, Gaona, Hachtel, Macii, Pardo and Somenzi [1]), binary moment diagrams (BMDs) (Bryant and Chen [5]), and extended BDDs (XBDDs) (Jeong, Plessier, Hachtel and Somenzi [9]). OFDDs are based on the Reed–Muller expansion while OBDDs are based on the Shannon expansion. EVBDDs, ADDs and BMDs allow integers at the sinks and/or the edges. In graph driven BDDs the variables do not have to be tested on all paths in the same order and in IBDDs the variables can be tested more than once. XBDDs allow some amount of nondeterminism.

But in all these models all constants (Boolean, integers or reals) are treated in the same way. Minato ([14]–[17]) has introduced zero–suppressed BDDs (ZBDDs) for the efficient representation of Boolean functions f, where the inputs a such that $f(a) = 1$ have typically more zeros than ones. In certain situations 0–tests can be suppressed leading to a saving of storage space. Minato has described algorithms for set operations like union, intersection, difference, product and weak–division and successful experiments for cube set representations, fault simulation, timing analysis and the n–queens–problems.

Here the theory of ZBDDs is developed, interesting structural properties of ZBDDs are described, efficient algorithms on ZBDDs, in particular a generic synthesis algorithm, are presented and analyzed, the relation between the OBDD size and the ZBDD size of Boolean functions is determined and a new application to the open problem of counting the knight's tours on 8×8–chessboards is presented.

In Section II OBDDs and ZBDDs are defined and it is shown that the quasi–reduced OBDD for f is isomorphic to the quasi–reduced ZBDD for f with the same variable ordering.

In Section III it is described which subfunctions of f have to be represented on each level of reduced and quasi–reduced ZBDDs for f. Moreover, it is shown how the functions represented at some node and its successors are related. The structural results of Section II and III simplify the discussion of ZBDDs and the analysis of algorithms on ZBDDs.

We believe that ZBDDs might have advantages for certain functions in all types of applications, where OBDDs are used. Therefore, a generic synthesis algorithm for all Boolean operators on ZBDDs is presented and analyzed in Section IV. It turns out that 0–preserving operators like AND, OR and XOR are easier to handle than not 0–preserving operators like NAND, NOR and NXOR. An operator is called 0–preserving if it outputs 0 for the input consisting of zeros only.

In Section V efficient algorithms for the other important operations on BDD like representations, in particular the replacement of variables by constants or functions, are

*Supported in part by DFG grant We 1066/7–3.

presented.

Minato has observed that in certain situations ZBDDs are "considerably smaller" than OBDDs. In Section VI it is proved that ZBDDs can be smaller or larger than OBDDs at most by a factor of $n+1$. Examples show that these bounds are optimal.

Minato [17] used ZBDDs for the computation of the number of solutions of the n–queens–problem. In Section VII the other famous combinatorical chess problem is considered. A knight's tour is a legal closed path for a knight on a chessboard, where each square is visited exactly once. The existence of knight's tours on $n \times n$ chessboards for even $n \geq 6$ is proved (Conrad, Hindrichs, Morsy and Wegener [6]). But the computation of the number of knight's tours on the 8×8 chessboard is still a challenging open problem. The best known upper bound of $3.019 \cdot 10^{22}$ is improved in Section VII with the help of BDDs to $7 \cdot 10^{14}$, and a new lower bound of $5 \cdot 10^{12}$ is presented. To our knowledge this is the first combinatorical result which has been proved for the first time with BDD tools.

II. Zero–Suppressed BDDs and Ordered BDDs

The underlying graph structure (the syntax) of ZBDDs is the same as for OBDDs, but the semantics differ. A BDD graph G is a directed acyclic graph with at most two sinks labeled by the Boolean constants 0 and 1. Each inner node is labeled by one of the Boolean variables $x_1, \ldots, x_n$ and has two outgoing edges one labeled by 0 and the other by 1. The graph has to respect a given variable ordering, we use $x_1, \ldots, x_n$. Then each edge from an x_i–node v (index of v is i or $ind(v) = i$) leads to an x_j–node, where $j > i$, or to a sink whose index is defined as $n+1$.

Let G be a BDD graph. If G is interpreted as an OBDD, the function f_v^O represented at the node v is defined as follows. For the computation of $f_v^O(a)$ start at v. At an inner node labeled x_i choose the outgoing edge with label a_i. Then $f_v^O(a)$ is equal to the label of the sink finally reached. On the other hand, let f_v^Z be the function represented at v, if G is interpreted as a ZBDD. For the computation of $f_v^Z(a)$ the same path is considered as for the computation of $f_v^O(a)$. But $f_v^Z(a)$ equals the label of the sink only if $a_j = 0$ for all j, where x_j is not tested on the considered path. Otherwise $f_v^Z(a) = 0$.

As an example we consider the BDD graph G of Fig.1. It contains three paths from the source to the 1–sink. They are labeled by $(0,0,0)$, $(0,1,1)$ and $(1,?,1)$. Hence, $(0,0,0)$ and $(0,1,1)$ are mapped to 1, if G is an OBDD or a ZBDD. The path $(1,?,1)$ has different meanings for OBDDs and ZBDDs. In OBDDs a non–tested variable is interpreted as don't care, i.e. $(1,0,1)$ and $(1,1,1)$ are mapped to 1, if G is an OBDD. In ZBDDs a non–tested variable is interpreted as 0, i.e. $(1,0,1)$ but not $(1,1,1)$ is mapped to 1, if G is a ZBDD.

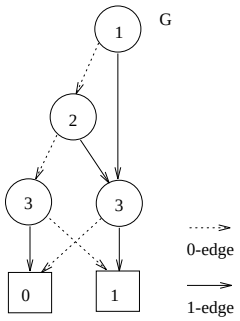

Fig. 1. A BDD graph G, the nodes are labeled with the indices

OBDDs contain for each input a computation path and a path may represent many inputs. In ZBDDs each path ending in the 1–sink represents exactly one input mapped to 1.

The following reduction rules (see Fig.2b–d) do not change the represented function.

Merging rule for OBDDs and ZBDDs: If two nodes have the same label, the same 0–successor and the same 1–successor, they can be merged.

Elimination rule for OBDDs: If the 0–successor of some node v is equal to the 1–successor of v, the node v can be eliminated.

Elimination rule for ZBDDs: If the 1–successor of some node v is the 0–sink, the node v can be eliminated.

Bryant [4] has shown that OBDDs are a canonical form, i.e. the OBDD of minimal size (for f and a fixed variable ordering) is unique (up to isomorphism) and can be obtained from each OBDD for f with the reduction rules. In several papers (e.g. Liaw and Lin [13] and Wegener [22]) also quasi–reduced OBDDs are considered. They are obtained from complete decision trees for f by applying only the merging rule. It is known that quasi–reduced OBDDs are a canonical form.

In complete decision trees each variable is tested on each path from the source to a sink. Hence, the decision tree representing f as an OBDD also represents f as a ZBDD. The property that each variable is tested on each path also holds for the quasi–reduced OBDD. Hence, the quasi–reduced OBDD for f also represents f as a ZBDD and therefore it is also called quasi–reduced ZBDD. We apply this fact often in this paper.

III. Structural Properties of ZBDDs

OBDDs rely on the Shannon expansion of Boolean functions and OFDDs on the Reed–Muller expansion. In both cases a simple relation between the function represented at some node and the functions represented at the direct successors is given in advance. The situation for ZBDDs

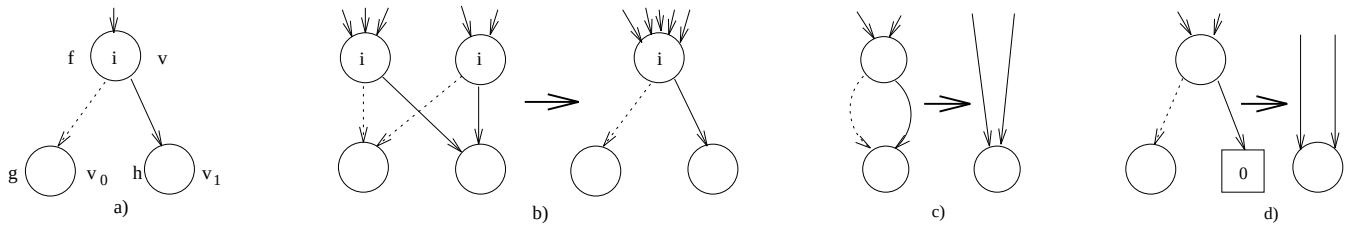

Fig. 2. a) a node and its successors b) merging rule c) elimination rule for OBDDs d) elimination rule for ZBDDs

is different. They are defined by the mode of evaluation (as in Section II) or by the reduction rules (as in Minato [16]). In order to understand the structure of ZBDDs we describe the expansion on which ZBDDs are based (see Fig.2a).

Theorem 1: Let f be represented in a ZBDD at an x_i–node v and let g be represented at the 0–successor v_0 of v and h at the 1–successor v_1 of v. Then the following relations hold.

i) $g = \overline{x}_i f_{|x_i=0}$
ii) $h = \overline{x}_i f_{|x_i=1}$
iii) $f = g + x_i h_{|x_i=0}$

Proof: i) and ii) are proved by case inspection.
iii) follows directly from i) and ii). □

With the next result we describe which functions are represented in quasi–reduced and reduced ZBDDs for f.

Definition 1: A Boolean function f is called 1–simple with respect to x_i if $f_{|x_i=1}$ is equal to the constant 0.

Theorem 2: i) The nodes labeled x_i in the quasi–reduced ZBDD for the Boolean function f represent the different functions $\overline{x}_1 \ldots \overline{x}_{i-1} f_{|x_1=a_1,\ldots,x_{i-1}=a_{i-1}}$, where $a_1, \ldots, a_{i-1} \in \{0,1\}$.

ii) The nodes labeled x_i in the reduced ZBDD for the Boolean function f represent the different functions $\overline{x}_1 \ldots \overline{x}_{i-1} f_{|x_1=a_1,\ldots,x_{i-1}=a_{i-1}}$, where $a_1, \ldots, a_{i-1} \in \{0,1\}$ and $f_{|x_1=a_1,\ldots,x_{i-1}=a_{i-1}}$ is not 1–simple with respect to x_i.

Proof: Let G be the complete decision tree representing f as ZBDD. It follows from Theorem 1 that the node reached for $x_1 = a_1, \ldots, x_{i-1} = a_{i-1}$ represents $\overline{x}_1 \ldots \overline{x}_{i-1} f_{|x_1=a_1,\ldots,x_{i-1}=a_{i-1}}$. The merging rule prevents that the same function is represented by more than one x_i–node. This proves i). The elimination rule deletes all x_i–nodes representing functions which are 1–simple with respect to x_i. This proves ii). □

It follows that a ZBDD can be reduced by a levelwise bottom–up application of the reduction rules, which is possible in linear time $O(|G|)$ using the bucket sort technique of Sieling and Wegener [19].

Liaw and Lin [13] and more precisely Wegener [22] have investigated the OBDD size of almost all Boolean functions. By our considerations in Section II the results on quasi–reduced OBDDs also hold for quasi–reduced ZBDDs. Examining the fraction of nodes which cannot be eliminated, it turns out that all the results about the size of reduced OBDDs contained in [13] and [22] also hold for reduced ZBDDs.

IV. A Generic Synthesis Algorithm

The most important operation on BDD structures is the synthesis. Given a Boolean operator $\otimes$ on m inputs and representations $G_1, \ldots, G_m$ for $f_1, \ldots, f_m$ a representation G for $f = \otimes(f_1, \ldots, f_m)$ has to be computed.

The known OBDD synthesis algorithm works because each node represents functions like $ite(x_i, f_{|x_i=1}, f_{|x_i=0})$ and the ite operator commutes with each operator $\otimes$. This means that $f = \otimes(ite(x_i, f_1|_{x_i=1}, f_1|_{x_i=0}), \ldots, ite(x_i, f_m|_{x_i=1}, f_m|_{x_i=0})) = ite(x_i, \otimes(f_1|_{x_i=1}, \ldots, f_m|_{x_i=1}), \otimes(f_1|_{x_i=0}, \ldots, f_m|_{x_i=0}))$.

This type of algorithm does not work for other BDD structures. The Reed–Muller expansion commutes with XOR but not with AND, OR, NAND or NOR. Becker, Drechsler and Werchner [2] present examples, where the AND–synthesis leads to an exponential blow–up of the OFDD size. The same effect happens for BMDs (Bryant and Chen [5]). Minato ([14]–[17]) investigates some special operators without an analysis of the resulting ZBDD size. We present and analyze a generic synthesis algorithm for ZBDDs.

Definition 2: A Boolean operator $\otimes : \{0,1\}^m \to \{0,1\}$ is 0–preserving if $\otimes(0, \ldots, 0) = 0$.

The following binary operators are 0–preserving: AND, OR, XOR, $\overline{x}_1 x_2$ and $x_1 \overline{x}_2$. The negations of these operators are not 0–preserving. We present a generic ZBDD synthesis algorithm for 0–preserving operators. Later we show how not 0–preserving operators can be replaced by 0–preserving ones.

The ZBDD synthesis algorithm is based on a simultaneous depth–first traversing of $G_1, \ldots, G_m$. The operator is applied to the Boolean constants, if sinks have been reached in all G_i. Otherwise some nodes $v_1, \ldots, v_m$ are reached in $G_1, \ldots, G_m$. In G a node $v = (v_1, \ldots, v_m)$ with label x_i is created where $i = min\{ind(v_1), \ldots, ind(v_m)\} \leq n$. Its 0–successor is computed with a recursive call to

the nodes $v_1^0, \ldots, v_m^0$, where v_j^0 is the 0–successor of v_j, if $ind(j) = i$, and $v_j^0 = v_j$ otherwise. The 1–successor is computed with a recursive call to the nodes $v_1^1, \ldots, v_m^1$, where v_j^1 is the 1–successor of v_j, if $ind(j) = i$, and the 0–sink of G_j otherwise. If G_j does not contain a 0–sink, a dummy 0–sink has to be added. With hashing strategies it is guaranteed that only one node is created for each tuple $(v_1, \ldots, v_m)$. The reduction process can be integrated into the synthesis process.

Theorem 3: The generic synthesis algorithm works correctly for 0–preserving operators. The size of G is bounded by $|G_1^*| \cdot \ldots \cdot |G_m^*|$, where $|G_i^*| = |G_i|$, if G_i contains a 0–sink, and $|G_i^*| = |G_i| + 1$ otherwise.

Proof: The statement about the size of G follows from the fact that G contains only nodes $v = (v_1, \ldots, v_m)$, where v_i is a node of G_i or the dummy 0–sink of G_i.

The correctness is proved by induction. The result is true if $v = (v_1, \ldots, v_m)$ are sinks. The represented functions $f, f_1, \ldots, f_m$ all compute 0, if $(a_1, \ldots, a_m) \neq (0, \ldots, 0)$. For the vector a^0 consisting only of zeros $f(a^0) = \otimes(f_1(a^0), \ldots, f_m(a^0))$.

For the induction step we consider a node $v = (v_1, \ldots, v_m)$ with index i in G. Let $f_1, \ldots, f_m$ be the functions represented at $v_1, \ldots, v_m$ and let f be the function represented at v. We have to prove that $f = \otimes(f_1, \ldots, f_m)$. If $a_k = 1$ for some $k < i$, $f(a) = f_1(a) = \ldots = f_m(a) = 0$. Since $\otimes$ is 0–preserving, the claim follows for such inputs. In the following $a_k = 0$ for all $k < i$.

If the index of v_j is equal to i, the 0–successor of v_j in G_j represents by Theorem 1 $g_j = \overline{x}_i f_{j|x_i=0}$ and the 1–successor represents $h_j = \overline{x}_i f_{j|x_i=1}$. Moreover, $f_j = g_j + x_i h_{j|x_i=0}$ by Theorem 1. If the index of v_j is equal to $k > i$, we apply in G_j the inverse elimination rule and include a new node v_j^* with index i, whose 0–successor is v_j and whose 1–successor is the 0–sink. The node v_j^* represents f_j, its 0–successor represents $g_j = f_j = \overline{x}_i f_{j|x_i=0}$ and its 1–successor represents $h_j = 0 = \overline{x}_i f_{j|x_i=1}$. Again $f_j = g_j + x_i h_{j|x_i=0}$.

The 0–successor of v in G represents by induction hypothesis $\otimes(g_1, \ldots, g_m)$ and the 1–successor represents $\otimes(h_1, \ldots, h_m)$. By Theorem 1 the node v represents $\otimes(g_1, \ldots, g_m) + x_i \left(\otimes(h_1, \ldots, h_m)\right)_{|x_i=0}$. We have to prove that this function equals $\otimes(f_1, \ldots, f_m)$.

$$\begin{aligned}
\otimes(g_1, \ldots, g_m) &= \otimes\left(\overline{x}_i f_{1|x_i=0}, \ldots, \overline{x}_i f_{m|x_i=0}\right) \\
&\overset{(*)}{=} \overline{x}_i \left(\otimes\left(f_{1|x_i=0}, \ldots, f_{m|x_i=0}\right)\right) \\
&\overset{(**)}{=} \overline{x}_i \left(\otimes\left(f_1, \ldots, f_m\right)\right)_{|x_i=0}.
\end{aligned}$$

The equality $(*)$ holds obviously, if $x_i = 0$. If $x_i = 1$, it holds, since $\otimes$ is 0–preserving. The equality $(**)$ holds, since the replacement by constants commutes with each operator.

$$\begin{aligned}
& x_i \left(\otimes(h_1, \ldots, h_m)\right)_{|x_i=0} \\
&= x_i \left(\otimes\left(\overline{x}_i f_{1|x_i=1}, \ldots, \overline{x}_i f_{m|x_i=1}\right)\right)_{|x_i=0} \\
&= x_i \left(\otimes\left(f_{1|x_i=1}, \ldots, f_{m|x_i=1}\right)\right) \\
&= x_i \left(\otimes\left(f_1, \ldots, f_m\right)\right)_{|x_i=1}.
\end{aligned}$$

The equalities hold, since the replacement of variables commutes with each operator and since $f_{j|x_i=1}$ is independent of x_i.

$$\begin{aligned}
& \otimes(g_1, \ldots, g_m) + x_i \left(\otimes(h_1, \ldots, h_m)\right)_{|x_i=0} \\
&= \overline{x}_i \left(\otimes(f_1, \ldots, f_m)\right)_{|x_i=0} + x_i \left(\otimes(f_1, \ldots, f_m)\right)_{|x_i=1} \\
&= \otimes(f_1, \ldots, f_m).
\end{aligned}$$

The Shannon expansion theorem completes the proof. □

The time analysis of the algorithm is easy. Without the time for the operations on the tables it runs in time proportional to the number of computed $v = (v_1, \ldots, v_m)$. Using AVL trees as tables the extra factor can be bounded by a logarithmic factor. In practice one uses hashing strategies and hopes that the extra factor is a constant. If the reduction is integrated into the synthesis, the storage space can be bounded by the sum of the input size and the size of the "computed table".

For a not 0–preserving operator $\otimes : \{0,1\}^m \to \{0,1\}$ let $\tilde{\otimes} : \{0,1\}^{m+1} \to \{0,1\}$ be defined by

$$\tilde{\otimes}(a_1, \ldots, a_m, a_{m+1}) := \overline{\otimes(a_1, \ldots, a_m)} \oplus a_{m+1},$$

where $\oplus$ stands for XOR. Then $\tilde{\otimes}$ is by definition 0–preserving. Moreover,

$$\tilde{\otimes}(a_1, \ldots, a_m, 1) = \otimes(a_1, \ldots, a_m).$$

Hence, the $\otimes$–synthesis of $G_1, \ldots, G_m$ can be replaced by the $\tilde{\otimes}$–synthesis of $G_1, \ldots, G_m$ and G_{one}, the ZBDD representing the constant function 1 (see Fig.3). This leads to the following theorem.

Theorem 4: The generic synthesis algorithm can be used also for the synthesis for not 0–preserving operators (if modified in the appropriate way). The size of the resulting ZBDD G is bounded by $|G_1^*| \cdot \ldots \cdot |G_m^*| \cdot (n+1)$.

Negation is a unary not 0–preserving operator. Hence, Theorem 4 implies that negation may increase the ZBDD size at most by a factor of $n + 1$. An example shown in Fig.3 proves that this bound is best possible. The ZBDD G_1 represents $g = \overline{x}_1 \overline{x}_2 \ldots \overline{x}_n$ with one node, and $|G_1^*| = 2$. The reduced ZBDD for $\overline{g} = x_1 + \ldots + x_n$ contains $|G_1^*|\,(n+1) - 1 = 2n + 1$ nodes.

V. Further Algorithms

For many other problems it is easy to see how they can be solved efficiently. Minato [16] shows that the number of satisfying inputs is equal to the number of paths from the source to the 1–sink and can be computed in time $O(|G|)$.

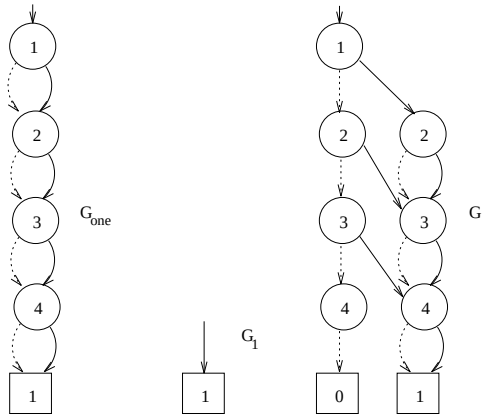

Fig. 3. ZBDD G_{one} represents the constant 1, G_1 represents $g = \overline{x_1}\,\overline{x_2}\,\overline{x_3}\,\overline{x_4}$, G represents $\overline{g} = x_1 + x_2 + x_3 + x_4$

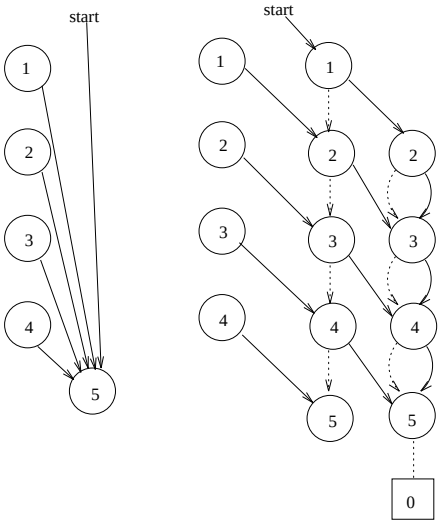

Fig. 4. Construction of quasi–reduced ZBDDs from reduced ZBDDs, the replacement of all possible edge-types to a node with index 5.

The equivalence test for ZBDDs is easy, since reduced ZBDDs are a canonical form. If the ZBDDs may share nodes (like in SBDDs, see Minato, Ishiura and Yajima [18]), a simple pointer comparison is sufficient.

The first problem which is more difficult than for OBDDs is the replacement of variables by constants.

Theorem 5: Let G be the reduced ZBDD representing f and let G_c be the reduced ZBDD representing $f_{|x_i=c}$. Then G_c can be computed in time $O(|G|)$. Moreover, $|G_1| \leq |G^*|$ and $|G_0| \leq |G^*| + \lceil |G|/2 \rceil$.

Proof: First we consider quasi–reduced ZBDDs G. Then it is sufficient to replace the $\overline{c}$–edges leaving x_i–nodes v by edges to the c–successor of v. This is correct, since this procedure works for OBDDs and quasi–reduced OBDDs are quasi–reduced ZBDDs for the same function.

In the general case we first ensure that each path from the source to a sink contains an x_i–node. For this purpose we consider the edge to the source as an edge starting from a node with index 0. Let w be a node with index $j > i$ which is reached directly by an edge from some node v with index $k < i$. Then a new node $\tilde{w}$ is created. Its index is i, its 0–successor is w and its 1–successor is the 0–sink (which perhaps has to be created). All edges leading to w from nodes with an index less than i are redirected to lead to $\tilde{w}$. Because of the ZBDD elimination rule we have not changed the function represented at the source.

Now the replacement technique for quasi–reduced ZBDDs can be applied. If $c = 1$, all new nodes besides the 0–sink can be eliminated again. Hence, $|G_1| \leq |G^*|$.

If $c = 0$, the number of new nodes can be bounded in the following way. Let t be the number of nodes w in G, where $ind(w) > i$. Then by construction $|G_0| \leq |G^*| + t$. The number of new nodes is bounded also by the number of edges leading from some node v, where $ind(v) < i$, to some node w, where $ind(w) > i$. Let s be the number of nodes, where $ind(v) < i$. There are $2s$ edges leaving these nodes. At least $s - 1$ of these edges reach other nodes v', where $ind(v') < i$. Hence, the number of additional nodes is also bounded by $s+1$. Moreover, $s+t \leq |G|$. The upper bound $\min\{s+1, t\}$ is bounded by $\lceil |G|/2 \rceil$. This leads to the upper bound on $|G_0|$.

The so–constructed ZBDDs are not necessarily reduced but they can be reduced in linear time. □

The operation replacement by a constant is quite important, see e. g. the application of ZBDDs in Section VII. Also the quantification of variables is based on the replacement by constants. The same holds for the replacement of variables by functions. The following result follows from Theorem 3 and Theorem 5.

Corollary 1: The operation replacement by functions can be performed for ZBDDs in time $O(|G_g|^2 |G_h|)$ and the size of the resulting ZBDD G_f is bounded by $\frac{3}{2} |G_h^*| \left|G_g^*\right|^2$.

VI. Comparing the Size of OBDDs and ZBDDs

Minato ([14]–[17]) has shown in many applications that ZBDDs can be considerably smaller than reduced OBDDs for the same function (and variable ordering). But how much can we gain or loose? Becker, Drechsler and Werchner [2] have shown that the size of OBDDs and OFDDs for a function may differ exponentially (in both directions). We prove that the reduced ZBDD for f may be at most by a factor of $n+1$ smaller than the reduced OBDD for f and vice versa. The main idea is to construct the quasi–reduced ZBDD (OBDD) from the reduced ZBDD (OBDD). This quasi–reduced ZBDD (OBDD) is also the quasi–reduced OBDD (ZBDD), see Section 2, and the reduced OBDD (ZBDD) is not larger than the quasi–reduced one.

Theorem 6: Let G^Z be the reduced ZBDD representing f and let G^O be the reduced OBDD representing f. Then

$$|G^O| \leq (n+1)|G^Z| + 1.$$

Proof: From G^Z we construct the quasi–reduced ZBDD representing f (see Fig.4). We add a 0–sink, if not existent. For each node v with $ind(v) = j$ we add $j-1$ nodes $v_1, \ldots, v_{j-1}$ whose indices are $1, \ldots, j-1$. The 0–edge leaving v_k leads to v_{k+1}, where $v_j := v$. The 1–edge leaving v_k leads to the node with index $k+1$ created for the 0–sink. Finally, an edge from w with index $ind(w) = i < j-1$ to node v is replaced by an edge to v_{i+1}. The edge to the source is considered as an edge from a node with index 0.

For each of the $|G^Z|$ nodes of G^Z and perhaps for the new 0–sink we have created at most n new nodes. This quasi–reduced ZBDD is also the quasi–reduced OBDD for f. The reduced OBDD is obtained with the OBDD reduction rules. At least the new nodes created for the 0–sink can be eliminated (see Fig.4). □

The upper bound of Theorem 6 is optimal for the ZBDD consisting of the 1–sink only. This ZBDD represents $\overline{x_1}\,\overline{x_2}\ldots\overline{x_n}$ and the reduced OBDD for this function contains $n+2$ nodes. In most cases the upper bound can be improved, since the number of new nodes created for a node with index j which is reached directly only from nodes with index $k \geq i$ can be bounded by $j-i-1$.

Theorem 7: Let G^O be the reduced OBDD representing f and let G^Z be the reduced ZBDD representing f. Then

$$|G^Z| \leq (n+1)|G^O|.$$

Proof: The proof is quite similar to the proof of Theorem 6. The difference is that it is not necessary to create a 0–sink and that both edges leaving v_k lead to v_{k+1}. □

The upper bound of Theorem 7 is optimal for the OBDD consisting of the 1–sink only, since the reduced ZBDD for the constant 1 contains $n+1$ nodes.

One might think that it is possible to save a factor $\Theta(n)$ of the nodes only if the given reduced OBDD or ZBDD has at most linear size. But examples like a multiplexer circuit show that functions with OBDD size $\Theta(n)$ have ZBDDs of size $\Theta(n^2)$ and vice versa.

VII. On the Number of Knight's Tours

In this section we discuss an application of our ZBDD theory. The two most famous combinatorical chess problems are the n-queens problem and the knight's tour problem. Minato [17] has solved the n-queens problem up to $n = 13$ using ZBDDs and a coding of the problem in the unate cube set algebra. His approach contains many nice ideas, but the obtained results are not new. The n-queens problem can be solved easily up to $n = 16$ with a backtracking algorithm. This seems to be difficult with Minato's approach, since the size of the ZBDDs grows "almost proportional with the number of solutions", so one can expect more than $40 \cdot 10^6$ nodes.

We consider the knight's tour problem on $n \times n$ chessboards. The undirected knight's graph contains n^2 vertices representing the squares of the chessboard and the edges describe the legal moves of a knight. A knight's tour is a closed path visiting each square of the chessboard exactly once, i.e. it is a undirected Hamiltonian circuits on the knight's graph. Knight's tours are a challenging problem considered by famous mathematicians like Euler, Legendre and Vandermonde during the last 400 years. It is known that they exist if and only if $n \geq 6$ and n is even (Conrad, Hindrichs, Morsy and Wegener [6]). Is is easy to enumerate the 9,862 knight's tours on 6×6 chessboards. But it is still a challenging problem to determine or estimate the number of knight's tours on 8×8 chessboards. It is known that more than 10^9 tours exist and one expects that the number of knight's tours is much larger. Backtracking algorithms could not increase the lower bound much, since the number of deadlocks is too large. The previously best known upper bound of size $3.019 \cdot 10^{22}$ is due to Kyek, Parberry and Wegener [11].

Our direct attempts to determine the number of knight's tours on 8×8 chessboards with BDD representations failed because of lack of storage space. A well-known relaxation for the traveling salesman problem (also a Hamiltonian circuit problem) is the assignment problem for directed graphs. A solution of the assignment problem is a directed subgraph, where each vertex has outdegree 1 and indegree 1. Such a graph is a cycle covering, since it consists of disjoint directed cycles covering all vertices. In the following we investigate the directed version of the knight's tour graph, where each undirected edge is replaced by two directed edges.

Theorem 8: The number of cycle coverings of the directed knight's graph for 8×8 chessboards is equal to α^2, where $\alpha = 2{,}849{,}759{,}680$, i. e. $\alpha^2 \approx 8.1 \cdot 10^{18}$.

Proof: The knight's graph is a bipartite graph, since the knight can move only from a white square to a black one and vice versa. Each cycle covering can be partitioned into $n^2/2 = 32$ vertex disjoint edges starting at the 32 white squares and reaching the 32 black squares and 32 vertex disjoint edges from the black to the white squares. Let α be the number of sets of 32 vertex disjoint edges from the white to the black squares. Because of the symmetry of the knight's graph α is also the number of sets of 32 vertex disjoint edges from the black to the white squares. Since each combination leads to a cycle covering, we conclude that α^2 is the number of cycle coverings.

Our task is to determine α. Each white square of the knight's graph has 2, 3, 4, 6 or 8 direct successors. We describe the chosen successor with 1, 2 or 3 Boolean variables. Altogether 78 Boolean variables are sufficient. We design a circuit which decides whether the variables de-

scribe 32 vertex disjoint edges from the white to the black squares. The circuit tests whether for each black square w there is one possible predecessor v such the variables belonging to v describe the edge to w. The circuit is then transformed with the generic synthesis algorithm into a ZBDD. The chosen variable ordering tests the variables for each square blockwise and the white squares are ordered rowwise from left to right. The resulting ZBDD has 406,660 nodes, the OBDD 598,472, but the number of nodes generated during the synthesis was nearly the same. This is the reason why some of the following results were obtained using OBDDs. The Sat–Count algorithm finally computes the number of satisfying inputs. □

The computation of α took 6.5 minutes (SUN 670/140, 128MB) with ZBDDs, while a backtracking algorithm took more than 30 days. Theorem 8, the determination of the number of cycle coverings for the 8×8 chessboard, seems to be the first combinatorical result which first has been obtained with BDD like representations and where all other known techniques are much slower.

A knight's tour is a cycle covering consisting of exactly one undirected cycle. Each undirected knight's tour leads to two directed cycle coverings, so we may fix for one corner how we pass the corner. We take the upper right corner, a black square. Most of the cycle coverings contain more than one cycle, so we improve the bound by excluding all cycles of length 2 containing one of the four squares in one of the corners. These squares correspond to vertices in the knight's graph with lowest degree and have the largest probability to lie on a cycle of length 2. Then we distinguish the 2^{23} possibilities how we pass through the 15 squares, 8 of them are white and 7 black.

If a possibility is fixed, we compute the number of cycle coverings with moves starting at white squares, containing the fixed moves for the 8 white squares and not containing the moves which would result in a cycle of length 2 in combination with a move from one of the 7 black squares. This number is multiplied by the number of cycle coverings for black squares computed in the same manner. To fix the edge (v, w) is equivalent to replacing the variable representing the edge starting at v by that constant which indicates that the edge leads to w. At the end, we sum up all these possibilities, which leads to the following result:

Theorem 9: The number of undirected knight's tours on 8×8 chessboards is bounded above by 697,825,558,035,068.

Using this technique, the bound cannot be improved much more. The number of possible moves on the 16 chosen squares is already so large that it took days to compute the sum of all cycle coverings.

We are also interested in improving the lower bound. All known backtracking algorithms suffer from the fact that they run deep into deadlocks, in which they spend much time without finding a solution. There are heuristics

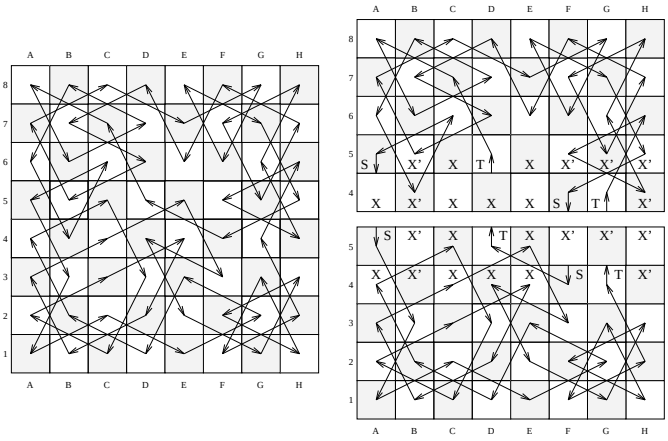

Fig. 5. An example of a knight's tour

like the one from Warnsdorff [21] which can produce many solutions quickly, but they need as much time as the naive algorithm when all "simple" solutions are found.

So we tried to divide the problem. We take an 8×5 board, in the following called a "half board". Two of these half boards result in an 8×8 board, if they overlap in the rows 4 and 5. We partition the squares in three sets, one for the lower half board (L), one for the upper (U) and one for the middle (M). A knight's tour can only switch from one half board to the other, if a square in the middle is visited. For a fixed knight's tour M is partitioned into four sets: Let S be the set of squares where the knight enters the lower half board and T the set where it leaves it. Moreover, let X (resp. X') be the set of squares where the knight remains on the lower (resp. upper) half board. As an example, the left side of Fig.5 shows a random knight's tour. On the right side the partition of this tour in two half boards is presented. The squares in the overlapping area are marked with the sets they belong to.

In a natural way, each tour is divided into $2k$ ($1 \leq k \leq 8$) segments, each lying completely on one half board. We look at one half board, say the lower one. The knight's tour visiting all squares in $L \cup S \cup T \cup X$ exactly once defines a bijective function $f : S \to T$. Our task will be to compute the number $c(f)$ of tours representing the same function f, which are different on the lower half board.

The tour, we looked at, defines another bijective function f' on the upper half board. Here the board is entered at a square from T and left at a square from S. Each square in $U \cup T \cup S \cup X'$ with $X' = M \setminus (S \cup T \cup X)$ is visited and the function described is $f' : T \to S$.

The composition of these two functions $f' \circ f : S \to S$ represents a permutation over S. A tour visiting the squares in S in the order $s_{i_1}, s_{i_2}, \ldots, s_{i_k}$ describes always the permutation $(i_1, i_2, \ldots, i_k)$, regardless of the path between these squares. Notice that all permutations described by knight's tours consist only of one cycle of length k in the sense that the original order is obtained the first

time only after a k–times repetition of the permutation. We say that in this case $f' \circ f$ builds one cycle. A permutation with two or more cycles represents cycle coverings of the complete board which also have two or more cycles.

To compute the number of knight's tours, we proceed as follows. We fix the sets S, T and X and sum up the tours over all possible combinations. Analogously to the backtracking algorithm we take two OBDDs, one representing the cycle covering for white squares of a half board, the other for the black squares. There are 40 (non Boolean) variables for the 8×5 squares of a half board. For the squares in the overlapping area (rows 4 and 5, set M), there is an extra coding which describes the possibility that the knight leaves this square to the other half board (square belongs to T or X). Additionally there is one Boolean variable introduced for each square in M, which describes that the knight enters this square from the other half board; the square belongs to S or X. It is easy to compute OBDDs respecting the given restrictions, we only have to substitute some variables by constants. The backtracking algorithm keeps track of the path segments connecting S and T. This leads to the function f described before, but we only count the possibilities for each function $c(f)$. Analogously to this procedure, we count the possibilities for each function f' on the other half board. The only remaining task is to check, which combinations of f and f' lead to one cycle, representing a knight's tour, and which lead to more cycles, representing only a cycle covering. A legal combination of f and f' then stands for $c(f) \cdot c(f')$ different tours, since each cycle covering for f can be combined with each for f', always resulting in a tour. We sum up these numbers over all possible sets, resulting in the number of tours on an 8×8 board. We ran the algorithm on some computers and in 4 weeks they completed about a quarter of the search space. This led us to the following lower bound, which is not too far from the upper bound of approx. $7 \cdot 10^{14}$.

Theorem 10: The number of undirected knight's tours on 8×8 chessboards is bounded below by $5 \cdot 10^{12}$.

The precise result will be available soon.

References

[1] R. I. Bahar, E. A. Frohm, C. M. Gaona, G. D. Hachtel, E. Macii, A. Pardo and F. Somenzi: Algebraic decision diagrams and their applications. ICCAD, pp. 188–191, 1993.

[2] B. Becker, R. Drechsler, and R. Werchner: On the relation between BDDs and FDDs. LATIN '95, to appear in LNCS, April 1995.

[3] K. S. Brace, R. L. Rudell and R. E. Bryant: Efficient implementation of a BDD package. 27th DAC, pp. 40–45, 1990.

[4] R. E. Bryant: Graph-based algorithms for Boolean function manipulation. IEEE Trans. on Computers 35, pp. 677–691, 1986.

[5] R. E. Bryant and Y.-A. Chen: Verification of arithmetic functions with binary moment diagrams. 32nd DAG, 1995.

[6] A. Conrad, T. Hindrichs, H. Morsy, and I. Wegener: Solution of the knight's Hamiltonian path problem on chessboards. Discrete Applied Mathematics 50, pp. 125–134, 1994.

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

[8] J. Jain, M. Abadir, J. Bitner, D. S. Fussell and J. A. Abraham: IBDDs: An efficient functional representation for digital circuits. EDAC, pp. 440–446, 1992.

[9] S.-W. Jeong, B. Plessier, G. Hachtel and F. Somenzi: Extended BDDs: Trading off canonicity for structure in verification algorithms. ICCAD, pp. 464–467, 1991.

[10] U. Kebschull, E. Schubert and W. Rosenstiel: Multilevel logic synthesis based on functional decision diagrams. EDAC, pp. 43–47, 1992.

[11] O. Kyek, I. Parberry and I. Wegener: Bounds on the number of knight's tours. Tech. Rep. 555, Univ. Dortmund, 1994

[12] Y.-T. Lai and S. Sastry: Edge-valued binary decision diagrams for multi-level hierachical verification. 29th DAC, pp. 608–613, 1992.

[13] H.-T. Liaw and C.-S. Lin: On the OBDD representation of general Boolean functions. IEEE Trans. on Computers 41, pp. 661–664, 1992.

[14] S. Minato: Fast generation of irredundant sum-of-products forms from binary decision diagrams. Proc. Synthesis and Simulation Meeting and International Interchange SASIMI, pp. 64–73, 1992.

[15] S. Minato: Fast weak-division method for implicit cube representation. Proc. Synthesis and Simulation Meeting and International Interchange SASIMI, pp. 423–432, 1993.

[16] S. Minato: Zero-suppressed BDDs for set manipulation in combinatorial problems. 30th DAC, pp. 272–277, 1993.

[17] S. Minato: Calculation of unate cube set algebra using zero-suppressed BDDs. 31st DAC, pp. 420–424, 1994.

[18] S. Minato, N. Ishiura and S. Yajima: Shared binary decision diagrams with attributed edges for efficient Boolean function manipulation. 27th DAC, pp. 52–57, 1990.

[19] D. Sieling and I. Wegener: Reduction of OBDDs in linear time. Information Processing Letters 48, pp. 139–144, 1993.

[20] D. Sieling and I. Wegener: Graph driven BDDs—a new data structure for Boolean functions. Theoretical Computer Science 143, 1995.

[21] H. C. Warnsdorff: Des Rösselsprunges einfachste und allgemeinste Lösung, Schmalkalden, 1823.

[22] I. Wegener: The size of reduced OBDDs and optimal read-once branching programs for almost all Boolean functions. IEEE Trans. on Computers 43(11), pp. 1262–1269, 1994.