Boolean Manipulation with Free BDD's. First Experimental Results. *

Jochen Bern   Jordan Gergov   Christoph Meinel   Anna Slobodová

FB IV – Informatik, Universität Trier,
D 54 286 Trier, Germany

Abstract

It is shown that Free Binary Decision Diagrams (FBDD's), with respect to a predefined type,

- *provide a canonical representation and*
- *allow efficient solutions of the basic tasks in Boolean manipulation*

similarly as the well-known OBDD's do. But in contrast to OBDD's, typed FBDD's allow more succinct representations of Boolean functions.

For experimentation we have used an FBDD–package. The types we work with are tree–based. Due to different type–creating heuristics we compare the size of FBDD–representations of some ISCAS benchmarks with the size of their OBDD–representations.

1 Introduction

The need for efficient data structures for Boolean functions becomes obvious if one has applications in mind as computer aided circuit design. In order to support algorithms for synthesis, verification, or testing circuits, one needs schemes for representing Boolean functions that are as succinct as possible and at the same time support efficient solutions of the basic tasks in Boolean manipulation, especially

- efficient algorithms for deriving the representations of the involved Boolean functions (e.g. from a circuit description) and
- efficient algorithms to perform equivalence test or similar tests such as satisfiability or tautology test in terms of these representations.

In order to come to a full understanding of the fundamental trade–off between succinctness of the representation on the one hand and efficiency of solving the basic tasks on the other hand, observe that the complexity of the algorithms has to be measured in the length of the representations of the input functions. Hence, working, e.g., with truth–tables one has much time for computations. However, such a representation requires in any case space resources that are exponential in the number of primary inputs. On the other side, working with, e.g., formulas often one obtains very succinct and, consequently, space efficient representations but solving, e.g., the equivalence test becomes NP–hard.

A systematic inspection of the different succinct representation schemes [e.g., GJ79, GM92] has shown that most of the Boolean function representations investigated in the past do *not* support efficient solutions of the Boolean manipulation basic tasks. Even worse, performing the basic tasks often requires the solution of NP–hard problems. The only exception seems to be BDD–based Boolean function representation [e.g. Ake78, Bry86, Mei91, GM92] that provide with so–called OBDD's the state–of–the–art data structure for Boolean functions. OBDD's allow

- canonical representations (and, hence, efficient solutions of the equivalence test and similar tests such as satisfiability or tautology test), and
- efficient execution of binary Boolean synthesis steps and, hence, efficient procedures for deriving the OBDD–representation of a Boolean function from a given circuit description and/or formula.

Due to these nice properties OBDD's are successfully used in many applications such as, e.g., sequential circuit verification [e.g. BCMD90, CMB90], testing [e.g. ADK91, Bec92, KBS93], or logic optimization [Kar89]. For an excellent survey see [Bry92].

Unfortunately, OBDD–representations are not very succinct and, hence, not very space–efficient. The question arises whether there are more sophisticated BDD–representations that are, first, *more succinct* and space–efficient as OBDD's, and, second, *allow efficient solutions* of the basic tasks of Boolean manipulation similarly as OBDD's do. In the past, a number of OBDD–extensions were proposed with the aim

*Proc. of *European Design and Test Conference1994*,
IEEE Computer Society Press, 200–207, 1994.

to overcome the mentioned disadvantages of OBDD's [e.g. ADG91, JPHS91, BJAAF92]. However, the obtained increase in space–efficiency of the representation was paid with a considerable loss in efficiency of the basic tasks. (In the above mentioned approaches the equivalence test becomes co–NP–complete.) In [GM92b] it was proposed to work with FBDD's, i.e., with an OBDD–extension that insists on the restriction that each variable is tested at most once on each path but removes the ordering restriction of OBDD's. It could be proved that, with respect to a predefined type, FBDD's

- provide a canonical representation and
- allow efficient solutions of the basic tasks in Boolean manipulation

similarly as OBDD's do. In contrast to OBDD's, FBDD's allow more succinct representations of Boolean functions. For example, FHS–function [FHS 78], indirect storage access function [BHR91], and hidden weighted bit function can be represented by quadratic size FBDD's while OBDD–representations of these functions are of exponential size.

In the following we describe this FBDD–based approach in more detail and report some preliminary experimental results. These results could be obtained by means of an FBDD–package [SM93] which was derived from the seminal OBDD–package of Brace, Rudell and Bryant [BRB90]. The types we work with are tree–based. Due to different type–creating heuristics we compare the size of FBDD–representations of some ISCAS85 and ISCAS89 benchmarks with the size of their OBDD–representations obtained by using the OBDD–heuristic presented in [MWBS88]. The results of these experiments prove the practical usefulness of our FBDD–based approach. However, more research is necessary to reveal its full potential.

This paper is organized as follows. After giving some preliminaries in Section 2, in Section 3, the efficient Boolean manipulation with FBDD's developed in [GM92b] is described. Section 4 gives a short review of the FBDD–package developed and used in Trier. Results obtained by applying this package to some ISCAS85 and ISCAS89 benchmarks are presented in the final Section 5. These results are obtained by means of some tree–based type–creating heuristics described in this section, too.

2 Preliminaries

BDD-based data structures for Boolean functions use the following representation scheme.

Definition. (BDD, OBDD, FBDD)
A **binary decision diagram (BDD)** *over a set $X_n = \{x_1, \ldots, x_n\}$ of Boolean variables is a directed acyclic graph with one source and at most two sinks labeled 0 and 1. Each non-sink node v is labeled with a Boolean variable $l(v) \in X_n$ and has two outgoing edges, one labeled with 0 and the other with 1. The then–son of v is reached via the 1–edge, the else–son is reached via the 0–edge. The* **computation path** *for an input $a = (a_1, \ldots, a_n)$ starts at the source. At an inner node with label x_i, the outgoing edge with label a_i is chosen. The BDD P* **represents** *a Boolean function $f \in \mathbb{B}_n$ if the computation path for each input a leads to the sink labeled $f(a)$. f is sometimes denoted by f_P.*

A BDD is called **free binary decision diagram** *(FBDD) if, on each path, each variable is tested at most once. An* **ordered binary decision diagram** *(OBDD) is an FBDD with the property that on each path the variables are tested in a predefined order.*

Examples of an FBDD and an OBDD are given in Figure 1. (Generally we draw BDD's in such a way that, if the edges are not labeled, we assume the left edge to be labeled 0 and the right edge 1.)

It is well known that each Boolean function $f \in \mathbb{B}_n$ over X_n can be represented in terms of BDD's, in terms of FBDD's or, for any variable ordering, in terms of OBDD's. Optimal BDD–representations are, in comparison with two–level–representations such as DNF's or CNF's or with multi–level–representations such as Boolean formulas, more succinct and space–efficient [Mei89]. However, this succinctness makes it often difficult or sometimes even infeasible to perform basic tasks of Boolean manipulation. For example, it is a co–NP-complete problem to test whether two BDD's represent the same Boolean function [CHS74].

The situation is changed if one works with restricted types of BDD's. Then, sometimes, it is possible to perform efficiently basic tasks of Boolean manipulation although restriction properties have to be maintained in the course of the computation. Bryant [Bry86] was the first who observed that OBDD's have this property. In more detail, OBDD's possess the following outstanding properties.

Fact 1 [Bry86].

1. *With respect to a fixed variable ordering, the representation of a Boolean function by means of a reduced OBDD is canonical, i.e., uniquely determined.*
2. *Let P' and P'' be two OBDD's with the same variable ordering. A synthesis step (i.e. the application of a binary Boolean operator) can be performed in time $O(size(P) \cdot size(P'))$.*

Due to these properties, OBDD's defined over a fixed variable ordering are well suited as data structure for Boolean functions [Bry86, Bry92]. However, the disadvantage of the OBDD data structure is the often very low space efficiency of OBDD–representations. Though wide classes of practically important Boolean functions possess – at least with respect to some well–suited variable ordering – space–efficient (i.e. polynomial size) OBDD–representations, there are many important functions without such succinct representations. For example, there exist Boolean functions such as FHS–function [FHS78], integer multiplication, hidden weighted bit function (HWB) [Bry91], or indirect storage access function (ISA) [BHR91] which can't be represented by OBDD's of polynomial size for any variable ordering.

Fact 2 [Bry91, FHS89, BHR91].
OBDD–representations of FHS–function, integer multiplication, hidden weighted bit function (HWB), and indirect storage access function are, for each variable ordering, of exponential size.

3 Efficient Boolean manipulation with FBDD's

3.1 Computational advantages of FBDD's

The example of the hidden weighted bit function HWB demonstrates the efficiency of FBDD–representations in comparison with OBDD–representations. While in the case of OBDD's (Fact 2) HWB needs, for each variable ordering, exponential size, it can be represented by means of quadratic size FBDD's. To explain this we use a construction proposed by Bryant [Bry92b] which is based on the computation of appropriately chosen restrictions of HWB. If $x_{i:j}$ represents the variables $x_i, \ldots, x_j$, and $wt(x_{i:j}) = x_i + \ldots + x_j$, let

$$H(x_{i:j}) = \begin{cases} 0 & \text{if } wt(x_{i:j}) = 0 \\ x_{i+wt(x_{i:j})-1} & \text{otherwise} \end{cases}$$

$$G(x_{i:j}) = \begin{cases} 1 & \text{if } wt(x_{i:j}) = j-i+1 \\ x_{i+wt(x_{i:j})} & \text{otherwise} \end{cases}$$

One can easily verify that

$$HWB(x_1, x_2, \ldots, x_n) = H(x_{1:n}),$$

$$H(x_{i:j}) = \bar{x_j} H(x_{i:j-1}) + x_j G(x_{i:j-1}),$$

$$G(x_{i:j}) = \bar{x_i} H(x_{i+1:j}) + x_i G(x_{i+1:j}),$$

$$G(x_{i:i}) = H(x_{i:i}) = x_i.$$

The main idea of the FBDD construction is explained in Figure 2. $H_{1:n}$ and $G_{1:n}$ are computed simultaneously. Identical restrictions on the same level are merged together. Since, on each level, k is fixed, we have at most $2n$ nodes on each level. Since on any source–to–sink–path each variable is tested exactly once, we get

Fact 3.
The FBDD–representation P_{HWB} of $HWB \in \mathbb{B}_n$ is of size at most $2n^2$.

Similar results could be obtained for the FHS–function and the indirect storage access function.

3.2 FBDD types

In order to be able to work with FBDD's as efficiently as with OBDD's, in [GM92b] the notion of a *type* of an FBDD was introduced which generalizes the linear variable ordering of OBDD's.

Definition. (Type, tp(P), complete type)
*An (FBDD–)***type** *is defined similarly as an FBDD with the only exception that it possesses merely one sink, which is labeled with a symbol t.*
A type τ is **complete** *over $X_n = \{x_1, x_2, \ldots, x_n\}$ if, on each source-to-sink path, each variable of X_n is tested.*
If P is an FBDD then $tp(P)$ is the type derived from P by identifying the sinks of P.

Of course, linear orderings provide types with a very simple structure, so–called OBDD–types. A more interesting example is given in Figure 3. For $n = 7$, it shows the type $\tau_{HWB} = tp(P_{HWB})$ where P_{HWB} is the FBDD for HWB given in Figure 2. (τ_{HWB} roots in the source labeled x_7.) Since, on each source–to–sink path, each variable is tested, τ_{HWB} is complete.

OBDD's, FBDD's, and types can be reduced by means of the two following syntactical rules. Note that these reductions change neither the functional behavior of an OBDD or FBDD nor substantially the way an OBDD, FBDD, or type tests its input variables.

Merging rule:
If two nonterminal nodes u and v have the same label $l(u) = l(v)$, and if their then–sons as well as their else–sons coincide, then eliminate one of these two nodes and redirect all incoming edges to the other node. Considering BDD's as an algebraical structure, the application of the merging rule defines a congruence relation on the set of nodes of the BDD. For this reason, we sometimes call an application of the merging rule an *algebraical reduction*.

Deletion rule:
If then–son and else–son of a nonterminal node v coin-

cide, then eliminate v and redirect all incoming edges to its son. Opposite to the merging rule, the deletion rule decreases the information contained in a type. Sometimes we call an application of the deletion rule a *simple reduction*.

An FBDD P is said to be *reduced* if neither the merging rule nor the deletion rule can be applied to P. A type τ or an FBDD P is said to be *algebraically reduced* if the merging rule can't be applied to τ or to P. For example, the FBDD P_{HWB} for the HWB given in Figure 2 is reduced, and the type τ_{HWB} given in Figure 3 is algebraically reduced.

Interestingly, the reduced FBDD $red(P)$ obtained from FBDD P by applying merging and deletion rules as long as possible is uniquely determined and can be computed efficiently. Similarly, the algebraically reduced FBDD $red_a(P)$ and the algebraically reduced type $red_a(\tau)$ obtained from FBDD P and type τ, respectively, by applying the merging rule as long as possible are uniquely determined and can be computed efficiently.

Proposition 1. (Unique reduction)
Let P be an FBDD, and let τ be a type. The reduced FBDD $red(P)$ is uniquely determined [GM92b] and can be computed in linear time $O(size(P))$ [SW92]. Similarly, the algebraically reduced FBDD $red_a(P)$ and the algebraically reduced type $red_a\tau$ are uniquely determined and can be computed in linear time.

In order to identify classes of FBDD's which test their input variables in a similar way, it suffices to compare the types of the FBDD's. For two types τ and τ' we write $\tau \leq \tau'$ if τ can be obtained from τ' by means of applicating merging and deletion rules.

Definition. (FBDD of type τ)
An **FBDD** P **is of type** τ *if there exists a type τ' such that $tp(P) \leq \tau'$, and $red_a(\tau') = red_a(\tau)$.*

Sometimes, we write τFBDD to indicate an FBDD to be of a type τ. It is easy to see that if τ is a complete type over a set X_n, each Boolean function $f \in \mathbb{B}_n$ can be represented by means of a τFBDD.

Proposition 2. (Efficient type–check)
Let P be an FBDD, and let τ be a complete type. Then it can be checked efficiently whether P is of type τ.

3.3 Efficient algorithms for the basic tasks of boolean manipulation in terms of FBDD's

3.3.1 Canonical representation

In [GM92b] it was proved that, with respect to a complete type, reduced FBDD's similar as reduced OBDD's provide a canonical representation for Boolean functions.

Theorem 3. (Canonical FBDD representation)
Let τ be a complete type over X_n, and let $f \in \mathbb{B}_n$. There is, up to isomorphisms, exactly one reduced FBDD of type τ which represents f.

As an easy consequence of Theorem 3 and Proposition 1 we obtain an efficient algorithm which solves the equivalence problem $EQU_{\tau FBDD}(P, P')$ for two FBDD's P and P' of type τ.

Corollary 4. (Efficient equivalence test)
$EQU_{\tau FBDD}$, the equivalence of two FBDD's P' and P'' of the same complete type, can be decided in linear time $O(size(P') + size(P''))$.

Let us mention that the best result for solving the general equivalence test for FBDD's EQU_{FBDD} is a probabilistic polynomial time algorithm [BCW80].

3.3.2 Efficient execution of boolean synthesis steps

Next we show how the efficient *apply* procedure for OBDD's of the same variable ordering [Bry86] can be extended to FBDD's of the same type [GM92b, SW92]. Figure 4 gives the pseudo–code for the apply procedure for FBDD's of the same type.

Theorem 5. (Efficient Boolean synthesis)
Let $*$ be a binary Boolean operation, and let P', P'' be FBDD's of the same complete type τ. An FBDD P of type τ representing the function $f = f_{P'} * f_{P''}$ can be constructed in time $O(size(\tau) \cdot size(P') \cdot size(P''))$.

In the case when synthesized FBDD's encode all information of the type we do not need the type at all. As an example, let us mention the FBDD P_{HWB} for the hidden weighted bit function (Figure 2) and its type (Figure 3) described in Section 3: Working with P_{HWB} we do not need the type τ_{HWB}. Hence, for repeated applications of *synthesis* we get a quadratic upper bound for the run time instead of cubic time.

Finally, we note that the straightforward algorithm presented in the proof of Theorem 5 is not optimal. The programming techniques used, e.g., in the OBDD–package developed by Brace, Rudell and Bryant [BRB90] can be applied in the case of FBDD's, too. For instance, with the help of an appropriate hash table, we can force *synthesis* to produce only reduced FBDD's. Hence, no reduction algorithm is needed.

4 FBDD–package

Our implementation of the algorithms for FBDD–based Boolean manipulation is based on the widespread OBDD–package of Brace, Rudell, and Bryant described in [BRB90]. The resulting FBDD–package which provides the same functions as the OBDD–based one is described in detail in [SM93]. It constitutes a truly counterpart to the OBDD–package that makes it possible to measure the effect of using FBDD's instead of OBDD's as data structure for Boolean functions. Its implementation shows practically that FBDD–based Boolean manipulation constitutes not only a theoretical possibility but also a practical alternative at least for such cases were OBDD–based manipulation is too space consuming.
The package provides the following operations:

- creation of variables
- creation of new FBDD formulas by means of Boolean operations like *and, or, not,*
- performing restrict, compose, and quantify operations (with some restrictions on the type)
- freeing unneeded formulas
- performing tests on formulas like equivalence (F≡G), tautology (F≡1), implication (F≤G)

All user's functions are represented as reduced FBDD's which are joined into a single multi–rooted graph (denoted by $\mathcal{RFBDD}$) by sharing all common subgraphs. They are assumed to be of a fixed complete type called $\mathcal{RFBDD}$*–type*, which has to be given at the very beginning (similarly to the variable ordering in the case of OBDD's). In the course of the manipulation of the $\mathcal{RFBDD}$, this condition is automatically maintained, since all operations performed on formulas of the fixed type yield results which are of this type, too.

Each node of the $\mathcal{RFBDD}$ as well as of the type stores the information about the tested variable and its then– and else–son. Complement edges are used in the same way as in [BRB90], i.e., then-edges are always assumed to be regular edges. In order to keep the *canonical form* of $\mathcal{RFBDD}$, all nodes are hashed into a unique table as in [BRB90]. Before a new node is created, we look up the table. In the case of matching, i.e., if a node labeled with the same variable and the same low and high sons already exists, then no new node is created and the existing node is used. The *equivalence test* is reduced, hence, to a table lookup.

A *synthesis step* is performed with the help of the *if-then-else* operator which has an aditional argument, the type, with respect to which the operation has to be performed. (In the case of OBDD's, the role of the type was played by the order.) In more detail, $ite(T,F,G,H)$, where T is a type, F, G, and H are formulas of type T, computes the uniquely determined $FBDD$–representation of type T of the Boolean function *"if F, then G, else H"*. The computation of ite is recursively implemented. The recursion is based on

$$ite(T,F,G,H) = (v, ite(T_v, F_v, G_v, H_v), ite(T_{\overline{v}}, F_{\overline{v}}, G_{\overline{v}}, H_{\overline{v}}))$$

with v the top variable of F, G, H with respect to T.

The exponential behaviour of the procedure is avoided by using the *computed table*, which stores results of recent computations. It is realized (similarly to the OBDD–computed–table) as a hash–based cache initialized only once, namely before the very first call.

The memory usage per $\mathcal{RFBDD}$–node is 22 bytes as in [BRB90]. The type is usually substantially smaller and requires 12 bytes per node.

5 First heuristics and experimental results

In the following section we report some preliminary results which demonstrate the power of our FBDD–based data structure. Of course, the potential of FBDD's fully comes to expression when one works with types that are specially designed for the considered circuit or circuit family as it was done in [GM92b] for the verification of the HWB circuit. However, in many situations one has to derive a type from a given circuit description by means of a heuristic similarly as one has to extract the variable ordering in case of OBDD's.

5.1 Heuristics for type creation

For experiments with heuristics, we implemented a package called TRUST [BMS93] which allows to determine types from circuit netlists and let the FBDD-package compute the functions of the outputs in terms of the inputs. We currently use TRUST heuristics which derive from given circuit netlists easily structured tree–like types. This is done in three steps: First, we choose an input to *split* on and repeat this process for two appropriately reduced circuits recursively up to a certain depth. Since on the different paths of the resulting trees usually different sets of variables are tested, we have to apply a *fill* operation which appends an OBDD-type to every leaf consisting of the missing variables. Finally, a global OBDD-type for the remaining variables is appended.

5.2 Results

Table 1 presents first results obtained in comparing FBDD–sizes and OBDD–sizes for some ISCAS85 and

ISCAS89 benchmark circuits. The OBDD's are constructed with respect to the fan–in heuristic (*fh*) of [MWBS88] or the TFI-oriented heuristic (*th*) used in SIS. The FBDD's are constructed mainly with respect to heuristics which combine the *fh* or *th* heuristic for completion with two split heuristics choosing either the max–level (deep split, *ds*) or the min–level (flat split, *fs*) input from those yielding at least a given minimum total reduction (*mtr*) of the circuits. For example, *fs12-mtr3-fh* means that splitting was done

- for at most 12 levels, choosing the min-level input / variable from those which
- reduce the circuit by 3 % or more (mean of both restricted circuits),

and that we used the *fh* order to complete the type.

Examples of how different parameters given to these heuristics affect the result are given in the tables 2 and 3.

6 Conclusion

We have investigated a new FBDD–based data structure for Boolean manipulation proposed in [GM92b]. The FBDD–package [SM93] we work with was obtained by modifying the OBDD–package of [BRB90]. By means of some type–creating components offered by our programming environment TRUST [BMS93] we have implemented some very easily structured tree–based type creating heuristics such as *fs..mtr..-fh*, and *ds..mtr..-fh* which extend the fan–in heuristic for OBDD's proposed in [MWBS88]. Similarly as in the case of OBDD's, it becomes obvious that different type creating heuristics provide rather different FBDD's.

Altogether the experimental results show clearly that the FBDD–approach is not only of theoretical interest but also of practical usefulness even for such easy structured tree–based types. However, the full power of typed FBDD's comes to expression when one works with types that are specially designed for the considered circuit or circuit family [see, e.g., [GM93]).

Our next goals in investigating typed FBDD's are

- to design good types for arithmetical circuits like, e.g., multipliers.
- to improve and refine the mentioned type creating heuristics, e.g. by taking advantage of techniques like sifting,
- to investigate some new ideas for type–creating heuristics.

Acknowledgement. The authors would like to thank the members of the research group of Bernd Becker at the Johann–Wolfgang–Goethe–Universität Frankfurt for many helpful discussions.

References

[ADG91] P. Ashar, S. Devadas, A. Ghosh: Boolean Satisfiability and Equivalence Checking Using General Binary Decision Diagrams, Proc. Int. Conf. on Computer Design (Cambridge, Mass., Oct.), 259–264, 1991.

[ADK91] P. Ashar, S. Devadas, K. Keutzer: Gate–Delay–Fault Testability Properties of Multiplexor–Based Networks, Proc. of Int. Test Conference, 887–896, 1991.

[Ake78] S. B. Akers: Binary Decision Diagrams, IEEE Trans. Comput. C-27, 6 (Aug.), 509–516, 1978.

[BCMD90b] J. R. Burch, E. M. Clarke, K. L. McMillan, D. L. Dill: Sequential Circuit Verification Using Symbolic Model Checking, Proc. of 27th ACM/IEEE Design Automation Conference (Orlando, June), 46–51, 1990.

[BCW80] M. Blum, A. K. Chandra, M. N. Wegman: Equivalence of Free Boolean Graphs Can Be Decided Probabilistically in Polynomial Time, Inf. Process. Lett. 10, 2 (Mar.), 80–82, 1980.

[Bec92] B. Becker: Synthesis for Testability: Binary Decision Diagrams, Proc. of 9th Annual Symposium on Theoretical Aspects of Computer Science (Feb.), Lecture Notes in Computer Science 577, 501–512, 1992.

[BJAAF92] J. Bitner, J. Jain, M. Abadir, J. A. Abraham, D. S. Fussel: Efficient Verification of Multipliers and Other Different Functions Using IBDD's, Proc. of Custom Integrated Circuits Conference (May), 1992.

[BHR91] Y. Breitbart, H. B. Hunt III, D. Rosenkrantz: The Size of Binary Decision Diagrams Representing Boolean Functions, Manuscript, 1991.

[BRB90] K. S. Brace, R. L. Rudell, R. E. Bryant: Efficient Implementation of a BDD Package, Proc. of 27th ACM/IEEE Design Automation Conference (Orlando, June), 40–45, 1990.

[BMS93] J. Bern, Ch. Meinel, A. Slobodova: TRUST – a Programming Environment for the FBDD–package, in print, Univ. Trier, 1993.

[Bry86] R. E. Bryant: Graph-Based Algorithms for Boolean Function Manipulation, IEEE Trans. Comput. C-35, 6 (Aug.), 677–691, 1986.

[Bry91] R. E. Bryant: On the Complexity of VLSI Implementations and Graph Representations of Boolean Functions with Applications to Integer Multiplication, IEEE Trans. Comput. 40, 2 (Feb.), 205–213, 1991.

[Bry92] R. E. Bryant: Symbolic Boolean Manipulation with Ordered Binary Decision Diagrams, ACM Computing Surveys, Vol. 24, No. 3 (Sep.), 293–318, 1992.

[Bry92b] R. E.Bryant, personal communic., (June), 1992.

[CHS74] R. L. Constable, H. B. Hunt III, S. Sahni: On the Computational Complexity of Scheme Equivalence, Proc. of 8th Princeton Conf. on Information Sciences and Systems, 1974.

[CMB90] O. Coudert, J.-C. Madre, C. Berthet: Verifying Temporal Properties of Sequential Machines without Building Their State Diagrams, Proc. of Computer-Aided Verification (Rutgers, N.J., June), 75–84, 1990.

[FHS78] S. Fortune, J. Hopcroft, E. M. Schmidt: The Complexity of Equivalence and Containment for Free Single Program Schemes, Proc. of Automata, Languages and Programming, Lecture Notes in Computer Science 62, 227–240, 1978.

[GJ79] M. R. Garey, D. Johnson: Computers and Intractability, W. H. Freeman & Co., New York, 1978.

[GM92] J. Gergov, Ch. Meinel: Analysis and Manipulation of Boolean Functions in Terms of Decision Graphs, Proc. of 18th International Workshop on Graph-Theoretic Concepts in Computer Science (June, 1992), Lecture Notes in Computer Science 657, 310–320, 1993.

[GM92b] J. Gergov, Ch. Meinel: Efficient Analysis and Manipulation of OBDD's Can Be Extended to Read-once–only Branching Programs, Tech. Rep. 92-10 (June; to appear in IEEE Transactions on Computers), Univ. Trier, 1992.

[GM93] J. Gergov, Ch. Meinel: Frontiers of Feasible and Probabilistic Feasible Boolean Manipulation with Branching Programs, Proc. of 10th Annual Symposium on Theoretical Aspects of Computer Science (Feb.), Lecture Notes in Computer Science 665, 576–585, 1993.

[JBFA92] J. Jain, J. Bitner, D. S. Fussel, J. Abraham: Probabilistic Verification of Boolean Functions, Formal Methods in System Design, 1: 63–117, 1992.

[JPHS91] S.-W. Jeong, B. Plessier, G. Hachtel, F. Somenzi: Extended BDD's: Trading off Canonicity for Structure in Verification Algorithms, Proc. of International Conference on Computer-Aided Design (Santa Clara, Calif., Nov.), 464–467.

[Kar89] K. Karplus: Using if-then-else DAGs for Multi-Level Logic Minimization, Proc. Advanced Research in VLSI, C. Seitz Ed., MIT Press, Cambridge, Mass., 101–118, 1989.

[KBS93] R. Krieger, B. Becker, R. Sinković: A BDD-based Algorithm for Computation of Exact Fault Detection Probabilities, Proc. of the International Symposium on Fault-Tolerant Computing, 1993.

[Mei89] Ch. Meinel: Modified Branching Programs and Their Computational Power, Springer-Verlag Berlin, Lecture Notes in Computer Science 370, 1989.

[Mei91] Ch. Meinel: Branching Programs - An Efficient Data Structure for Computer-Aided Circuit Design, Preprint UGH Paderborn, Nr. 93 (June), 1991.

[MWBS88] S. Malik, A. Wang, R. K. Brayton, A. Sangiovanni–Vincentelli: Logic Verification Using Binary Decision Diagrams in a Logic Synthesis Environment, Proc. of the IEEE International Conference on Computer-Aided Design (Santa Clara, Calif., Nov.), 6–9, 1988.

[SM93] A. Slobodova, Ch. Meinel: Efficient Manipulation of FBDD's by Means of a Modified OBDD–Package, Tech. Rep. 93-09 (April), Univ. Trier, 1993.

[SW92] D. Sieling, I. Wegener: Graph-driven BDDs — A New Data Structure for Boolean Functions, manuscript, 1992 (Aug.).

Circuit	OBDD Size	FBDD Size	Heuristic	Improvement (%)
c2670	—	313495	s 10 3	—
c5315	32842	8076	f' 12 3	76
s9234.1	20935	10060	t 7	52
c432	31290	19043	W 5	35
c1908	17176	10886	f 3 2	37
s1238	2020	1415	s' 2 1	30
s349	159	120	W 10	25
s838.1 (Z)	138	106	f 17 4	24
s838.1	1136	877	d 7 7	23
s344	158	122	W 10	23
s820	414	329	f 3 0	21
s526n	353	287	d 3 2	19
s526	360	297	d 3 2	18
s832	396	333	f 4 0	16
c3540	116806	98323	d 5 3	16
s1494	544	459	W 2	16
s1488	544	459	W 2	16
s420.1	320	274	d 15 12	15
c17	9	8	f 4 0	12
s953	768	689	d 3 14	11
s1196	1762	1580	d 2 10	11

Table 1: **Some FBDD–sizes w.r.t. various tree–like FBDD–types compared with OBDD–sizes w.r.t. the ordering heuristic of [MWBS88] of some ISCAS85 and ISCAS89 benchmarks.**

Abbreviations:
f .. .. = fs..mtr..-fh; f' .. .. = fs..mtr..-th; d .. .. = ds..mtr..-fh; W = wps-f-wpc; w = wps-wpc-f; s .. .. = ss..mtr..-fh; s' .. .. = ss..mtr..-td; t = ts..-fh; (. . .) = Output . . . only.

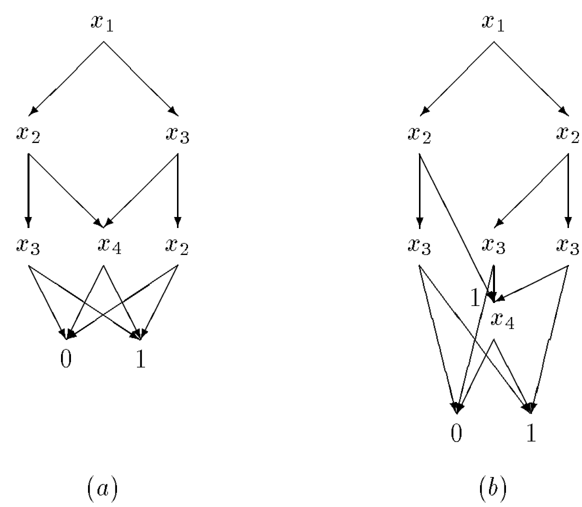

Figure 1: Example of an FBDD (a) and an OBDD (b) for the function $f(x_1, x_2, x_3, x_4) = \bar{x_1}\bar{x_2}x_3 + \bar{x_1}x_2x_4 + x_1\bar{x_3}x_4 + x_1x_2x_3$.

mtr (%)	maximal depth of *split* 0	1	2	3	4	5	6	7
0	[360]	*	*	*	407	487	626	764
1	[360]	*	*	*	407	487	626	764
2	[360]	*	*	**297**	396	565	582	699
3	[360]	*	*	**297**	396	565	582	697
4	[360]	*	321	392	517	574	598	622
5	[360]	*	*	*	*	488	607	734
6	[360]	*	*	*	*	506	638	790
7	[360]	*	*	448	437	527	545	569
8	[360]	*	*	342	350	436	393	393
9	[360]	*	341	383	372	463	455	455
10	[360]	*	*	*	*	*	*	*

Table 2: **FBDD–type–creation heuristic** *ds..mtr..-fh* **applied to ISCAS89 benchmark circuit s526**

In case of "*" the heuristic produces an OBDD-type. All depth-0 runs yield the pure *fh*–OBDD–heuristic (bracketed for clarification) given in [MWBS88].

mtr (%)	maximal depth of *split* 0	1	2	3	4	5	6	7
0	[414]	*	*	**329**	345	403	434	477
2	[414]	*	*	**329**	345	403	434	477
3	[414]	*	331	347	420	433	461	505
4	[414]	*	331	347	420	433	457	495
5	[414]	*	331	347	420	433	457	499
6	[414]	*	331	347	420	433	457	495
8	[414]	*	331	347	420	433	442	510
10	[414]	*	331	347	416	435	438	524
15	[414]	*	331	347	369	404	408	439
17	[414]	*	331	347	369	369	405	412
18	[414]	*	*	*	*	*	*	*

Table 3: **FBDD–type–creation heuristic** *fs..mtr..-fh* **applied to ISCAS89 benchmark circuit s820**

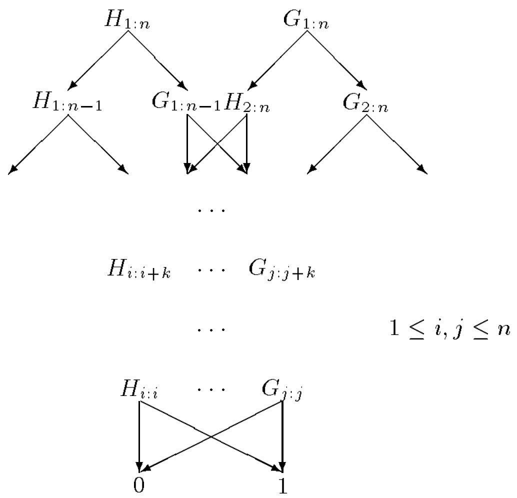

Figure 2: Construction of an FBDD P_{HWB} **that computes the hidden weighted bit function** HWB.

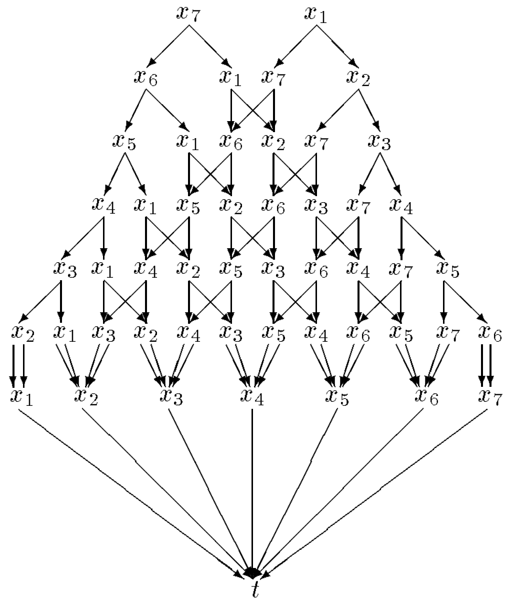

Figure 3: The type $\tau_{HWB} = tp(P_{HWB})$ **for** $n = 7$ **of the FBDD for the hidden weighted bit function given in Section 3 obtained from the types for** $H_{1:n} (= HWB)$ **and for** $G_{1:n}$. **(**τ_{HWB} **roots in the source labeled by** x_7.**)**

Algorithm for $SYN_{\tau FBDD}$
Input:
- a Boolean operation $* \in \mathbb{B}_2$;
- a type τ;
- two FBDD's P' and P'' of type τ;

Output:
- an FBDD P of type τ representing $f = f_{P'} * f_{P''}$;

begin
  Q:=*synthesis*($*$, τ, P', P'');
  P := *red*(Q);
  return(P);
end.

synthesis($*$, τ, P', P'')
begin
if (P' or P'' is a sink) /* **terminal case** [a] */
  then return the result;
  /* the construction of P is straightforward */
  else
    $x := top(\tau)$ [b] ;
    $P_0 := synthesis(*, \tau|_{x=0}, P'|_{x=0}, P''|_{x=0})$;
    $P_1 := synthesis(*, \tau|_{x=1}, P'|_{x=1}, P''|_{x=1})$;
    construct P such that
    $top(P) = x$ and $P|_{x=\alpha} = P_\alpha$, $\alpha \in \{0, 1\}$;
    return(P);
end.

[a] The terminal case is reached if one of the two input FBDD's P' and P'' represents a constant. Then the recursion stops, and the output FBDD can be derived by modifying the sinks of the other FBDD. For instance, if P' is the 1-sink and $* = \oplus$, each β-sink of P'' has to be replaced by a $\bar{\beta}$-sink, $\beta \in \{0,1\}$.

[b] Let $top(\tau)$ denote the top variable (the label of the source) of τ. Then $\tau|_{top(\tau)=\alpha}$ is the type rooting in the α-successor, $\alpha \in \{0,1\}$, of the source of τ. Similar notations are used in the case of FBDD's.

Figure 4: Pseudo–code for the apply procedure for FBDD's of the same type.