Electronic Colloquium on Computational Complexity, Report No. 39 (1998)

ECCC

Ordered Binary Decision Diagrams and Their Significance in Computer-Aided Design of VLSI Circuits – a Survey

Christoph Meinel
FB IV – Informatik
Universität Trier
D – 54286 Trier
meinel@uni-trier.de

Thorsten Theobald
Zentrum Mathematik
TU München
D – 80290 München
theobald@mathematik.tu-muenchen.de

Abstract

Many problems in computer-aided design of highly integrated circuits (CAD for VLSI) can be transformed to the task of manipulating objects over finite domains. The efficiency of these operations depends substantially on the chosen data structures. In the last years, ordered binary decision diagrams (OBDDs) have proven to be a very efficient data structure in this context. Here, we give a survey on these developments and stress the deep interactions between basic research and practically relevant applied research with its immediate impact on the performance improvement of modern CAD design and verification tools.

1 Introduction

The development of digital circuits by means of CAD (Computer-Aided Design) systems has a strong influence on many areas of computer science. Applications in information processing, telecommunication or in industrial control systems permanently require the construction of more and more powerful high-speed circuits. On the one hand, this imposes bigger and bigger challenges upon the CAD systems. On the other hand, all these systems underlie the inherent complexity in the manipulation of switching functions which has been extensively studied in theoretical computer science, see e.g. [32, 46]. One of the main problems here is to get the immensely increasing complexity of mathematical objects, the so-called *combinatorial explosion*, under control.

A central problem in the design of CAD systems for VLSI circuits (Very Large Scale Integration) is to *represent* the functional behavior of a circuit. For an illustration we will shortly consider the problem of *combinational circuit verification*. Hereby it is to check whether a combinational circuit C satisfies a given specification S. For the solution of this problem computer-internal representations of C and S have to be determined which can then be used to test the relevant properties. Of course, this approach only leads to a practical procedure, if both representations can be computed efficiently and practical algorithms are available to decide equivalence, satisfiability and similar properties by means of the representations.

The mentioned representations are realized internally via *data structures*. Within the last decade, ordered binary decision diagrams have proven to be the most suitable data structure in this context. Although they were originally only used as data structure in the context of CAD applications, meanwhile, ordered binary decision diagrams have also been applied successfully in many other areas like the design and verification of communication

ISSN 1433-8092

protocols [16, 35] or for solving combinatorial problems [19, 38]. In the present article we survey the foundations, applications and current developments in this environment.

2 Data Structures for Switching Functions

In computer-aided design, Boolean functions $f : \{0,1\}^n \to \{0,1\}$ are of central importance for describing the switching behavior of digital circuits. Hence, those functions are also called *switching functions*. By introducing a suitable 0-1-encoding, all finite problems can – at least in principle – be modeled by means of switching functions. The great importance of switching functions stems from the possibility to obtain substantially simplified, optimized and with optional properties provided circuits by applying optimization techniques during the design process. In the area of VLSI circuits this task is performed by CAD systems. Before it is possible to apply optimization techniques, the switching functions themselves have to be *described* (or equivalently: *represented*) uniquely and as efficiently as possible in computers.

2.1 Classic Representations

Well-known classic representations of switching functions include truth tables, disjunctive normal forms, Boolean formulas, or multi-level representations using net-lists of gates which are all based on the idea to describe the given switching function by means of a computation rule. With permanently increasing performance requirements the drawbacks of these representations have become more and more serious: Descriptions in form of a truth table are e.g. never compact. For the more compact representations there are at the moment insurmountable problems regarding the algorithmic handling: Already the test if two disjunctive normal forms, two Boolean formulas or two net-lists of gates represent the same function is co-NP-complete [25].

2.2 OBDDs – Ordered Binary Decision Diagrams

In 1986, by introducing *ordered binary decision diagrams (OBDDs)*, Randy Bryant from Carnegie Mellon University got ahead a fundamental step in the search for suitable data structures in circuit design [9, 11]. In contrast to conventional descriptions based on computation rules, OBDDs are based on a decision process. That way, Bryant combined two crucial advantages: the new established data structure is not only very compact but can also be handled excellently from the algorithmic point of view.

We explain ordered binary decision diagrams by inspecting an example. The Boolean function

$$f = bc + a\bar{b}\bar{c}$$

can be represented by means of a (binary) *decision diagram* like in Figure 1. Such decision diagrams are directed, acyclic graphs which have exactly one node without predecessor, the *root*. Each non-terminal node is labeled by a variable and has two outgoing edges: a solid drawn *1-edge* and a dashed drawn *0-edge*. Each terminal node is labeled by one of the constants 0 or 1 and is called *sink*.

Decision diagrams represent Boolean functions in a natural manner: Each assignment to the input variables defines a unique path through the graph from the root to a sink. The value of this sink defines the function value on this input. A decision diagram is called *ordered* if the sequence of variables on each path from the root to the sinks is consistent with a *fixed* order. Obviously, for each given variable order π, one can construct such an ordered binary

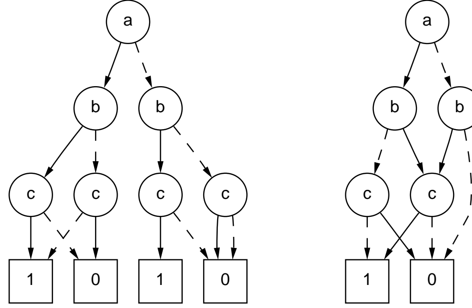


Figure 1: Two OBDDs of $f = bc + a\overline{b}\overline{c}$

decision diagram (OBDD), e.g. in form of a complete tree. The difficulty in representing Boolean functions by means of decision diagrams is caused by the missing uniqueness, like in many other representations. By using an ingenious reduction mechanism, this problem can be solved for OBDDs very elegantly. Obviously, the following three reduction rules keep the represented function invariant:

Terminal rule: Delete all terminal nodes with a given label but one, and redirect all incoming edges in the eliminated nodes to the remaining one.

Elimination rule: If 1- and 0-edge of a node v point to the same node u, then eliminate v, and redirect all incoming edges to u.

Merging rule: If the non-terminal nodes u and v are labeled by the same variable, their 1-edges lead to the same node and their 0-edges lead to the same node, then eliminate one of the two nodes u, v, and redirect all incoming edges to the remaining node.

The elimination rule and the merging rule are illustrated in Figure 2.

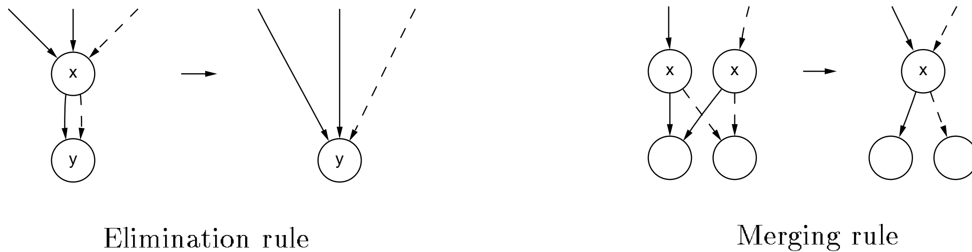


Figure 2: Reduction rules

Definition 1. *An OBDD is called reduced if none of the three reduction rules can be applied.*

Hence, the right OBDD in Figure 1 is reduced. Regarding the algorithmic properties of reduced OBDDs, the following property of canonicity is of basic importance:

> *With respect to each fixed variable order, the reduced OBDD of a Boolean function f is determined uniquely.*

3 Construction and Manipulation of OBDDs

Besides canonicity, OBDDs have another property which is of the same importance: the brilliant algorithmic manipulation.

3.1 Binary Operations

By $*$ we denote an arbitrary Boolean operation, e.g. the conjunction or the disjunction. In order to compute the OBDD of $f * g$ from the OBDD representations of two functions f and g, one can use Shannon's expansion w.r.t. the leading variable x in the variable order π:

$$f * g \;=\; x\;(f|_{x=1} * g|_{x=1}) \;+\; \overline{x}\;(f|_{x=0} * g|_{x=0}),$$

where $f|_{x=1}$ is the subfunction that results from f after replacing the variable x by the value 1. By repeated application of this decomposition an OBDD of the function $f*g$ is computed. In order to perform this operation efficiently, multiple calls with the same argument pairs are avoided – instead, the already computed results from earlier stages are being recalled from a table. In this way, the originally exponential number of decompositions is now bounded by the product of the two OBDD-sizes. If *size* denotes the number of nodes in an OBDD, the basic algorithmic property concerning efficient manipulation can be formulated as follows:

> *Let the two Boolean functions f_1 and f_2 be represented by reduced OBDDs P_1 and P_2 w.r.t. the same variable ordering. For each binary operation $*$ the reduced OBDD P of $f = f_1 * f_2$ can be determined in time $O(size(P_1) \cdot size(P_2))$.*

3.2 Implementation Techniques

A variety of design decisions has contributed substantially to efficient implementations of the OBDD data structure and hence crucially to its success. We would like to outline the most important ones.

Shared OBDDs. Several functions can be represented in a single directed acyclic graph with several roots like in Figure 3.

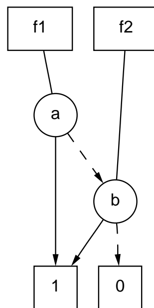


Figure 3: Functions f_1, f_2 in a shared OBDD

Unique table. It should be avoided to build non-reduced OBDDs which have to be reduced afterwards. In order to guarantee the reduced form at each moment, a table is used for bookkeeping which functions have already been represented within the graph.

Strong canonicity. Due to the unique table two equivalent functions are represented by exactly the same subgraph within the shared OBDD. This property is called *strong canonicity* and allows to test the equivalence of two functions by means of a single pointer comparison.

Complemented edges. The edges are equipped with an additional attribute bit. By means of this bit a function and its complement can be represented by the same subgraph, and a function can be complemented in constant time. Some additional requirements to the allowed positions of the complement edges help to preserve the canonicity property.

Computed table. The table in which previously computed results are stored is implemented by means of a hash table in order to guarantee a fast retrieval.

Memory management. In a typical OBDD application a large number of OBDDs are constructed and then deleted again. For efficient administration of the nodes in memory, the nodes which are no longer used are not freed immediately. Instead, a *garbage collection* is called from time to time in which these nodes are freed jointly.

3.3 Symbolic Simulation

A central problem in circuit design is the question whether two *combinational circuits* (i.e. circuits without feedback) C_1 and C_2, being given through net-lists of gates, agree in their logic behavior, see Figure 4.

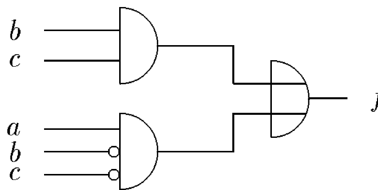


Figure 4: Combinational circuit of $f = bc + a\overline{b}\overline{c}$

The conversion into the OBDD representation is done my means of a *symbolic simulation*: Starting with the (trivial) OBDD representations of the input nodes one successively constructs, in topological order, OBDDs for each gate from the OBDDs of the corresponding predecessor gates. In case of a strong canonical representation, the equivalence test itself consists of a single pointer comparison. Each step in the iteration can be performed efficiently w.r.t. the OBDD-sizes of the predecessor gates. This shows that the difficulty of the NP-complete equivalent test [25] has now been shifted into the representation size. Of course, it may happen that the OBDDs of the circuits are quite large. However, many circuits of the real world inherently contain much structure – hence, the reduction rules of the OBDDs cause the graphs describing the circuit to remain pleasantly small.

3.4 Implementations

In the last years several so-called BDD packages have been developed which provide numerous functions for the efficient manipulation of switching functions. Although many of these packages have been developed at academic institutions they have been employed in commercial CAD systems nevertheless.

The first package in this historical development has been developed by Karl Brace at Carnegie Mellon University [7]. Many of the above mentioned implementation techniques

go back to this package. Some time later the experiences with this package have been converted into a new, improved package being developed and implemented by David Long at the same university [29]. An important innovation in this package were techniques for dynamic constructing good variable orders which will be discussed in the next section. The packages of Brace and Long enjoyed large worldwide dissemination. An additional step with regard to improved efficiency and improved algorithms for finding good variable orders has been carried out by Fabio Somenzi (University of Colorado at Boulder) with the so-called CUDD package in 1996 [44].

4 The Importance of Variable Ordering for OBDDs

4.1 Influence

The size of an OBDD and hence the complexity of its manipulation depends on the underlying variable order – this dependency can be quite strong. An extreme example is shown in Figure 5. With respect to the variable order $x_1, x_2, \ldots, x_{2n-1}, x_{2n}$ the function

$$x_1x_2 + x_3x_4 + \ldots + x_{2n-1}x_{2n}$$

can be represented by an OBDD of linear size. For the variable order $x_1, x_3, \ldots, x_{2n-1}, x_2, x_4, \ldots, x_{2n}$ however, the size of the reduced OBDD grows exponentially in n.

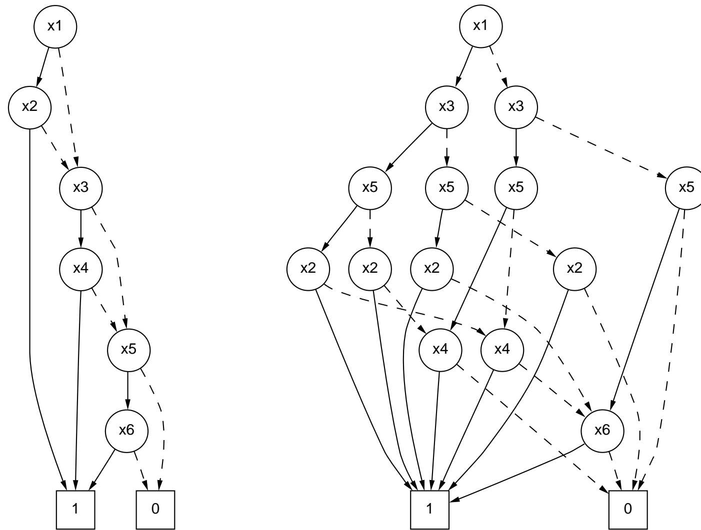


Figure 5: Influence of the variable order

The same effect occurs in the case of adder functions – here too, depending on the variable order, the OBDD-size varies from linear to exponential in the number of input bits. Other important functions, e.g. the multiplication of two n-bit numbers imply OBDDs of exponential size w.r.t. each variable order [10].

4.2 Optimization Strategies

Due to the strong dependence of the OBDD-size upon the chosen variable order it is one of the most important problems in the use of OBDDs to construct good orders. However,

the problem to construct an optimal order of a given OBDD is NP-hard [45, 4]. The currently best known exact procedure is based on dynamic programming and has running time $O(n^2 \cdot 3^n)$ [24]. Unfortunately, for serious applications this method is useless. The practically relevant optimization strategies can be classified into two categories: *heuristics* and *dynamic reordering.*

Heuristics. Here, the idea is to deduce a priori some information from the application which is useful for determining a good variable order. In the context of a symbolic simulation, numerous methods have been developed to obtain a good order from the topological structure of a circuit [30]. One of the drawbacks of the heuristic methods is that their effectiveness is quite problem specific and that so far, there is no heuristic which is suitable for all cases. A current research task is the construction of heuristics that directly deduce a good order from a given OBDD.

Dynamic reordering. Another technique to minimize OBDD-sizes is to improve the variable order during the processing dynamically. The currently best reordering strategy goes back to Richard Rudell and is called *Sifting* [41]. The method is based primarily on a subroutine that looks for the best position of a specific variable if the positions of all other variables remain fixed. The algorithm can be implemented efficiently and produces excellent results in practical applications. By appropriately exploiting additional criteria like symmetry relations among individual variables [39] or structural considerations (*block-restricted Sifting* [33]) the basic algorithm of Sifting can be further improved.

5 OBDD-Based Analysis of Sequential Systems

In the design of complex systems it becomes more and more important to guarantee correctness. The dramatic extent that an error can lead to is illustrated by the example of the Intel Pentium processor from 1994: In the case of the Pentium implementation, a table of the well-known "SRT" divider circuit (named after the initials of the three inventors) [1] contained incorrect entries [18]. Although Intel argued for a longer time that in practice this mistake would not have a serious influence on the computations [42], a recall offer became unavoidable. The costs of this recall were estimated to US$ 475 million. Very high follow-up costs of such design errors have made the area of *hardware verification* become one of the essential steps within a design process.

5.1 Formal Verification

Many verification problems can be modeled by means of synchronous systems with finitely many states, so-called *sequential systems* or *finite state machines*. Figure 6 shows an example of those systems. One of the basic tasks which often has to be solved in this context is the equivalence test of two finite state machines which are given by net-lists of gates (see Figure 7).

For the two given sequential systems we would like to prove that their input/output behavior is identical. In a typical application one machine may specify a functional behavior and the other machine is a highly optimized implementation. The equivalence test of finite state machines has been investigated in computer science for many years. However, for systems whose states are encoded e.g. by 80 bits the number of possible states is 2^{80}. Such a large number is accessible to an intuitive comprehension only with great difficulties, and hence, it shall be illustrated using a comparing number from the real world: The age of our

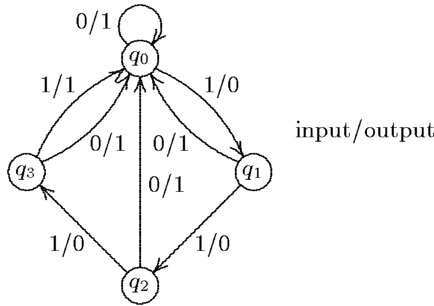


Figure 6: Simple finite state machine

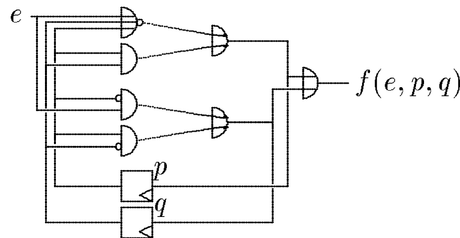


Figure 7: Gate representation of a simple sequential system with latches p, q for remembering the state

whole universe is about 2^{34} years. A computer which has been investigating 2 million states per seconds since this hour of birth would not have been ready yet !

As a consequence of this dilemma the correctness of real sequential systems has been verified only by means of a large number of simulations. Usually, this approach does not cover all cases. In contrast to this, the approach of *formal verification* aims at providing a complete proof of the correctness of a circuit. By using OBDD data structures this approach has become feasible in completely new dimensions. Hereby, the conversion of a problem's difficulty into the manageable size of the representation is of basic importance. Of course, it may happen that systems with 80 state bits lead to very large OBDDs. Many systems however contain very regular structures which keep the relevant OBDDs and hence the running times quite small.

The core of the OBDD-based method is to reduce the verification of global properties like equivalence to the verification of local properties which hold for all those states that can be reached from the initial state. For this reason, *reachability analysis* plays a central role in the process of formal verification: this term denotes the efficient computation and compact representation of the set of reachable sets by using OBDD data structures.

The equivalence test of two finite state machines M_1 and M_2 itself can be reduced to a reachability analysis by using the construction in Figure 8: Let M denote the so-called product machine whose state space is the Cartesian product of the spaces of M_1 and M_2. The output of M for a given state and a given input is 1 if and only if for this configuration the outputs of M_1 and M_2 agree. M_1 and M_2 have the same input/output behavior if and only if the output of M is 1 for all reachable states.

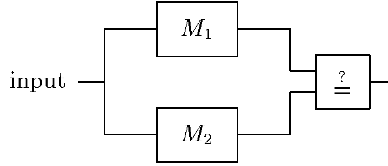


Figure 8: Product machine

5.2 Reachability Analysis Based on OBDDs

As already mentioned in the previous section, the set of reachable states can be quite large. Hence, an explicit representation of this set, e.g. in form of a list, cannot be suitable under any circumstances. Coudert, Berthet and Madre have investigated the characteristic function of state sets which can be considered as a Boolean function and therefore be represented by an OBDD [20, 21]. They have shown that this representation form goes well together with the operations which have to be performed for the computation of the reachable states: If reachable states are computed according to a breadth-first-traversal then the representation via the characteristic function allows to compute all corresponding successor states within a single computation. For this reason, one also uses the term *symbolic breadth-first traversal.* Once more, the complexity of the computation depends on the OBDD-size of the occurring state sets.

A basic variant of symbolic breadth-first traversal can be outlined as follows: For a finite state machine M with p input bits, n state bits and next-state function $\delta : \{0,1\}^{n+p} \to \{0,1\}^n$ let $\chi_j(x_1,\ldots,x_n) : \{0,1\}^n \to \{0,1\}$ be the characteristic function of all states being reachable in at most j steps. The computation of the function χ_{j+1} starting from the function χ_j can be described by the following Boolean equation which reflects the image computation of all states of χ_j under the mapping δ:

$$\begin{aligned} \chi_{j+1}(y_1,\ldots,y_n) &= \chi_j(y_1,\ldots,y_n) \\ &\quad + \exists x_1,\ldots,x_n \, \exists e_1,\ldots,e_p \left(\prod_{i=1}^{n} (y_i \equiv \delta_i(x,e)) \, \chi_j(x_1,\ldots,x_n) \right), \end{aligned}$$

where $\equiv$ is the Boolean equivalence function and $\exists x_i$ the Boolean existential quantifier

$$\exists x_i f = f|_{x_i=0} + f|_{x_i=1}.$$

This iteration step is repeated until a fixed point is reached which represents the set of reachable states. There are many refinements and variants of this form of image computation which all aim at keeping possible intermediate results small. The currently best methods for image computation are based essentially on partitioning the part of the above equation that is constant for all iterations,

$$\prod_{i=1}^{n} (y_i \equiv \delta_i(x,e)),$$

the so-called *transition relation.* By choosing a suitable partition it is possible to perform the quantifications rather efficiently [13].

5.3 Model Checking

Model checking is the problem to decide whether an implementation satisfies a specification that is given by a Boolean formula. Due to the formulation of the specification within a formal logic it is possible to describe system properties like invariants, liveness or fairness properties completely independent of implementation details. One of the temporal logics which often forms the basis of those specifications is the so-called *Computation Tree Logic* CTL [15].

The idea to combine model checking algorithms with symbolic OBDD algorithms has been developed independently by several research groups: On the one hand by Coudert, Madre and Berthet [22], on the other hand by Burch, Clarke, McMillan and Dill [14], and as third group by Bose and Fischer [6].

Due to the symbolic OBDD representations one also uses the term *symbolic model checking*. As the OBDD data structures can automatically recognize regularities, this approach made it possible to verify real systems with up to 10^{100} states [13] – as a comparison: The number of atoms in the universe amounts to approximately 10^{77}. For systems containing less regularity the OBDD-based approach often allows a formal verification up to the region of 10^{25} to 10^{30} states. We would like to explain the basic ideas of CTL model checking.

5.3.1 CTL

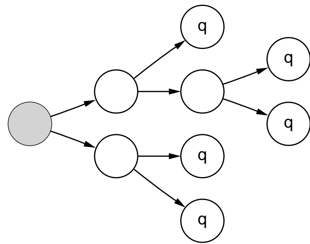
The formulas of the logic describe properties of computation paths. Hereby, a computation path is an infinite sequence of states which are traversed during the processing. In addition to the logical operators `AND`, `OR` and `NOT`, the logic CTL contains four operators to express temporal relations.

The *next time* operator **X** denotes a condition that is valid in the next state of a computation. For a CTL formula f the formula $\mathbf{X}f$ holds on a computation path p if and only if f holds in the successor state of p's initial state. The *global* operator **G** denotes a property which holds globally in all states of the computation path. The *future* operator **F** denotes a property that holds eventually in the future. Finally, the *until* operator $f\mathbf{U}g$ holds on a computation path p if and only if there is a state s on p in which g is valid and f holds in all states preceding s.

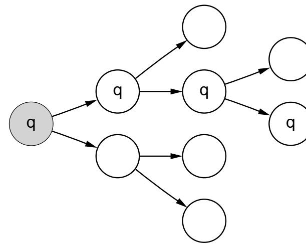
In general, more than one computation path starts in a given state. For this reason, each operator in CTL is preceded by a *path quantifier*. If a temporal operator is preceded by the *universal path quantifier* **A** then the property has to hold on all possible computation paths beginning in the relevant state. Hence, $\mathbf{AG}f$ is valid in a state s if f globally holds on all possible computation paths beginning in s. The *existential path quantifier* **E** expresses that the subsequent condition holds on at least one state of the computation path that start in the relevant state. Both path quantifiers are illustrated in Figure 9.

5.3.2 Symbolic Model Checking with CTL

We now sketch an OBDD-based method for deciding whether a given formula f holds in a particular state of a sequential system that is given through its transition relation R. The algorithm is based on the function CHECK which has two arguments: a formula f and a representation R of its transition relation. It returns an OBDD with the following property: CHECK(f, R) returns `TRUE` in a given state of the sequential system if and only if the formula f is valid in this state. Analogous to OBDD-based reachability analysis, all state sets are represented symbolically by means of their characteristic functions.



AFq holds in the marked state.



EGq holds in the marked state.

Figure 9: Path quantifiers

The transition relation R of the sequential system shall be represented by an OBDD. $R(s_0, s_1)$ is 1 if and only if s_1 is a successor state of s_0. We assume that we have computed an OBDD which represents all states satisfying the formula f. From this, we want to obtain an OBDD representing all states that satisfy the formula $\mathbf{EX}f$. The formula is valid in state s_0 if and only if a there exists a successor state of s_0 satisfying f. In Boolean notation we can write

$$\textsc{Check}(\mathbf{EX}f, R)(s_0) = \exists s_1 \left(R(s_0, s_1) \texttt{ AND } \textsc{Check}(f, R)(s_1) \right).$$

Analogously, the other operators can also be described by Boolean equations that work together with OBDD data structures quite efficiently. Partially, these formulations lead to fixed point computations similar to reachability analysis.

5.4 Implementations

Based on the presented techniques several OBDD-based model checkers have been implemented and used in industrial design cycles. First, we want to mention the symbolic model checker SMV developed by Ken McMillan at Carnegie Mellon University [31]. This system has also been used within numerous other systems. The VIS system (Verification Interacting With Synthesis) being developed primarily at the University of California at Berkeley and the University of Colorado at Boulder unifies the mentioned verification techniques for finite state machines and techniques for synthesis of VLSI circuits [8]. Meanwhile, there are also commercial systems, e.g. CVE (Circuit Verification Environment) by Siemens [5], or the system RuleBase by IBM [2] which is built on top of SMV.

6 Variants and Extensions of OBDDs

For further improving the efficiency of the data structures, several variants and extensions of OBDDs have been proposed. For some specific application fields, these refined models are better suited than the "classic" OBDDs. We would like to sketch some particularly interesting and important developments. Altogether, research efforts in this area have not been completed yet, and hence, the quality and the importance of many variants may not be finally judged yet.

6.1 Relaxing the Ordering Restrictions

Some important functions like multiplication of binary numbers or indirect storage access (e.g. the *hidden weighted bit* function) have provably exponential-size OBDDs w.r.t. each variable order. A possible approach to eliminate this problem is to relax the linear ordering restriction of OBDDs without destroying the excellent algorithmic properties too much. Indeed, it is possible to allow different orders on different root-to-sink-paths (see Figure 10). As long as each variable on each path is read at most once and all represented functions obey the same generalized order, the canonicity and the polynomial time complexity of performing binary operations are preserved [26, 43]. By means of these so-called *free binary decision diagrams* (FBDDs) the hidden weighted bit function can be represented in polynomial space – for the multiplication of two binary numbers however it is known that FBDDs need exponential space, too [40].

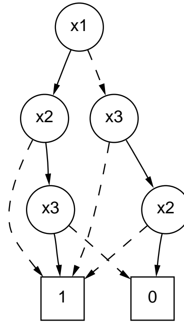


Figure 10: Free binary decision diagram

6.2 Transformations

Another recently developed variant converts Boolean functions into functions with easier representations [3], similar to classic transformation concepts like Fourier transformation.

More precisely, let $\mathbb{B}_n$ denote the set of all n-variable Boolean functions $f : \{0,1\}^n \to \{0,1\}^n$. The transformation approach is based on *cube transformations* τ which are bijective mappings from $\{0,1\}^n \to \{0,1\}^n$. A cube transformation τ induces a mapping $\Phi_\tau : \mathbb{B}_n \to \mathbb{B}_n$ with $\Phi_\tau(f)(a) = f(\tau(a))$ for every $a = (a_1,\ldots,a_n) \in \{0,1\}^n$. Now, instead of representing and manipulating the original function $f \in \mathbb{B}_n$, the idea is to use the transformed function $\Phi_\tau(f) = f(\tau)$. This variable transformation preserves the efficient manipulation as shown by the following fact:

Fact 2. *If $\tau : \{0,1\}^n \to \{0,1\}^n$ is a cube transformation, and $f_1, f_2 \in \mathbb{B}_n$ are Boolean functions, then Φ_τ defines an automorphism on $\mathbb{B}_n$, i.e. the following holds:*

1. *$f_1 = g_1$ if and only if $\Phi_\tau(f_1) = \Phi_\tau(g_1)$.*
2. *Let $*$ be any binary operation on $\mathbb{B}_n$. If $f = f_1 * f_2$, then $\Phi_\tau(f) = \Phi_\tau(f_1) * \Phi_\tau(f_2)$.*

In other words, due to the second statement the polynomial complexity of the Boolean operations remain valid even if we work with the transformed functions. If, for example, one wants to check two given functions for equivalence, statement 1 tells us, that in this situation it is not necessary at all to retransform the functions.

The realization of this general framework requires to find good techniques for finding suitable variable transformations τ which lead to small OBDD-sizes for the transformed versions of the relevant functions. Suitable transformations that have already demonstrated their optimization power include graph-driven transformations [3] and linear transformations [36, 34].

6.3 Alternative Decompositions

If f denotes the function being represented by an OBDD-node with label x_i, and g, h denote the functions being represented in the two sons, then Shannon's decomposition holds:

$$f = x_i g + \overline{x_i} h.$$

However, it is also possible to perform other decompositions in the nodes, e.g. the so-called Reed-Muller decomposition

$$f = g \oplus x_i h.$$

These *ordered functional decision diagrams* (OFDDs), introduced in [27], are particularly suited in the context of problems based on the exclusive-or operation, e.g. the minimization of AND-XOR-polynomials. One step further, in [23] it is shown that different decomposition types can be combined within the same subgraph while preserving good algorithmic properties (*ordered Kronecker functional decision diagrams*, OKFDDs).

6.4 Zero-Suppressed BDDs

In many applications with combinatorial background the corresponding Boolean functions have a 1 at only very few positions. In the so-called *zero-suppressed BDDs* (ZBDDs, ZDDs) this fact is exploited by means of a modified reduction rule [37, 38]: One does not eliminate (like in OBDDs) those nodes having identical 0- and 1-successor, but those nodes whose 1-successor is the sink with label 0 (see Figure 11). By using this data structure many problems in the area of two-level and multi-level logic optimization have been solved efficiently [19, 38]. An example from a quite different area may illustrate the fundamental influence of OBDD-based data structures: Löbbing and Wegener from the University of Dortmund report successful ZDD experiments for solving difficult combinatorial problems which occur in the analysis of knight moves on a chess board.

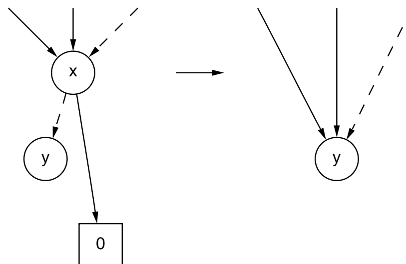


Figure 11: Elimination rule in a zero-suppressed BDD

6.5 Multi-Valued Functions

Several approaches have tried to extend the efficient manipulation to non-binary functions. In this context, we only mention the so-called *multi-terminal BDDs* (MTBDDs) which realize this idea in a natural manner by adding further sinks [17].

On the contrary, *binary moment diagrams* (BMDs) employ the decomposition

$$f = (1 - x_i)g + x_i h$$

and are better suited for representing and manipulating arithmetic functions of the type $f : \{0,1\}^n \to \mathbb{Z}$ like e.g. multiplication [12].

Finally, *edge-valued binary decision diagrams* (EVBDDs) include edge weights in order to improve the sharing of subgraphs [28]. Interesting applications of this data structure include solving combinatorial optimization problems.

7 Summary and Outlook

The search for efficient data structures supporting the manipulation of switching functions in CAD applications provides an instructive example of the exciting and manifold interaction between real problems and fundamental questions in computer science research. Caused by the question for improved data structures the performance frontier of existing design systems has been extended substantially. Considering the fact that each improvement of the representation immediately propagates to the efficiency and practicability of many applications, further intensive research and development work will have to be carried out.

References

[1] D. E. Atkins. Higher-radix division using estimates of the divisor and partial remainder. *IEEE Transactions on Computers*, C–17:925–934, 1968.

[2] I. Beer, S. Ben-David, C. Eisner, and A. Landver. RuleBase: An industry-oriented formal verification tool. In *Proc. 33rd ACM/IEEE Design Automation Conference (Las Vegas, NV)*, pages 655–660, 1996.

[3] J. Bern, Ch. Meinel, and A. Slobodová. OBDD-based Boolean manipulation in CAD beyond current limits. In *Proc. 32nd ACM/IEEE Design Automation Conference (San Francisco, CA)*, pages 408–413, 1995.

[4] B. Bollig and I. Wegener. Improving the variable ordering of OBDDs is NP-complete. *IEEE Transactions on Computers*, 45:993–1002, 1996.

[5] J. Bormann, J. Lohse, M. Payer, and G. Venzl. Model checking in industrial hardware design. In *Proc. 32nd ACM/IEEE Design Automation Conference (San Francisco, CA)*, pages 298–303, 1995.

[6] S. Bose and A. Fisher. Automatic verification of synchronous circuits using symblic logic simulation and temporal logic. In *Proc. IMEC-IFIP International Workshop on Applied Formal Methods for Correct VLSI Design*, pages 759–764, 1989.

[7] K. S. Brace, R. L. Rudell, and R. E. Bryant. Efficient implementation of a BDD package. In *Proc. 27th ACM/IEEE Design Automation Conference (Orlando, FL)*, pages 40–45, 1990.

[8] R. K. Brayton, G. D. Hachtel, A. Sangiovanni-Vincentelli, F. Somenzi, et al. VIS: A system for verification and synthesis. In *Proc. Computer-Aided Verification '96*, volume 1102 of *Lecture Notes in Computer Science*, pages 428–432. Springer, 1996.

[9] R. E. Bryant. Graph-based algorithms for Boolean function manipulation. *IEEE Transactions on Computers*, C–35:677–691, 1986.

[10] R. E. Bryant. On the complexity of VLSI implementations and graph representations of Boolean functions with application to integer multiplication. *IEEE Transactions on Computers*, C–40:205–213, 1991.

[11] R. E. Bryant. Symbolic Boolean manipulation with ordered binary decision diagrams. *ACM Computing Surveys*, 24(3):293–318, 1992.

[12] R. E. Bryant and Y.-A. Chen. Verification of arithmetic circuits with binary moment diagrams. In *Proc. 32nd ACM/IEEE Design Automation Conference (San Francisco, CA)*, pages 535–541, 1995.

[13] J. R. Burch, E. M. Clarke, D. E. Long, K. L. McMillan, and D. L. Dill. Symbolic model checking for sequential circuit verification. *IEEE Transactions on Computer-Aided Design of Integrated Circuits*, 13:401–424, 1994.

[14] J. R. Burch, E. M. Clarke, K.L. McMillan, and D. L. Dill. Sequential circuit verification using symbolic model checking. In *Proc. 27th ACM/IEEE Design Automation Conference (Orlando, FL)*, 1990.

[15] E. M. Clarke, E. A. Emerson, and A. P. Sistla. Automatic verification of finite-state concurrent systems using temporal logic specifications. *ACM Transactions on Programming Languages and Systems*, 8:244–263, 1986.

[16] E. M. Clarke, O. Grumberg, H. Hiraishi, S. Jha, D. E. Long, K. L. McMillan, and L. A. Ness. Verification of the Futurebus+ cache coherence protocol. In *Proc. Conference on Hardware Description Languages*, 1993.

[17] E. M. Clarke, K. L. McMillan, X. Zhao, M. Fujita, and J. C.-Y. Yang. Spectral transforms for large Boolean functions with application to technology mapping. In *Proc. 30th ACM/IEEE Design Automation Conference (Dallas, TX)*, pages 54–60, 1993.

[18] T. Coe. Inside the Pentium FDIV bug. *Dr. Dobb's Journal*, 20(4):129–135, 1995.

[19] O. Coudert. Two-level logic minimization: An overview. *INTEGRATION, the VLSI journal*, 17:97–140, 1994.

[20] O. Coudert, C. Berthet, and J. C. Madre. Verification of synchronous sequential machines using symbolic execution. In *Proc. Workshop on Automatic Verification Methods for Finite State Machines*, volume 407 of *Lecture Notes in Computer Science*, pages 365–373. Springer, 1989.

[21] O. Coudert and J. C. Madre. The implicit set paradigm: A new approach to finite state system verification. *Formal Methods in System Design*, 6(2):133–145, 1995.

[22] O. Coudert, J. C. Madre, and C. Berthet. Verification of synchronous sequential machines using symbolic execution without building their state diagrams. In *Computer-Aided Verification '90*, volume 531 of *Lecture Notes in Computer Science*. Springer, 1990.

[23] R. Drechsler, A. Sarabi, M. Theobald, B. Becker, and M. A. Perkowski. Efficient representation and manipulation of switching functions based on ordered Kronecker functional decision diagrams. In *Proc. 31st ACM/IEEE Design Automation Conference (San Diego, CA)*, pages 415–419, 1994.

[24] S. J. Friedman and K. J. Supowit. Finding the optimal variable ordering for binary decision diagrams. *IEEE Transactions on Computers*, 39:710–713, 1990.

[25] M. R. Garey and M. Johnson. *Computers and Intractibility: A Guide to the Theory of NP-Completeness*. W. H. Freeman, 1978.

[26] J. Gergov and Ch. Meinel. Efficient analysis and manipulation of OBDDs can be extended to FBDDs. *IEEE Transactions on Computers*, 43(10):1197–1209, 1994.

[27] U. Kebschull, E. Schubert, and W. Rosenstiel. Multilevel logic synthesis based on functional decision diagrams. In *Proc. European Design Automation Conference*, pages 43–47, 1992.

[28] Y.-T. Lai, M. Pedram, and S. B. K. Vrudhula. EVBDD-based algorithms for integer linear programming, spectral transformation and function decomposition. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 13:959–975, 1994.

[29] D. Long. Efficient implementation of an OBDD package, 1992.

[30] S. Malik, A. Wang, R. K. Brayton, and A. Sangiovanni-Vincentelli. Logic verification using binary decision diagrams in a logic synthesis environment. In *Proc. IEEE International Conference on Computer-Aided Design (Santa Clara, CA)*, pages 6–9, 1988.

[31] K. L. McMillan. *Symbolic Model Checking*. Kluwer Academic Publishers, 1993.

[32] Ch. Meinel. *Modified Branching Programs and Their Computational Power*, volume 370 of *Lecture Notes in Computer Science*. Springer, 1989. Reprinted by World Publishing Corporation, Beijing, 1991.

[33] Ch. Meinel and A. Slobodová. Speeding up variable ordering of OBDDs. In *Proc. International Conference on Computer Design (Austin, TX)*, 1997.

[34] Ch. Meinel, F. Somenzi, and T. Theobald. Linear sifting of decision diagrams. In *Proc. 34th ACM/IEEE Design Automation Conference (Anaheim, CA)*, pages 202–207, 1997.

[35] Ch. Meinel and Ch. Stangier. OBDD-based verification of communication protocols – methods for the verification of data link protocols. Technical Report 97-28, Universität Trier, 1997.

[36] Ch. Meinel and T. Theobald. Local encoding transformations for optimizing OBDD-representations of finite state machines. In *Proc. International Conference on Formal Methods in Computer-Aided Design (Palo Alto, CA)*, volume 1166 of *Lecture Notes in Computer Science*, pages 404–418. Springer, 1996.

[37] S. Minato. Zero-suppressed BDDs for set manipulation in combinatorial problems. In *Proc. 30th ACM/IEEE Design Automation Conference (Dallas, TX)*, pages 272–277, 1993.

[38] S. Minato. *Binary Decision Diagrams and Applications for VLSI CAD*. Kluwer Academic Publishers, 1996.

[39] S. Panda and F. Somenzi. Who are the variables in your neighborhood. In *Proc. IEEE International Conference on Computer-Aided Design (San José, CA)*, pages 74–77, 1995.

[40] S. Ponzio. A lower bound for integer multiplication with read-once branching programs. In *Proc. ACM Symposium on Theory of Computing*, 1995.

[41] R. Rudell. Dynamic variable ordering for ordered binary decision diagrams. In *Proc. IEEE International Conference on Computer-Aided Design (Santa Clara, CA)*, pages 42–47, 1993.

[42] H. P. Sharangpani and M. L. Barton. Statistical analysis of floating point flaw in the Pentium processor. Technical report, Intel, Nov. 30, 1994.

[43] D. Sieling and I. Wegener. Graph driven BDDs – a new data structure for Boolean functions. *Theoretical Computer Science*, 141:283–310, 1995.

[44] F. Somenzi. *CUDD: Colorado University Decision Diagram Package*. ftp://vlsi.colorado.edu/pub/, 1996.

[45] S. Tani, K. Hamaguchi, and S. Yajima. The complexity of the optimal variable ordering problems of shared binary decision diagrams. In *Proc. International Symposium on Algorithms and Computation*, volume 762 of *Lecture Notes in Computer Science*, pages 389–398. Springer, 1993.

[46] I. Wegener. *The Complexity of Boolean Functions*. John Wiley & Sons and Teubner-Verlag, 1987.

ECCC ISSN 1433-8092
http://www.eccc.uni-trier.de/eccc
ftp://ftp.eccc.uni-trier.de/pub/eccc
ftpmail@ftp.eccc.uni-trier.de, subject 'help eccc'