Local Encoding Transformations for Optimizing OBDD-Representations of Finite State Machines*

Christoph Meinel, Thorsten Theobald*
FB IV – Informatik, Universität Trier
D – 54286 Trier, Germany

Abstract

Ordered binary decision diagrams are the state-of-the-art representation of switching functions. In order to keep the sizes of the OBDDs tractable, heuristics and dynamic reordering algorithms are applied to optimize the underlying variable order. When finite state machines are represented by OBDDs the state encoding can be used as an additional optimization parameter. In this paper, we analyze local encoding transformations which can be applied dynamically. First, we investigate the potential of re-encoding techniques. We then propose the use of an XOR-transformation and show why this transformation is most suitable among the set of all encoding transformations. Preliminary experimental results illustrate that the proposed method in fact yields a reduction of the OBDD-sizes.

1 Introduction

Ordered binary decision diagrams (OBDDs) which have been introduced by Bryant [Bry86] provide an efficient graph-based data structure for switching functions. The main optimization parameter of OBDDs is the underlying variable order. In order to find a good order two techniques were applied so far: the use of heuristics which try to exploit the structure of a circuit representation (see e.g. [MWBS88]), and dynamic reordering techniques [Rud93]. Unfortunately, there are many applications, in particular in the field of sequential analysis, where these two optimization techniques for OBDDs reach their limits. Hence, one essential problem in logic synthesis and verification is to find new techniques to minimize OBDDs in these applications.

When OBDDs are used to represent finite state machines the OBDD-size does not only depend on the variable order but also on the state encoding. For a fixed state encoding there are many finite state machines whose OBDD-representations are large w.r.t. all variable orders [ATB94]. Therefore the relationship between the OBDD-size and the state encoding becomes of increasing interest, see e.g. [QCC+95, TM96]. The importance of this relationship is underlined by recent ideas to apply heuristic state re-encoding techniques to speed up a verification process between similar-structured finite state machines [QCC+95].

The underlying general problem of all these efforts is the following: Given the OBDDs for the next-state and output functions of a finite state machine – if one is interested in the input/output behavior of the machine, in how far can the internal state encoding be exploited to minimize OBDD-sizes ? Our approach targets at applying *local* encoding transformations, i.e. transformations which involve only a limited number of encoding bits. These

*in: Proceedings FMCAD'96, LNCS
*Supported by DFG-Graduiertenkolleg "Mathematische Optimierung".

transformations can be interpreted as a re-encoding of the symbolic states. The aim is to minimize OBDD-sizes by the iterated application of local transformations. The advantage of this approach is that the costs for applying these transformations are still manageable.

The paper is structured as follows: We begin with recalling some important definitions and point out the principle potential of state re-encodings w.r.t. OBDD-sizes. Then, in Section 4, we analyze the advantages of local encoding transformations. In Section 5 we propose the application of the *XOR-transformation* and show why this transformation is most promising among the set of all encoding transformations. At the end of the paper we describe an implementation of this transformation and give some first experimental results which illustrate the positive impact of the presented ideas.

2 Preliminaries

2.1 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, all components of the state machine are assumed to be binary encoded. Let p be the number of input bits, n be the number of state bits and m be the number of output bits. In particular, with $\mathbb{B} = \{0, 1\}$, δ is a function $\mathbb{B}^n \times \mathbb{B}^p \to \mathbb{B}^n$, λ is a function $\mathbb{B}^n \times \mathbb{B}^p \to \mathbb{B}^m$, and Q_0 is a subset of $\mathbb{B}^n$.

2.2 Binary decision diagrams

Ordered binary decision diagrams (OBDDs) [Bry86] are rooted directed acyclic graphs representing switching functions. Each OBDD has 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 1-edge is indicated by a solid line and the 0-edge by a dotted line). A linear variable order π is placed on the input variables. The variable occurrences on each OBDD-path have to be consistent with this order. An OBDD computes a switching function $f : \mathbb{B}^n \to \mathbb{B}$ in a natural manner: each assignment to the input variables x_i defines a unique path through the graph from the root to a sink. The label of this sink defines 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 switching functions $f : \mathbb{B}^n \to \mathbb{B}$ with respect to a given variable order [Bry86]. The *size* of an OBDD is the number of its nodes. Several functions can be represented by a multi-rooted graph called *shared OBDD*. In the following, all next-state and output functions are represented by a shared OBDD.

2.3 The transition relation

For a finite state machine M, the characteristic function of its *transition relation* is defined by

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

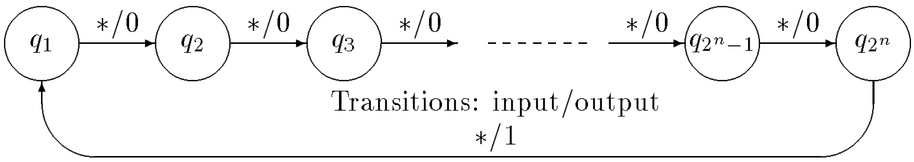
Hence, the function T computes the value 1 for a triple (x, y, e) if and only if the state machine in state x and input e enters the state y. The variables $x_1, \ldots, x_n$ are called *current-state variables* and the variables $y_1, \ldots, y_n$ are called *next-state variables.*

3 Motivation: The Potential of Re-encoding

In order to demonstrate how much the size of an OBDD-representation depends on the choice of the state encoding, let us consider an *autonomous counter*, a finite state machine with a very simple structure:

Example 1. An *autonomous counter* (see for example [GDN92]) with 2^n states $q_1, \ldots, q_{2^n}$ is an autonomous (i.e. input-independent) finite state machine with $\delta(q_i) = q_{i+1}$, $1 \leq i \leq 2^n - 1$, and $\delta(q_{2^n}) = q_1$.

Figure 1: Autonomous counter



The following theorem shows that almost all encodings for the autonomous counter lead to exponential-size OBDDs, even for their optimal variable order.

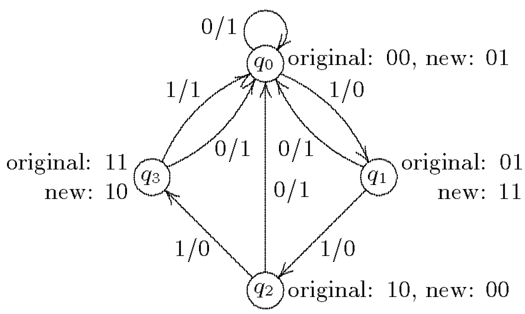
Theorem 1. *Let $e(n)$ denote the number of n-bit encodings for the autonomous counter with 2^n states which lead to a (shared) OBDD of size at most $2^n/n$ w.r.t. their optimal variable order. Let $a(n) = (2^n)!$ denote the number of all possible counter encodings. Then the ratio $e(n)/a(n)$ converges to zero as n tends to infinity.*

The proof of the theorem can be found in the appendix. It is based on ideas of [LL92] and classical counting results of Shannon [Sha49]. An analogous result can be established for the characteristic function of the transition relation and its OBDD-size.

Definition 2. An *encoding transformation*, shortly called a *re-encoding*, is a bijective function $\rho : \mathbb{B}^n \to \mathbb{B}^n$ that transforms the given state encoding to a new encoding. (For an example see Figure 2.) If a state s is encoded by a bit-string $c \in \mathbb{B}^n$, then its new encoding is $\rho(c)$.

Figure 2: Encoding transformation $\rho(c_1, c_2) = (c_2, \overline{c_1})$

symbolic state	original encoding	new encoding
q_0	00	01
q_1	01	11
q_2	10	00
q_3	11	10



This modification of the internal state encoding does not modify the input/output behavior of the state machine. The machine with the new encoding is denoted by

$M' = (Q', I, O, \delta', \lambda', Q'_0)$. Its encoded next-state function, output function and set of initial states are computed as follows:

$$\begin{aligned} \delta'(s,e) &= \rho(\delta(\rho^{-1}(s),e)), \\ \lambda'(s,e) &= \lambda(\rho^{-1}(s),e), \\ Q'_0 &= \rho(Q_0), \end{aligned} \tag{1}$$

where $\delta, \delta' : \mathbb{B}^n \times \mathbb{B}^p \to \mathbb{B}^n$, $\lambda, \lambda' : \mathbb{B}^n \times \mathbb{B}^p \to \mathbb{B}^m$, and $Q_0, Q'_0 \subset \mathbb{B}^n$.

The transition relation of the re-encoded machine M' can be obtained from the transition relation of M as follows:

Lemma 3. *Let $T(x,y,e)$ be the characteristic function of the transition relation of M. Then the characteristic function $T'(x,y,e)$ of the transition relation of M' is*

$$\prod_{i=1}^{n} \left(\rho_i^{-1}(y) \equiv \delta_i(\rho^{-1}(x)) \right).$$

Therefore $T'(x,y,e)$ can be obtained from $T(x,y,e)$ by the substitutions $y_i \mapsto \rho_i^{-1}(y)$ and $x_i \mapsto \rho_i^{-1}(x)$, $1 \le i \le n$.

Proof. The lemma is a consequence of the following equivalences:

$$\begin{aligned} T'(x,y,e) = 1 \quad &\Longleftrightarrow \quad \forall i \; y_i = \rho_i(\delta(\rho^{-1}(x))) \quad \Longleftrightarrow \quad y = \rho(\delta(\rho^{-1}(x))) \\ &\Longleftrightarrow \quad \rho^{-1}(y) = \delta(\rho^{-1}(x)) \quad \Longleftrightarrow \quad \prod_{i=1}^{n} \left(\rho_i^{-1}(y) \equiv \delta_i(\rho^{-1}(x)) \right) = 1 \end{aligned}$$

□

Example 1 (ctd.). The large potential of re-encoding techniques can now be demonstrated at the example of the autonomous counter: There exists an encoding such that the transition relation of the autonomous counter with 2^n states and n encoding bits has at most $5n - 1$ nodes [TM96] even if the variable order is fixed to $x_1, y_1, \ldots, x_n, y_n$. Hence, for each given encoding of a finite state machine, there exists a re-encoding which leads to OBDDs of linear size. As according to Theorem 1 most encodings lead to OBDDs of exponential size, the gain between the original OBDD and the OBDD after a suitable re-encoding is exponential in most cases. The aim now is to *find* the suited re-encoding that leads to small OBDD-sizes.

4 Local Re-encodings

In the previous section we have shown that re-encodings may have a large impact on the OBDD-size. It is possible that the OBDD becomes much smaller, but in the case of a badly chosen re-encoding the OBDD could even become much larger. This situation is comparable to the problem of finding a good variable order for an OBDD. When changing the variable order of an OBDD, the graph may become much smaller in the best case or much larger in the worst case. This sensitivity is the main reason why it is hard to *find* a good re-encoding or a good variable order.

For the effective *construction* of good variable orders it has turned out that the most efficient strategies are based on local exchanges of variables. The presently best strategies for finding good variable orders dynamically are based on the *sifting algorithm* of Rudell [Rud93, PS95]. The main principle of this algorithm is based on a subroutine which finds the optimum position for one variable, if all other variables remain fixed. This subroutine is repeated for each variable. There are two main reasons why this strategy works so efficiently:

Bounded size alteration: If one variable x_i is moved to another position in the OBDD, the size of the OBDD cannot change arbitrarily much, in particular it cannot explode. [BLW95] have shown the following theorem:

Theorem 4. *Let P be an OBDD. If a variable x_i is moved to a later position in the variable order, then the size of the resulting OBDD P' satisfies*

$$size(P)/2 \leq size(P') \leq size(P)^2.$$

If a variable x_i is moved to an earlier position in the variable order, then the size of the resulting OBDD P' even satisfies the relation

$$size(P)^{1/2} \leq size(P') \leq 2\ size(P). \qquad \square$$

Practical studies have shown that in most cases the resulting sizes are even far below the worst-case estimations. Hence, each application of the above mentioned subroutine keeps the size of the OBDD manageable. However, this bounded size alteration for the subroutine does not mean that the optimization potential is limited. The iteration of this subroutine allows to minimize OBDDs very effectively.

Continuity: The procedure for moving a variable x_i to a different position in the order works continuously: During this process only the variables between the original and the new position of x_i are involved, and all nodes labeled by the remaining variables remain untouched. In particular, the time complexity of this operation is very small if x_i is moved to an adjacent position, and it increases with the number of variables between the original and the new position of x_i in the variable order.

In the case of re-encoding the situation is analogous. It seems to be very hard to find the right *global* re-encoding, whereas it is very promising to combine and iterate operations with restricted local effect. Our approach to construct *local* re-encodings $\rho : \mathbb{B}^n \to \mathbb{B}^n$ is to keep most of the bits fixed (i.e. $\rho_i(x) = x_i$) and vary only on a small number of bits. In particular, if we vary only 2 bits, we will speak of *two-bit re-encodings.* In this case it follows from the worst-case bounds for the synthesis and the substitution of OBDDs that the OBDDs remain polynomial.

Example 2. The *exchange variables* re-encoding $\rho_{i,j} : \mathbb{B}^n \to \mathbb{B}^n$, $1 \leq i, j \leq n$, is defined as follows (The following definition shows the case $i < j$, the case $i > j$ is defined analogously):

$$(q_1, \ldots, q_n) \mapsto (q_1, \ldots, q_{i-1}, q_j, q_{i+1}, \ldots, q_{j-1}, q_i, q_{j+1}, \ldots, q_n)$$

Obviously, the *exchange variables* re-encoding has the same effect on the next-state functions as exchanging the state variables x_i and x_j in the variable order. From all the $(2^n)!$ possible encoding transformations $n!$ can be generated by the iterated application of this transformation type. The inverse mapping ρ^{-1} from Equation 1 does not affect the size of the resulting OBDDs as this mapping only causes the renaming of the two functions δ_i and δ_j.

Note, that the transformation which exchanges the encodings of two fixed states may not be seen as a local operation, although the transformation seems to be very simple.

5 The XOR-Transformation

We will now propose *XOR-transformations.* This transformation is a local re-encoding which operates on two bits.

Definition 5. An *XOR-transformation* $\rho_{i,j}$, $1 \le i, j \le n$, is defined by

$$(q_1, \ldots, q_n) \mapsto (q_1, \ldots, q_{i-1}, q_i \oplus q_j, q_{i+1}, \ldots, q_n)$$

Short notation: $q_i \mapsto q_i \oplus q_j$. For an example see Figure 3.

Figure 3: The XOR-transformation $q_1 \mapsto q_1 \oplus q_2$

original encoding		new encoding	
q_1	q_2	q_1^{new}	q_2^{new}
0	0	0	0
0	1	1	1
1	0	1	0
1	1	0	1

Indeed, XOR-transformations provide a solid basis for the design of effective re-encodings due to the following facts:

1. The number of possible re-encodings generated by the iterated application of XOR-transformations is much larger than the number of possible variable orders. Thus, XOR-transformations considerably enlarge the optimization space. On the other hand, the number of these re-encodings is much smaller than the number of all re-encodings which makes it possible to keep the search space manageable.

2. The size influence of this transformation is bounded in a reasonable way like in the case of local changes in the variable order.

3. A precise analysis even shows that an XOR-transformation contains the same asymmetry as the movement of one variable in the variable order. Namely, the bounds for the effect of a transformation $x_i \mapsto x_i \oplus x_j$ depends on the position of x_i and x_j in the variable order.

4. The XOR-transformation is in fact the only new possible re-encoding on two variables.

5. The XOR-transformation can be implemented efficiently like an exchange of two variables in the order.

In the following subsection we will prove these statements.

5.1 Enumeration results

The following combinatorial statements characterize the size of the optimization space provided by the use of XOR-transformations.

Lemma 6. *(1) Let $t(n)$ be the number of possible encoding transformations that can be generated by the iterated application of XOR-transformations. It holds*

$$t(n) = \prod_{i=0}^{n-1} (2^n - 2^i).$$

(2) The quotient of $t(n)$ and the number of all possible encoding transformations converges to zero as n tends to infinity.
(3) Let $v(n) := (2n)!$ denote the number of possible variable orders for the transition relation of an autonomous finite state machine with n state bits. The fraction $v(n)/t(n)$ converges to zero as n tends to infinity.

Statement 3 says that in the case of autonomous state machines, there are much more encoding transformations generated by XOR-transformations than variable orders for the transition relation. This relation also holds when the number of input bits is fixed and the number of state bits becomes large.

Proof. (1) Obviously, each XOR-transformation is a regular linear variable transformation over the field $\mathbb{Z}_2$. Moreover, the XOR-transformations provide a generating system for all regular linear variable transformations. Therefore the state encodings which can be obtained by iterated XOR-transformations are in 1-1-correspondence with the regular $n \times n$-matrices over $\mathbb{Z}_2$.

The number of these matrices can be computed as follows: The first row vector b_1 can be chosen arbitrarily from $\mathbb{Z}_2^n \setminus 0$. The i-th row vector b_i, $2 \leq i \leq n$, can be chosen arbitrarily from

$$\mathbb{Z}_2^n \setminus \Big\{ \sum_{j=1}^{i-1} \lambda_j b_j \ : \ \lambda_1, \ldots, \lambda_{i-1} \in \mathbb{Z}_2 \Big\}.$$

These are $2^n - 2^{i-1}$ possibilities for the vector b_i. This proves the claimed number.
(2) This statement follows from the relation

$$\frac{t(n)}{(2^n)!} \leq \frac{2^{n^2}}{(2^n)!} \leq \frac{(n^2)!}{(2^n)!} \to 0 \quad \text{as} \quad n \to \infty.$$

(3) It holds

$$\frac{(2n)!}{t(n)} = \frac{2n(2n-1)}{2^n - 2^0} \cdot \frac{(2n-2)(2n-3)}{2^n - 2^1} \cdots \frac{2 \cdot 1}{2^n - 2^{n-1}} \to 0 \quad \text{as} \quad n \to \infty.$$

□

In particular, the number of possible encoding transformations which can be generated by the iterated application of XOR-transformations is smaller than 2^{n^2} which is exactly the number of all $n \times n$-matrices over $\mathbb{Z}_2$.

It follows from the previous proof that all exchanges of two state variables can be simulated by the iterated application of XOR-transformations.

5.2 Bounded size alteration

Let ρ be the XOR-transformation $q_i \mapsto q_i \oplus q_j$. Then the inverse transformation is defined by

$$\rho^{-1} : q_i \mapsto q_i \oplus q_j,$$

i.e. we have $\rho = \rho^{-1}$. The effect of $\rho(\delta(\rho^{-1}(\cdot)))$ in Equation 1 can be split into two parts:

1. Substitute the current-state variable x_i by $x_i \oplus x_j$.

2. Replace the function δ_i by $\delta_i \oplus \delta_j$.

It does not matter which of these two steps is executed first.

Lemma 7. *Let $P_1, \ldots, P_n$ be the OBDDs for $\delta_1, \ldots, \delta_n$ and $P'_1, \ldots, P'_n$ be the OBDDs after the application of the XOR-transformation $q_i \mapsto q_i \oplus q_j$. The following holds:*

$$\begin{array}{rcccll} \Omega(size(P_i)^{1/2}/size(P_j)^{1/2}) & \leq & size(P'_i) & \leq & \mathcal{O}(size(P_i)^2 \cdot size(P_j)^2), & \\ \Omega(size(P_k)^{1/2}) & \leq & size(P'_k) & \leq & \mathcal{O}(size(P_k)^2), & k \neq i. \end{array}$$

The upper bound immediately follows from the facts that the substitution of an OBDD P_2 into one variable of an OBDD P_1 leads to an OBDD of size at most $\mathcal{O}(\text{size}(P_1)^2 \cdot \text{size}(P_2))$, and that the operation $P_1 \oplus P_2$ leads to an OBDD of size at most $\mathcal{O}(\text{size}(P_1) \cdot \text{size}(P_2))$. For the lower bounds it suffices to observe $\rho = \rho^{-1}$.

In case of the transition relation both the current-state variables and the next-state variables have to be substituted. This leads to the result

$$\Omega(\text{size}(P)^{1/4}) \leq \text{size}(P') \leq \mathcal{O}(\text{size}(P)^4),$$

where P and P' are the original and the re-encoded OBDD for the transition relation, respectively.

5.3 Stronger bounds

For a more refined analysis of the XOR-transformation we use the following theorem from [SW93]. In particular, we will refine the analysis for the substitution of a variable x_i by $x_i \oplus x_j$ in an OBDD.

Theorem 8. *The reduced OBDD representing f with the variable order $x_1, \ldots, x_n$ contains as many x_i-nodes as there are different functions f^S, $S \subset \{1, \ldots, i-1\}$, depending essentially on x_i (i.e. $f^S_{x_i=0} \neq f^S_{x_i=1}$). Here $f^S = f_{x_1=a_1,\ldots,x_{i-1}=a_{i-1}}$ where $a_j = 0$ if $j \notin S$, and $a_j = 1$ otherwise.* □

Let s_k be the number of nodes labeled by x_k in the OBDD P and s^*_k be the number of nodes labeled by x_k in the OBDD P' which is the result of the transformation.

Theorem 9. *The size of an OBDD w.r.t. the variable order $x_1, \ldots, x_n$ after the application of the substitution $x_i \mapsto x_i \oplus x_j$ is bounded from above by*

$$2(s_1 + \ldots + s_{i-1}) + 1 + s_{i+1} + \ldots + s_n + 2 \quad \textit{in case } j < i$$

and by

$$s_1 + \ldots + s_i + (s_{i+1} + \ldots + s_{j-1} + s_{j+1} + \ldots + s_n + 2)^2 \quad \textit{in case } i < j.$$

The proof of this theorem can be found in the appendix. It applies ideas from [BLW95], in which local changes in the variable order are analyzed.

Corollary 10. *Let P be an OBDD and P' the resulting OBDD after the substitution $x_i \mapsto x_i \oplus x_j$. Then*

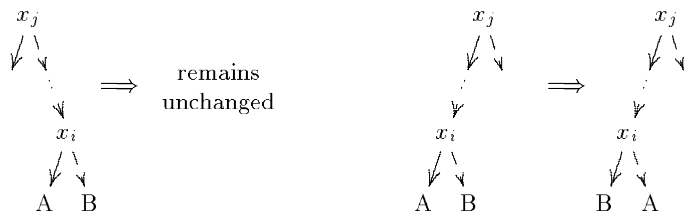
$$\begin{array}{rcccll} size(P)/2 & \leq & size(P') & \leq & 2\ size(P) & \text{if } x_j < x_i, \\ size(P)^{1/2} & \leq & size(P') & \leq & size(P)^2 & \text{if } x_i < x_j. \end{array}$$ □

The analogy between the behavior of the XOR-transformation and the local changes in the variable order recommends to use XOR-transformations for the optimization of OBDD-sizes.

The XOR-transformation $x_i \mapsto x_i \oplus x_j$ for $j < i$ and $j+1 \leq k \leq i$ can be visualized as shown in Figure 4. Let A and B be the two sub-OBDDs whose roots are the children of an x_i-node. Consider a path from x_j to x_i. If this path contains the 0-edge of x_j, the subgraph rooted in x_i remains unchanged. If instead the path contains the 1-edge of x_j, the 0- and the 1-successor of the x_i-node are exchanged. This modification can prevent subgraph-isomorphisms in the new sub-OBDDs which are rooted in an x_k-node, $j+1 \leq k \leq i$.

Figure 4: Mutation $x_i \mapsto x_i \oplus x_j$ for $x_j < x_i$

(a) path along 0-edge of x_j (b) path along 1-edge of x_j



5.4 General two-bit re-encodings

The effect of each two-bit re-encoding can be split into the two-parts “Substitute the two variables x_i and x_j by some functions” and “replace the two functions δ_i and δ_j by some functions”. The variable substitution has an impact on all functions which depend essentially on x_i or x_j, whereas the function replacement only affects the functions δ_i and δ_j.

The next table shows that all re-encodings which are induced by the $4! = 24$ bijective functions $f : \mathbb{B}^2 \to \mathbb{B}^2$ can be obtained by a combination of maximal one XOR-transformation, an exchange variable transformation and the identity. Hence, beside the exchange variable transformation merely XOR-transformations are needed to produce all two-bit re-encodings. We write a two-bit re-encoding which is induced by f as

$$\rho^f_{i,j} : (x_1, \ldots, x_n) \mapsto (x_1, \ldots, x_{i-1}, f_1(x_i, x_j), x_{i+1}, \ldots, x_{j-1}, f_2(x_i, x_j), x_{j+1}, \ldots, x_n)$$

	f(00)	f(01)	f(10)	f(11)	function
1	00	01	10	11	$(x_1, x_2) = \mathrm{id}$
2	00	01	11	10	$(x_1, x_1 \oplus x_2)$
3	00	10	01	11	(x_2, x_1)
4	00	10	11	01	$(x_1 \oplus x_2, x_1)$
5	00	11	01	10	$(x_2, x_1 \oplus x_2)$
6	00	11	10	01	$(x_1 \oplus x_2, x_2)$
7	01	00	10	11	$(x_1, \overline{x_1 \oplus x_2})$
8	01	00	11	10	$(x_1, \overline{x_2})$
9	01	10	00	11	$(x_2, \overline{x_1 \oplus x_2})$
10	01	10	11	00	$(x_1 \oplus x_2, \overline{x_2})$
11	01	11	00	10	$(x_2, \overline{x_1})$
12	01	11	10	00	$(x_1 \oplus x_2, \overline{x_1})$

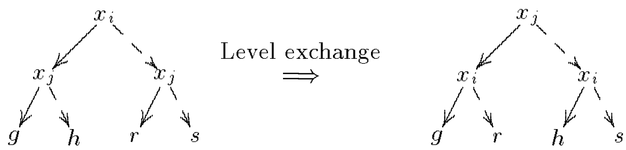
	f(00)	f(01)	f(10)	f(11)	function
13	10	00	01	11	$(\overline{x_1 \oplus x_2}, x_1)$
14	10	00	11	01	$(\overline{x_2}, x_1)$
15	10	01	00	11	$(\overline{x_1 \oplus x_2}, x_2)$
16	10	01	11	00	$(\overline{x_2}, x_1 \oplus x_2)$
17	10	11	00	01	$(\overline{x_1}, x_2)$
18	10	11	01	00	$(\overline{x_1}, x_1 \oplus x_2)$
19	11	00	01	10	$(\overline{x_1 \oplus x_2}, \overline{x_2})$
20	11	00	10	01	$(\overline{x_2}, \overline{x_1 \oplus x_2})$
21	11	01	00	10	$(\overline{x_1 \oplus x_2}, x_1)$
22	11	01	10	00	$(\overline{x_2}, \overline{x_1})$
23	11	10	00	01	$(\overline{x_1}, \overline{x_1 \oplus x_2})$
24	11	10	01	00	$(\overline{x_1}, \overline{x_2})$

The substitution $x_i \mapsto \overline{x_i}$ does not affect the size of the OBDD. As $\overline{x_i \oplus x_j} = \overline{x_i} \oplus x_j = x_i \oplus \overline{x_j}$, each of the above 24 transformations has the same effect w.r.t. the OBDD-size as a combination of the exchange variables transformation, the XOR-transformation and the identity operation. Moreover, for each of the 24 transformations, a combination of at most *two* of the "basis" transformations suffices.

5.5 Implementation aspects

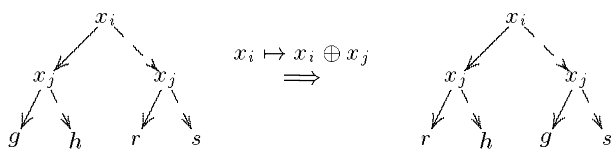
In this section we will describe how to implement the XOR-substitution $x_i \mapsto x_i \oplus x_j$ efficiently. Our starting point is the consideration of local changes in the variable order. In order to modify the variable order of OBDDs we iterate exchanging variables in adjacent levels. Since an exchange of adjacent variables is a local operation consisting only of the relinking of nodes in these two levels, this can be done efficiently as shown in Figure 5. In order to move a variable x_i behind an arbitrary variable x_j in the order, the exchanges of adjacent variables are iterated.

Figure 5: Exchanging two neighboring variables



In case of the XOR-operation and adjacent variables x_i and x_j, we can proceed analogous to the level exchange. Figure 6 shows the case where x_j is the direct successor of x_i in the order. The case where x_i is the direct successor of x_j in the order works analogously. If x_i and x_j are not adjacent, it would of course be helpful if we could simulate the substitution $x_i \mapsto x_i \oplus x_j$ by a sequence like $x_i \mapsto x_i \oplus x_{i+1}$, $x_i \mapsto x_i \oplus x_{i+2}$, …, $x_i \mapsto x_i \oplus x_j$. However, this straightforward idea does not work, as this would require operations in the intermediate steps which influence more than two adjacent levels.

Figure 6: Performing $x_i \mapsto x_i \oplus x_j$ for two neighboring variables x_i, x_j



A method that works and is only slightly more expensive than the exchange of two non-adjacent variables is the following: First, shift the variable x_i to a new position in the order which is adjacent to x_j. Then perform the XOR-operation, and then shift the variable x_i back to its old position. This technique retains the locality of the operation, as only nodes with a label x_k are influenced whose position in the order is between x_i and x_j.

6 Do All FSM-Descriptions Profit From XOR-Re-encodings?

In principle, the applicability of the XOR-transformation is not restricted to the use of OBDDs as underlying data structure. It can also be applied to other data structures for Boolean functions. However, the strong relationship between the XOR-transformation and local changes in the variable order like in the case of OBDDs does not always transfer to other representations. We will demonstrate this effect on OFDDs.

Ordered functional decision diagrams (OFDDs) [KSR92] are a modification of OBDDs which seem to be more compact for arithmetic functions. Each node v with label x_i in an OBDD represents a Shannon decomposition

$$f = x_i \cdot g + \overline{x_i} \cdot h,$$

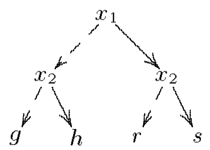
whereas each node v with label x_i in an OFDD represents a Reed-Muller decomposition

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

In both decompositions the functions g and h are independent of x_i and are the functions which are represented by the subgraphs rooted in the two successor nodes of v.

It has been shown in [BLW95] that local changes in the variable order have the same effect on OFDDs like on OBDDs. In particular, the exchange of two variables x_i and x_j in the order only affects the nodes of an OBDD resp. OFDD which are labeled by a variable whose position in the order is between x_i and x_j. These observations justify the notions of *local* changes. From the proof of Theorem 9 it follows that the XOR-transformation for OBDDs has also this pleasant local property. However, in spite of the fact that the Reed-Muller decomposition seems to operate well with XOR-transformations, the substitution $x_i \mapsto x_i \oplus x_j$ for OFDDs does not have the local property like in the case of OBDDs.

Figure 7: OFDD for $f = g \oplus x_2 \cdot h \oplus x_1 \cdot r \oplus x_1 \cdot x_2 \cdot s$



To prove this statement, consider the OFDD in Figure 7 which represents the function

$$f = g \oplus x_2 \cdot h \oplus x_1 \cdot r \oplus x_1 \cdot x_2 \cdot s$$

for some functions g, h, r, s independent of x_1, x_2. The function $f'(x_1, \ldots, x_n) = f(x_1 \oplus x_2, x_2, \ldots, x_n)$ is

$$\begin{aligned} f' &= g \oplus x_2 \cdot h \oplus (x_1 \oplus x_2) \cdot r \oplus (x_1 \oplus x_2) \cdot x_2 \cdot s \\ &= g \oplus x_2 \cdot (h \oplus r \oplus s) \oplus x_1 \cdot r \oplus x_1 \cdot x_2 \cdot s. \end{aligned}$$

Hence, there must exist a node in the OFDD for f' representing the function $h \oplus r \oplus s$. As h, r and s are arbitrary functions, the substitution operation $x_i \mapsto x_i \oplus x_j$ does not have the local property. However, as the $\oplus$-operation is a polynomial operation on OFDDs, the result of the substitution remains polynomial.

A tight relationship between our re-encoding techniques and specific OBDD-variants is the following: In a more general setting, the concept of domain transformations has been proposed for the manipulation of switching functions [BMS95, FKB95]. The corresponding variants of OBDDs are called TBDDs. In the following we show that our encoding transformations can also be interpreted as TBDD-transformations.

Definition 11. Let $f : \mathbb{B}^n \to \mathbb{B}$ be a switching function, and let $\tau : \mathbb{B}^n \to \mathbb{B}^n$ be a bijective mapping. A *τ-TBDD-representation* of f is an OBDD-representation of $f \circ \tau$, where $\circ$ denotes the composition of functions.

It turns out that every re-encoding function ρ defines a transformation within the TBDD-concept. However, the OBDDs for the next-state functions of a re-encoded machine with re-encoding function ρ are not isomorphic to the ρ-TBDDs, as in the OBDDs of the re-encoded machine, the transformation ρ^{-1} is also involved (see Equation (1) in section 3).

7 Experimental Results

In this section we present some very first experimental results on the extended optimization techniques for OBDDs. We built up some routines on top of the OBDD-package of D. Long and used the ISCAS89 benchmark circuits s1423, s5378, s9234 which have a large number of state bits and have also formed the set of examples in [RS95]. Each optimization run consists of three phases: First, we applied Rudell's sifting algorithm [Rud93] for finding a good variable order. Then some minimization based on XOR-transformations is performed. Finally, sifting is applied once more to re-establish a suitable variable order. The table shows the obtained shared OBDD-sizes of the next-state functions in comparison to the sizes that were obtained *without* the minimization by XOR-transformations.

Circuit	# state bits	without XOR	with XOR
s1423	74	3054	2853
s5378	121	1801	1517
s9234	228	4621	4359

The minimization based on XOR-transformation works as follows: In a preprocessing step we compute promising pairs (i, j) for an XOR-transformation. The heuristic criteria for considering a pair (i, j) as promising are:

1. the next-state functions δ_i and δ_j have a nearly equal support, or
2. the variables x_i and x_j appear in nearly the same functions.

Then, as long as improvements are possible, the best XOR-transformation among these pairs are applied. In order to avoid the expensive computations $\delta_i \oplus \delta_j$, we only perform this step if the variable substitution $x_i \mapsto x_i \oplus x_j$ yields a good intermediate result.

It must be admitted that in our experiments, the running times are significantly higher than the running times for pure sifting. This is due to a non-optimal implementation of the XOR-transformation and due to the large number of performed XOR-transformations. However, we think that intensive studies of different strategies (analogous to the intensive and finally quite successful studies of the variable order) should be able to improve these results by far. Hence, we think that these experimental results underline the optimization potential of the XOR-transformation.

An efficient implementation of the XOR-transformation and the construction of effective minimization strategies is in progress.

8 Conclusion and Future Work

We have proposed and analyzed new re-encoding techniques for minimizing OBDDs. In particular, we have proposed the XOR-transformation and shown that this transformation is in fact the only new transformation on two state variables. This transformation can in certain cases significantly enrich the set of basic operations for the optimization of OBDDs. In the future we propose to study heuristics for choosing the right transformation pairs and efficient combinations of variable reordering techniques with the new proposed transformations. Furthermore, the dynamic application of the new re-encoding technique in the traversal of a finite state machine seems promising: It helps to reduce the OBDD-sizes for the set of reached states and also to reduce the efforts of the image computations.

9 Acknowledgements

We wish to thank Stefan Krischer, Jan Roßmann, Anna Slobodová and Fabio Somenzi for interesting discussions and many valuable comments.

References

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

[BLW95] B. Bollig, M. Löbbing, and I. Wegener. Simulated annealing to improve variable orderings for OBDDs. In *International Workshop on Logic Synthesis*, 1995.

[BMS95] J. Bern, Ch. Meinel, and A. Slobodová. OBDD-based Boolean manipulation in CAD beyond current limits. In *32nd ACM/IEEE Design Automation Conference*, 1995.

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

[FKB95] M. Fujita, Y. Kukimoto, and R. K. Brayton. BDD minimization by truth table permutations. In *International Workshop on Logic Synthesis*, 1995.

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

[KSR92] U. Kebschull, E. Schubert, and W. Rosenstiel. Multilevel logic synthesis based on functional decision diagrams. In *European Design Automation Conference*, pp. 43–47, 1992.

[LL92] H.-T. Liaw and C.-S. Lin. On the OBDD-representation of general Boolean functions. *IEEE Transactions on Computers*, 41:661–664, 1992.

[MWBS88] S. Malik, A. Wang, R. K. Brayton, and A. Sangiovanni-Vincentelli. Logic verification using binary decision diagrams in a logic synthesis environment. In *IEEE International Conference on Computer-Aided Design*, pp. 6–9, 1988.

[PS95] S. Panda and F. Somenzi. Who are the variables in your neighborhood. In *IEEE International Conference on Computer-Aided Design*, 1995.

[QCC+95] S. Quer, G. Cabodi, P. Camurati, L. Lavagno, E. M. Sentovich, and R. K. Brayton. Incremental FSM re-encoding for simplifying verification by symbolic traversal. In *International Workshop on Logic Synthesis*, 1995.

[RS95] K. Ravi and F. Somenzi. High-density reachability analysis. In *IEEE International Conference on Computer-Aided Design*, 1995.

[Rud93] R. Rudell. Dynamic variable ordering for ordered binary decision diagrams. In *IEEE International Conference on Computer-Aided Design*, pp. 42–47, 1993.

[Sha49] C. E. Shannon. The synthesis of two-terminal switching circuits. *Bell System Technical Journal*, 28:59–98, 1949.

[SW93] D. Sieling and I. Wegener. NC-algorithms for operations on binary decision diagrams. *Parallel Processing Letters*, 3:3–12, 1993.

[TM96] T. Theobald and Ch. Meinel. State encodings and OBDD-sizes. Technical Report 96-04, Universität Trier, 1996.

Appendix

Proof of Theorem 1

Proof. Let $K = \lfloor 2^n/n \rfloor$. Suppose there are c_i variables with label x_i in the OBDD, and let $c_{n+1} = K - \sum_{i=1}^n c_i$. c_{n+1} serves to cover the case $\sum_{i=1}^n c_i < K$. Since each c_i, $1 \leq i \leq n+1$, is a nonnegative integer, the equation $\sum_{i=1}^n c_i = K$ has exactly $\binom{n+K}{K}$ solutions. Suppose that all the vertices are ordered by increasing positions in the variable order, then each of the two edges of a vertex can only connect to its successors, including the nonterminal vertices 0 and 1. Hence there are at most $(K+1)^2 \, K^2 \, (K-1)^2 \cdot \ldots \cdot 2^2 = ((K+1)!)^2$ ways to place the edges. Since there are $n!$ different variable orders, there are at most

$$n! \binom{n+K}{K} ((K+1)!)^2$$

shared OBDDs with at most K vertices for their optimal variable order. Due to the cyclic symmetry of the autonomous counter it follows

$$\begin{aligned} e(n) &\leq 2^n \, n! \, (n+K)! \, ((K+1)!)^2 \;\leq\; 2^n \, (2n+3K+2)! \\ &\leq 2^n (2n + 3 \cdot 2^n/n + 2)! \;\leq\; 2^n (4 \cdot 2^n/n)! \quad \text{for sufficiently large } n, \end{aligned}$$

and therefore

$$\frac{e(n)}{a(n)} \to 0 \quad \text{as} \quad n \to \infty.$$

□

Proof of Theorem 9

Proof. Let $f(x_1, \ldots, x_n)$ be a switching function,

$$f'(x_1, \ldots, x_{i-1}, x_i, x_{i+1}, \ldots, x_n) = f(x_1, \ldots, x_{i-1}, x_i \oplus x_j, x_{i+1}, \ldots, x_n),$$

and P, P' the OBDDs of f resp. f'.

CASE 1: $j < i$. As there is a 1-1-correspondence between (x_i, x_j) and $(x_i \oplus x_j, x_j)$, Theorem 8 implies directly that $s_k^* = s_k$ for $k \leq j-1$ or $k \geq i+1$. In particular, a subfunction $f_{x_1=a_1,\ldots,x_{k-1}=a_{k-1}}$ depends essentially on x_k if and only if $f'_{x_1=a_1,\ldots,x_{k-1}=a_{k-1}}$ depends essentially on x_k for $k \leq j-1$.

Next we show $s_j^* \leq s_1 + \ldots + s_{j-1} + 1$. It can happen that a subfunction $f_{x_1=a_1,\ldots,x_{j-1}=a_{j-1}}$ does not depend essentially on x_j but the subfunction $f'_{x_1=a_1,\ldots,x_{j-1}=a_{j-1}}$ does. Due to this effect s_j^* can become large and there is no upper bound for s_j^* depending only on s_j. Each

x_j-node in P' has to be reached from at least one node labeled by x_k, $k \leq j-1$, or the source is the only node labeled by x_j. There are at most

$$\begin{aligned} 2s_1^* + \ldots + 2s_{j-1}^* - s_1^* - s_2^* - \ldots - s_{j-1}^* + 1 &\leq s_1^* + s_2^* + \ldots + s_{j-1}^* + 1 \\ &= s_1 + s_2 + \ldots + s_{j-1} + 1 \end{aligned}$$

edges starting in a node labeled by x_k, $k \leq j-1$, and leading to a node labeled by x_j. Hence, $s_j^* \leq s_1 + \ldots + s_{j-1} + 1$.

Now we prove $s_k^* \leq 2s_k$, $j+1 \leq k \leq i$. Again, we use that $f_{x_1=a_1,\ldots,x_{k-1}=a_{k-1}}$ depends essentially on x_k if and only if $f'_{x_1=a_1,\ldots,x_{k-1}=a_{k-1}}$ does. Further, each subfunction $f'_{x_1=a_1,\ldots,x_{k-1}=a_{k-1}}(x_k,\ldots,x_n)$ is identical to

$$\begin{aligned} & f_{x_1=a_1,\ldots,x_{k-1}=a_{k-1}}(x_k,\ldots,x_i,\ldots,x_n) \quad \text{(in case } a_j = 0\text{)} \\ \text{or to}\quad & f_{x_1=a_1,\ldots,x_{k-1}=a_{k-1}}(x_k,\ldots,\overline{x_i},\ldots,x_n) \quad \text{(in case } a_j = 1\text{)}. \end{aligned}$$

These two subfunctions do not have to be necessarily different. If follows $s_k^* \leq 2s_k$.

CASE 2: $i < j$. For $k \leq i$ or $k \geq j+1$ we have $s_k^* = s_k$ due to Theorem 8. In particular, this relation holds for the index i, as $f_{x_1=a_1,\ldots,x_{i-1}=a_{i-1}}$ depends essentially on x_i if and only if $f'_{x_1=a_1,\ldots,x_{i-1}=a_{i-1}}$ does.

For $i+1 \leq k \leq j-1$ the number s_k^* is equal to the number of functions f'^S, $S \subset \{1,\ldots,k-1\}$, depending essentially on x_k. By the Shannon expansion it holds

$$f'^S = \overline{x_j} f'^S_{x_j=0} + x_j f'^S_{x_j=1}.$$

The function f'^S depends essentially on x_k if and only if at least one of the two subfunctions $f'^S_{x_j=0}$ and $f'^S_{x_j=1}$ depends essentially on x_k. By computing upper bounds on the number of different subfunctions $f'^S_{x_j=0}$ and $f'^S_{x_j=1}$, we can derive an upper bound for s_k^*. The number of different subfunctions $f'^S_{x_j=a}$, $a \in \{0,1\}$, is equal to the number of different subfunctions $f^S_{x_j=a'}$, $a' \in \{0,1\}$, (as both x_i and x_j are specified and due to the 1-1-correspondence between (x_i, x_j) and $(x_i \oplus x_j, x_j)$), and this number is at most the number of different subfunctions f^S. All the subfunctions f^S are represented in the OBDD P. Combining the three possible cases which of the two subfunctions $f'^S_{x_j=0}$ resp. $f'^S_{x_j=1}$ depend on x_k we obtain

$$s_k^* \leq s_k^2 + 2s_k(s_{k+1} + \ldots + s_n + 2).$$

It remains to find an upper bound for s_j^*. Let $S \subset \{1,\ldots,j-1\}$. Then f'^S depends essentially on x_j if and only if $f'^S_{x_j=0}$ and $f'^S_{x_j=1}$ are different. The number of subfunctions $f'^S_{x_j=a}$, $a \in \{0,1\}$, is identical to the number of subfunctions $f^S_{x_j=a'}$, $a' \in \{0,1\}$, as x_i and x_j are specified. The subfunctions $f^S_{x_j=a}$ are represented in the OBDD P. There are at most $s_{j+1} + \ldots + s_n + 2$ choices for $f^S_{x_j=0}$ and then at most $s_{j+1} + \ldots + s_n + 1$ choices for $f^S_{x_j=1}$. The size of an OBDD after the XOR-transformation $x_i \mapsto x_i \oplus x_j$ with $i < j$ is therefore

$$\begin{aligned} \text{size}(P') &\leq \sum_{k=1}^{i} s_k + \sum_{k=i+1}^{j-1} \Big(s_k^2 + 2s_k(s_{k+1} + \ldots + s_n + 2) \Big) \\ &\quad + (s_{j+1} + \ldots + s_n + 2)(s_{j+1} + \ldots + s_n + 1) + \sum_{i=j+1}^{n} s_k + 2 \\ &\leq s_1 + \ldots + s_i + (s_{i+1} + \ldots + s_{j-1} + s_{j+1} + \ldots + s_n + 2)^2 \end{aligned}$$

□