On the Influence of the State Encoding on OBDD-Representations of Finite State Machines*

Christoph Meinel, Thorsten Theobald**

FB IV – Informatik, Universität Trier, D–54286 Trier, Germany
{meinel,theobald}@ti.uni-trier.de

Abstract. Ordered binary decision diagrams are an important data structure for the representation of Boolean functions. Typically, the underlying variable ordering is used as an optimization parameter. When finite state machines are represented by OBDDs the state encoding can be used as an additional optimization parameter. In this paper, we analyze the influence of the state encoding on the OBDD-representations of counter-type finite state machines. In particular, we prove lower bounds, derive exact sizes for important encodings and construct a worst-case encoding which leads to exponential-size OBDDs.

1 Introduction

Ordered binary decision diagrams (OBDDs) introduced by Bryant [Bry86] provide an efficient graph-based data structure for Boolean functions (for a survey see [Bry92] or [Weg94]). The main optimization parameter of OBDDs is the underlying variable ordering. Many research efforts have tried to characterize the complexity of the relevant variable ordering problems [THY93, MS94, BW96] and to come up with efficient optimization algorithms for obtaining large size reductions without aiming at the global minimum [Rud93, BMS95]. Unfortunately, there are many important applications, in particular in the analysis of finite automata/finite state machines [CBM89, CM95], where this optimization technique reaches its limits.

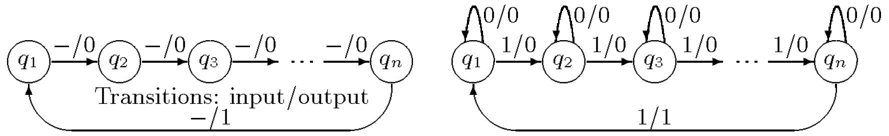
Quite recently, in [MT96] and [MST97] it was shown that in the context of finite state machines the state encoding can be used as an additional optimization parameter: In particular, the optimization techniques for finding suitable state encodings can be well integrated into existing algorithms for finding good variable orderings.

These optimization techniques immediately raise the question *in which extent* the choice of the state encoding can influence the OBDD-size at all. We will consider classes of counters which have a simple structure but which appear in numerous practical examples. For these classes, we analyze the relationship

* In Proc. MFCS'97, LNCS 1295, 408–417.
** Supported by DFG-Graduiertenkolleg "Mathematische Optimierung".

between the state encoding and the OBDD-size from a combinatorial point of view and give some precise answers concerning this relationship.

In particular, we consider the autonomous counter and the loop counter shown in Figure 1 [GDN92].



(a) Autonomous counter (b) Loop counter

Fig. 1. Counter types

In Figure 1 (a), the input symbol '$-$' means that this transition takes place for both input 1 and 0. Furthermore we analyze an acyclic counter which can be constructed out of the autonomous counter by deleting the "backward" edge from the last state to the first state. The results, although derived for a quite specific class of finite state machines, serve as reference examples for the task of finding re-encodings. The main contributions of this work are:

1. When fixing the variable ordering in a reasonable way, we derive the exact OBDD-sizes for the counters under some important encodings.
2. We present lower bounds for the OBDD-sizes of counter encodings which are very close to the derived OBDD-sizes for the standard encoding. These bounds underline the suitability of the standard encoding in this context.
3. We construct worst-case encodings which lead to exponential-size OBDDs and hence demonstrate the sensitivity in choosing the appropriate state encoding.
4. In general, we give some ideas for the analysis of an important and still growing topic, in which most of the previously known results are based on experimental work.

2 Preliminaries

2.1 Ordered binary decision diagrams

An *ordered binary decision diagram* (OBDD) is defined as a rooted directed acyclic graph with two sink nodes which are labeled 1 and 0. Each internal (= non-sink) node is labeled by an input variable x_i and has two outgoing edges, labeled 1 and 0 (in the diagrams the 0-edge is indicated by a dashed line). A linear variable ordering π is placed on the input variables. The variable occurrences on each OBDD-path have to be consistent with this ordering. An OBDD computes a Boolean function $f : \{0,1\}^n \to \{0,1\}$ in a natural manner: each assignment to the input variables x_i defines a unique path through the graph from the root to the sinks. The label of the sink gives the value of the function on that input.

The OBDD is called *reduced* if it does not contain any vertex v such that the 0-edge and the 1-edge of v leads to the same node, and it does not contain any distinct vertices v and v' such that the subgraphs rooted in v and v' are isomorphic. It is well-known that reduced OBDDs are a unique representation of Boolean functions $f : \{0,1\}^n \to \{0,1\}$ w.r.t. a given variable ordering [Bry86]. The *size* of an OBDD is the number of its internal nodes.

2.2 Finite state machines

Let $M = (Q, I, O, \delta, \lambda, Q_0)$ be a finite state machine, where Q is the set of states, I the input alphabet, O the output alphabet, $\delta : Q \times I \to Q$ the next-state function, $\lambda : Q \times I \to O$ the output function and Q_0 the set of initial states. As usual in VLSI design, all components of the state machine are assumed to be binary encoded. Let p be the number of input bits and n be the number of state bits. In particular, with $\mathbb{B} = \{0,1\}$, δ is a function $\mathbb{B}^n \times \mathbb{B}^p \to \mathbb{B}^n$. For a finite state machine M, the characteristic function of its *transition relation* is defined by

$$T(x_1, \ldots, x_n, y_1, \ldots, y_n, e_1, \ldots, e_p) = T(x, y, e) = \prod_{1 \le i \le n} (y_i \equiv \delta_i(x, e)).$$

Hence, the function T computes the value 1 for a triple (x, y, e) if and only if the state machine in state x and input e enters the state y.

In [BCL+94] and [CBM89] it was shown that the representation of a finite state machine by means of its transition relation goes well together with typical tasks for analyzing finite state machines like checking equivalence. All these applications are based on a reachability analysis. Hence, we can consider equivalently the transition relation of the underlying non-deterministic machine in which the inputs have been eliminated. In terms of Boolean manipulation this corresponds to an existential quantification over the inputs.

We want to remark that the most efficient implementations of this general concept work with a *partitioned* transition relation [BCL+94].

2.3 Variable orderings

For a finite state machine M, we derive the reduced OBDD-size for the characteristic function of its transition relation. This size is shortly called *OBDD-size of M*. The OBDD-size crucially depends on the chosen variable ordering. There are two variable orderings which often appear in connection with finite state machines: the *separated* ordering $x_1, \ldots, x_n, y_1, \ldots, y_n$ and the *interleaved* ordering $x_1, y_1, x_2, y_2, \ldots, x_n, y_n$ [ATB94].

For practical applications, the interleaved variable ordering is often superior to the separated ordering. If one considers for example a deterministic autonomous (i.e. input-independent) machine with a bijective next-state function, then the OBDD w.r.t. the separated ordering has exponential size. The reason is that after reading the variables $x_1, \ldots, x_n$ all induced subfunctions are different. The restriction to fix the variable ordering is reasonable in our context, as we want to analyze the effect of different state encodings.

3 The Lower Bound

We will investigate lower bounds for the OBDD-size of an autonomous counter with 2^n states where we keep the interleaved variable ordering fixed and vary over all $(2^n)!$ possible n-bit state encodings. For the first lower bound we use that the next-state function of the autonomous counter is bijective.

Theorem 1. *Let M_{2^n} be an autonomous finite state machine with 2^n states, n encoding bits and a bijective next-state function. The OBDD-size of M_{2^n} w.r.t. the interleaved variable ordering is at least $3n$.*

Proof. We show that for each $1 \leq i \leq n$, there are at least 3 nodes with label x_i or y_i. As the next-state function is bijective, each of the $2n$ variables appears on every path from the root to the 1-sink. There exists a node A labeled by x_i whose 1-edge leads to a sub-OBDD which does not represent the constant 0 (otherwise the next-state function cannot be a total function). In this sub-OBDD there exists a path from the root to the sink, and therefore the root must be labeled by y_i (see Figure 2 (a)). Analogously, there exists a node B labeled by x_i whose 0-edge leads to a sub-OBDD with root label y_i.

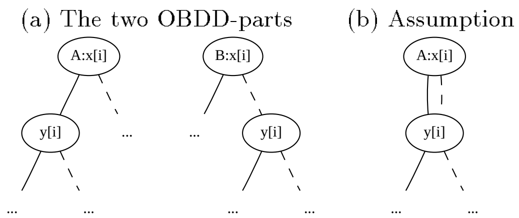


Fig. 2. Lower bound

If we assume that A and B are identical and that the two roots of the sub-OBDDs are identical, then node A can be reduced, see Figure 2 (b). The resulting OBDD has a path from the root to the 1-sink on which the variable x_i does not appear, a contradiction. □

The next theorem improves the lower bound by explicitly using the properties of a counter.

Theorem 2. *The OBDD-size of an autonomous counter with 2^n states, n encoding bits and interleaved variable ordering is at least $4n - 1$.*

Proof. First, we show that every variable $x_2, \ldots, x_n$ must appear at least twice in the OBDD. Assume for a contradiction that x_i appears only once. As the next-state function is bijective each 1-path in the OBDD goes through the vertex labeled by x_i. Every assignment $(x_1, y_1, \ldots, x_{i-1}, y_{i-1}) \in \{0,1\}^{2i-2}$ that leads to the x_i-node can be combined with every assignment $(x_i, y_i, \ldots, x_n, y_n) \in \{0,1\}^{2n-2i+2}$ that leads from the x_i-node to the 1-sink in order to construct a transition of the counter. Intuitively, these two groups of variables act independently. For each $(x_i, \ldots, x_n)$ there exists a vector $(y_i, \ldots, y_n)$ such that the assignment $x_i, y_i, \ldots, x_n, y_n$ leads from the x_i-node to the 1-sink. After exactly 2^{n-i} transitions of the form $x_i \ldots x_n \to y_i \ldots y_n$ the cycle w.r.t. the last $n-i+1$ variables is finished. Analogously, the first $i-1$ variables form a cycle of length 2^{i-1}. The resulting cycle is of length 2^n if and only if the least common multiple of 2^{i-1} and 2^{n-i+1} is 2^n which is impossible for $2 \le i \le n$. Analogously, we can show that each variable y_i must appear at least twice in the OBDD. Altogether there are at least $4n-1$ internal nodes. □

4 The Behavior of Important Encodings

4.1 The Standard Minimum-Length Encoding

First, we consider the standard encoding where 2^n states are represented by n bits, and the encoding of a state q_i is the binary representation of i. Let M_{2^n} be the autonomous counter with 2^n states under this encoding.

The *MSB-first* (most significant bit first) variable ordering is the interleaved variable ordering which reads the bits in decreasing significance. In contrast, the *LSB-first* variable ordering reads the bits in increasing significance.

Lemma 3. *For $n \ge 2$, the reduced OBDD for M_{2^n} w.r.t. the MSB-first variable ordering has $5n-3$ internal nodes.*

Proof. The idea is to use the OBDD for $M_{2^{n-1}}$ in order to construct the OBDD for M_{2^n}. Formally, this leads to a proof by induction. We show: The reduced OBDD is of the form like in Figure 3 (a) (in the sense that the shown nodes exist and are not pairwise isomorphic) with sub-OBDDs A and B and has exactly $5n-3$ nodes. The case $n=2$ can easily be checked.

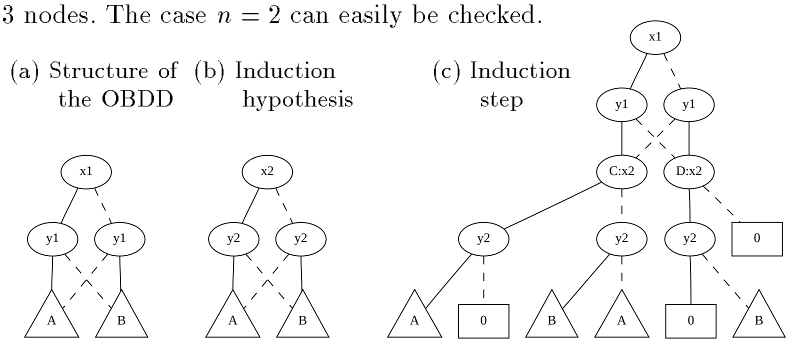


Fig. 3. MSB-first

Induction step: The OBDD for the $n-1$ bits $x_2, \ldots, x_n$ has the form like in Figure 3 (b). Let $|x|$ be the binary number that is represented by a bit-string $x \in \{0,1\}^+$. With this notation we have

A: leads to the 1-sink if and only if $|y_3 \ldots y_n| = |x_3 \ldots x_n| + 1$, $x_3 \ldots x_n \ne 11 \ldots 1$.
B: leads to the 1-sink if and only if $x_3 = \ldots = x_n = 1$, $y_3 = \ldots = y_n = 0$.

We construct the reduced OBDD for M_{2^n} like in Figure 3 (c). The subfunctions rooted in C and D have the following meanings:

C: leads to the 1-sink if and only if $|y_2 \ldots y_n| = |x_2 \ldots x_n|+1$, $x_2 \ldots x_n \neq 11 \ldots 1$.
D: leads to the 1-sink if and only if $x_2 = \ldots = x_n = 1$, $y_2 = \ldots = y_n = 0$.

It can easily be checked that all the subfunctions rooted in the new invented nodes are pairwise different. Therefore $\text{size}(M_{2^n}) = \text{size}(M_{2^{n-1}}) - 3 + 8 = \text{size}(M_{2^{n-1}}) + 5$. □

Analogously, it can be shown that for $n \geq 2$, the OBDDs for the LSB-first ordering lead to the same number of nodes as for MSB-first although the OBDDs are not isomorphic. The main reason for this equality is the fact that in both OBDDs there is only one bit of information that has to be passed from the level of y_{i-1} to the level of x_i for $2 \leq i \leq n$.

It is quite remarkable that these OBDD-sizes for the standard encoding nearly meet the lower bound of $4n-1$. We conjecture that the standard encoding is even optimal. In order to prove a better lower bound, one might have to establish much more sophisticated communication complexity arguments which connect local OBDD-properties with the global single-cycle-property of a counter.

4.2 The Gray Encoding

Another important minimum-length encoding is the one where the encoding of state i is the Gray code representation of i. The Gray code has the property that all successive code words differ in only one bit. The n-bit Gray code can be constructed by reflecting the $(n-1)$-bit Gray code. To all the new codewords, a leading 1 is added.

By analogous constructions like in the previous subsection, it can be verified that for $n \geq 2$ the OBDD-size of the Gray-encoded autonomous counter with MSB- or LSB-first variable ordering is $10n - 11$. However, the OBDDs are not isomorphic. The constructions in both cases are based on the reflecting property of the Gray code. The essential reason why the Gray encoding has a bigger factor than the standard encoding is the reflecting property of the Gray code which does not allow an immediate use of the OBDD for $M^G_{2^n-1}$ when constructing the OBDD for $M^G_{2^n}$.

5 A Worst-Case Encoding

Of course, it is easy to construct autonomous finite state machines with bijective next-state function together with an encoding which have exponential OBDD-size w.r.t. the interleaved variable ordering. Things become more difficult if one wants to construct the encoding of an autonomous *counter* that leads to exponential OBDD-size. In the following we construct an encoding in which the first nodes in the OBDD labeled by $x_1, y_1, \ldots, x_{n/2}, y_{n/2}$ and their outgoing edges lead to a complete tree. The number of nodes with label $x_{n/2+1}$ will therefore be 2^n and the OBDD will have more than 2^{n+1} nodes.

There are 2^n different assignments to the *leading variables* $x_1, y_1, \ldots, x_{n/2}, y_{n/2}$ and 2^n transitions in the finite state machine. To ensure that the leading variables in the OBDD generate a complete tree, we construct the next-state function in the following way:

1. For each assignment to the leading variables, there exists exactly one assignment to the *tail variables* $x_{n/2+1}, y_{n/2+1}, \ldots, x_n, y_n$ which leads to the 1-sink in the OBDD.
2. Each of the 2^n assignments to the tail variables appears exactly once in the construction.

For the construction of the worst-case counter, we build up two tables, the *transition table* and the *tail table.*

Transition table: This table consists of 2^n rows. Each row describes one transition $x_1 \ldots x_n \rightarrow y_1 \ldots y_n$ of the counter. Initially, some of the bits are already set to 1 or 0: All the 2^n assignments for the leading variables $x_1, y_1, \ldots, x_{n/2}, y_{n/2}$ are inserted in the table in the following way: In row i of the table, $0 \leq i < 2^n$, the integer i is represented in binary by the bits $x_1 \ldots x_{n/2} y_1 \ldots y_{n/2}$. The bits of the tail variables are not affected by this initial construction – these entries are marked by stars and will be filled during the construction. The transition table after this initial step is shown in Figure 4.

There is an additional entry called *visited* in each row which helps to keep track which rows of the table have already been filled during the construction algorithm. Initially, every entry in the *visited* column is set to 'FALSE'.

$x_1 \ldots x_{n/2}$	$x_{n/2+1} \ldots x_n$	$y_1 \ldots y_{n/2}$	$y_{n/2+1} \ldots y_n$	*visited*
$00 \ldots 00$	$\star$	$00 \ldots 00$	$\star$	FALSE
$00 \ldots 00$	$\star$	$00 \ldots 01$	$\star$	FALSE
$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$
$00 \ldots 00$	$\star$	$11 \ldots 11$	$\star$	FALSE
$00 \ldots 01$	$\star$	$00 \ldots 00$	$\star$	FALSE
$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$
$11 \ldots 11$	$\star$	$11 \ldots 11$	$\star$	FALSE

Fig. 4. Initial structure of the transition table

Tail table: This table consists of 2^n rows which contain the 2^n different assignments for the tail variables $x_{n/2+1}, \ldots, x_n, y_{n/2+1}, \ldots, y_n$. In each row there is an additional entry called *used* which helps to ensure that each assignment is only used once during the construction. Initially, every entry in the *used* column is set to 'FALSE'. The tail table is shown in Figure 5.

$x_{n/2+1} \ldots x_n$	$y_{n/2+1} \ldots y_n$	used
$00 \ldots 00$	$00 \ldots 00$	FALSE
$00 \ldots 00$	$00 \ldots 01$	FALSE
$\vdots$	$\vdots$	$\vdots$
$11 \ldots 11$	$11 \ldots 11$	FALSE

Fig. 5. Initial structure of the tail table

The entries of the transition table which are marked with a star are filled during the construction. The task is to put each assignment for the tail variables into one row of the transition table in such a way that the induced transitions $x_1 \dots x_n \to y_1 \dots y_n$ form a cycle of length 2^n. Note that this construction guarantees that the two properties are satisfied. We use the algorithm of Figure 6.

1. For each row of the transition table: Set $x_{n/2+1} \dots x_n := y_1 \dots y_{n/2}$.
2. Set the present row to the top row of the transition table.
3. While *visited* in the present row is set to 'FALSE'
   (a) Set the *visited* entry in the present row of the transition table to 'TRUE'.
   (b) Choose the maximal (w.r.t. the represented binary number) assignment for the tail variables from the tail table which matches the assignment for $x_{n/2+1} \dots x_n$ in the present row and whose *used* entry is set to 'FALSE'.
   (c) Set the *used* entry for the chosen assignment in the tail table to 'TRUE'.
   (d) Set $y_{n/2+1} \dots y_n$ in the transition table to the chosen assignment for $y_{n/2+1} \dots y_n$.
   (e) Let R be the row in the transition table in which the assignment for $x_1 \dots x_n$ is identical with the assignment for $y_1 \dots y_n$ in the present row.
   (f) Set the present row to row R.

Fig. 6. Construction of the worst-case counter

Claim. The finite state machine that is constructed by the algorithm is a counter, i.e. the cycle has length 2^n.

The claim follows from three statements:

1. In step 3.(a) there exists an assignment in the tail table with the desired properties.
2. If the while-condition in step 3 is not satisfied (i.e. the while-loop terminates), then $x_1 \dots x_n = 00 \dots 0$ in the present row of the transition table.
3. When the while-loop terminates, all 2^n assignments for the tail variables have been marked as used.

Before proving the statements we show why the claim follows from them: From statement 1 it follows that step 3.(b) is well-defined in each processing of the loop body. Due to the finiteness of the tail table and step 3.(c), the algorithm terminates. Statements 2 and 3 guarantee that the construction builds a cycle of length 2^n.

Proofs of the statements:
1. The *visited* column of the transition table guarantees that each row of the transition table is at most once the present row – otherwise, the while-loop immediately terminates. For a *fixed* assignment X to $x_{n/2+1} \dots x_n$ the number of rows in the transition table in which the assignment X appears is $2^{n/2}$. Therefore there are at most $2^{n/2}$ situations during the run of the algorithm in which an assignment is needed in step 3.(b) that extends X. On the other hand there are also $2^{n/2}$ assignments for the tail variables in the tail table which extend X.

Therefore in each processing of step 3.(b), there is at least one previously unused suitable assignment left.

2. Consider the assignment for $y_1 \dots y_n$ in the present row after step 3.(d). Due to step 1 of the algorithm this assignment is equal to $x_{n/2+1} \dots x_n y_{n/2+1} \dots y_n$ in the present row. Due to step 3.(d) and (e) this assignment determines the assignment for $x_1 \dots x_n$ in the new present row. It follows: Whenever a row with a given bit sequence for $x_1 \dots x_n$ becomes the present row in step 3.(f), this bit sequence has just been marked as used in the tail table in step 3.(c). This makes it impossible that a row becomes the present row in step 3.(f) more than once.

After the first processing of the loop body, the top row is marked as visited, but the bit sequence $00\dots0$ has not been marked as used in the tail table. Therefore the only possibility to enter a row whose *visited* entry is already set to 'TRUE' is to enter the top row.

3. Due to statement 2, the last chosen assignment for the tail variables is $00\dots0$. As in step 3.(b) the maximal assignment is chosen, all $2^{n/2}$ assignments of the form $00\dots0\dots \in 0^{n/2}\{0,1\}^{n/2}$ must have already been used now. Therefore, due to the bit shifting in step 1 and step 3.(e), all assignments $\dots00\dots0 \in \{0,1\}^{n/2}0^{n/2}$ in the tail table must have already been used now. Using again that in 3.(b) the maximal assignment is chosen, it follows that all 2^n assignments for the tail variables have been used when the while-loop terminates. □

Remark. The first paragraph of this section implies that the constructed OBDD has at least 2^n subfunctions of the form $f_{x_1=a_1,\dots,x_{n/2}=a_{n/2},y_1=b_1,\dots,y_{n/2}=b_{n/2}}$ for $a_1,\dots,a_{n/2},b_1,\dots,b_{n/2} \in \{0,1\}$. Hence, the worst-case construction also holds for a bigger class of variable orderings than only the fixed interleaved ordering $x_1,y_1,\dots,x_n,y_n$: Namely, it also holds for all variable orderings in which all variables from the upper half come from the set $\{x_1,\dots,x_{n/2},y_1,\dots,y_{n/2}\}$ and all variables from the lower half come from the set $\{x_{n/2+1},\dots,x_n,y_{n/2+1},\dots,y_n\}$.

6 Related Topologies

With similar techniques most of the results that have been proven for the autonomous counter can also be established for the loop counter and the acyclic counter. The lower bound of $4n-1$ can be transferred to the acyclic counter. The construction can be slightly modified to prove a $4n-2$ lower bound for the loop counter which becomes non-deterministic after the elimination of the inputs. The worst-case construction for the autonomous counter can also be used to construct a worst-case encoding for the acyclic counter.

7 Conclusion and Open Questions

We have given some precise results on the relation between state encodings and the size of OBDD-representations. These results show a strong dependence and therefore underline the importance of the OBDD-optimization by re-encoding techniques [MT96].

The general open problems are to analyze non-linear topologies of finite state machines, more general variable orderings and the behavior during reachability analysis in the context of state encodings. For these tasks, the presented results and analysis techniques form the basic ingredients.

References

[ATB94] A. Aziz, S. Taziran, and R. K. Brayton. BDD variable ordering for interacting finite state machines. In *Proc. 31st ACM/IEEE Design Automation Conference*, 283–288, 1994.

[BCL+94] 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 CAD*, 13:401–424, 1994.

[BMS95] J. Bern, Ch. Meinel, and A. Slobodová. Global rebuilding of OBDDs - avoiding memory requirement maxima. In *Proc. Computer-Aided Verification, Lecture Notes in Computer Science* 939, 4–15. Springer, 1995.

[Bry86] R. E. Bryant. Graph-based algorithms for Boolean function manipulation. *IEEE Transactions on Computers*, C–35:677–691, 1986.

[Bry92] R. E. Bryant. Symbolic Boolean manipulation with ordered binary decision diagrams. *ACM Computing Surveys*, 24(3):293–318, 1992.

[BW96] B. Bollig and I. Wegener. Improving the variable ordering of OBDDs is NP-complete. *IEEE Transactions on Computers*, 45:993–1002, 1996.

[CBM89] 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, Lecture Notes in Computer Science* 407, 365–373. Springer, 1989.

[CM95] 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.

[GDN92] A. Ghosh, S. Devadas, and A. R. Newton. *Sequential Logic Testing and Verification*. Kluwer Academic Publishers, 1992.

[MS94] Ch. Meinel and A. Slobodová. On the complexity of constructing optimal ordered binary decision diagrams. In *Proc. Mathematical Foundations in Computer Science, Lecture Notes in Computer Science* 841, 515–524, 1994.

[MST97] Ch. Meinel, F. Somenzi, and T. Theobald. Linear sifting of decision diagrams. To appear at 34th ACM/IEEE Design Automation Conference (Anaheim, CA), 1997.

[MT96] Ch. Meinel and T. Theobald. Local encoding transformations for optimizing OBDD-representations of finite state machines. In *Proc. International Conference on Formal Methods in CAD (Palo Alto, CA), Lecture Notes in Computer Science* 1166, 404–418. Springer, 1996.

[Rud93] R. Rudell. Dynamic variable ordering for ordered binary decision diagrams. In *Proc. IEEE International Conference on CAD*, 42–47, 1993.

[THY93] 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 '93, Lecture Notes in Computer Science* 762, 389–398. Springer, 1993.

[Weg94] I. Wegener. Efficient data structures for Boolean functions. *Discrete Mathematics*, 136:347–372, 1994.

This article was processed using the LaTeX macro package with LLNCS style