A Reducibility Concept for Problems Defined in Terms of Ordered Binary Decision Diagrams*

Christoph Meinel, Anna Slobodová †‡
FB IV – Informatik, Universität Trier, D–54286 Trier, Germany
meinel@uni-trier.de

Abstract

Reducibility concepts are fundamental in complexity theory. Usually, they are defined as follows: A problem Π is reducible to a problem Σ if Π can be computed using a program or device for Σ as a subroutine. However, this approach has its limitations if restricted computational models are considered. In the case of ordered binary decision diagrams (OBDDs), it allows merely to use the almost unmodified original program for the subroutine.

Here we propose a new reducibility concept for OBDDs: We say that Π is reducible to Σ if an OBDD for Π can be constructed by applying a sequence of elementary operations to an OBDD for Σ. In contrast to previous reducibility notions the newly introduced reduction is able to reflect the real needs of a reducibility concept in the context of OBDD-based complexity classes: it allows to reduce those problems to each other which are computable with the same amount of OBDD-resources and it gives a tool to carrying over lower and upper bounds.

1 Introduction

Reducibility is one of the most basic notions in complexity theory. It provides a fundamental tool for comparing the computational complexity of different problems. The key idea is to use a program for a device that solves one problem Σ as a subroutine within the computation of another problem Π. If this is possible, Π is said to be reducible to Σ. Reductions provide the possiblity to conclude upper bound results on the computational complexity of problem Π and lower bounds for Σ, if one insists that the program for Π designed around the subroutine for Σ respects certain resource complexity bounds of interest.

In the past, a great variety of different reducibility notions has been investigated in order to get a better understanding of the different computational paradigms and/or resource bounds. Here we only mention polynomial-time Turing reducibility, log-space reducibility, polynomial projection reducibility, and NC^1-reducibility (see, e.g., [Lee90], [BDG88]). This great variety of different reducibility notions is a consequence of the fact that the computational power available to the reduction must not be stronger than the computational power of the complexity class under consideration. Otherwise, the possibility of hiding some essential computations within the reduction is a threat to the relevance of the obtained results. For example, polynomial-time reducibility does not give any insight into the computational complexity of logarithmic-time bounded computations. The computational power implementable in a reducibility notion for complexity classes defined in terms of very restricted computational models (e.g., eraser Turing machines [KMW88], real-time branching programs [KW87], or ordered binary decision diagrams [Bry86]) becomes extremely limited, since in general, almost all of the resources are consumed

*in Proceedings STACS'97, LNCS
†New Address: ITWM-Trier, Bahnhofstr. 30-32, D–54292 Trier, Germany
‡We are grateful to DAAD ACCIONES INTEGRADAS, grant Nr. 322-ai-e-dr

already by the programs which are used as subroutines in the reductions. Hence, the traditional approach results in reducibility concepts which enable to relate merely highly similar problems (e.g., [BW96]). Since complexity classes defined by such restricted models are interesting for the theory – they occur in connection with our limited abilities in proving lower bounds [e.g., KMW88, KW87, Mei89] – and of practical importance – ordered binary decision diagrams are the state of the art data structure for computer aided circuit design [Bry86] – it is highly desirable to develop more powerful reducibility concepts.

Here, we consider the case of complexity classes defined by ordered binary decision diagrams (OBDDs), (i.e. read-once binary decision diagrams with a fixed variable ordering). We attempt to overcome the difficulties mentioned above by introducing a new reducibility concept that is based on the following idea: A problem Π is reducible to a problem Σ if an OBDD for Π can be constructed from a given OBDD for Σ by applying a sequence of elementary operations (here 'elementary' means 'performable in constant time'). In contrast to previous reducibility notions the newly introduced reduction notion is able to reflect the real needs of a reducibility concept in the context of OBDD-based complexity classes: Firstly, it allows to reduce those problems to each other which are computable with the same amount of OBDD-resources, and, secondly, it allows to carry over lower and upper bounds.

Although well-motivated, a reducibility based on sequences of elementary operations is difficult to describe and to handle since it has to deal with permanently changing OBDDs. For this reason, we develop a formalism which allows a more 'static' description in terms of the so-called OBDD-transformer. We prove that the size of an OBDD which is obtained by the application of a sequence of elementary operations can be estimated in terms of the sizes of the original OBDD and of the corresponding OBDD-transformer. Hence, the formalism gives a solid basis for complexity theoretic investigations.

2 Notations and Preliminaries

Let X_n denote the set $\{x_1, x_2, ..., x_n\}$ of Boolean variables. A *variable ordering* on X_n is a total order on X_n and is described by a permutation of the index set $I_n = \{1, ..., n\}$, i.e. $x_i < x_j$ iff $\pi^{-1}(i) < \pi^{-1}(j)$. Throughout the paper, we will work with the extension of an ordering to constants *false* and *true* which are defined to be maximal (*false* and *true* become incomparable). Identity defines the so-called natural ordering. Two orderings (possibly defined on different variable sets) are said to be *consistent* if there is no pair (x_i, x_j) such that x_i precedes x_j in one ordering and x_j precedes x_i in the order.

By functions, we mean Boolean functions $\{\mathit{false},\ \mathit{true}\}^n \longrightarrow \{\mathit{false},\ \mathit{true}\}$. The standard representation of *false* and *true* is 0 and 1, respectively. However, it will be more convenient for us to represent *false* by -1 and *true* by 1. $\cong$ is used for the isomorphism of labeled graphs.

Definition.
An ordered binary decision diagram (OBDD) *over X_n is a connected acyclic directed graph with the following properties:*

(i) *There is one distinguished node without incoming edges, called* root.
(ii) *Nodes without outgoing edges (called* sinks*) are labeled by -1 or 1.*
(iii) *Each non-sink node is labeled by a variable from X_n. The labeling fullfills the read-once property, i.e., on each root-to-sink path, any variable appears at most once.*
(iv) *Each non-sink node has two outgoing edges that are called* true- *and* false-edge *and are labeled by* 1 *and* -1*, respectively.*
(v) *Each node has a* negation mark -1 *or* 1.
(vi) *All variable orderings defined by the occurence of variables on root-to-sink paths are consistent.*

The nodes that are labeled by the same variables form a *level.* Let π be a variable ordering. An ordered binary decision diagram is a πOBDD if all variable orderings defined by the occurence of variables on root-to-sink paths are consistent with π. The size of an OBDD P is defined as the number of its nodes and is denoted by $size(P)$.

In order to make figures more compact, we omit the labels of the edges and use solid lines for the true-edges and dotted lines for the false-edges of an OBDD. The negation mark 1 is usually omitted.

Let v be a node of an OBDD. Each input assignment $\alpha = (a_1, \ldots, a_n)$ uniquely determines a v-to-sink path $p_v(\alpha)$ according to the following rule: At an inner node with label x_i, the outgoing edge with label a_i is chosen. Let $\mathrm{sgn}_v(\alpha)$ denote the product of the negation marks on the nodes on $p_v(\alpha)$ and $sink_v(\alpha)$ be the label of the sink on $p_v(\alpha)$. The Boolean function which is represented by v is defined by $f_v(\alpha) = \mathrm{sgn}_v(\alpha) \cdot sink_v(\alpha)$, for any input α. The function represented by an OBDD is the function represented by its root. We will not distinguish between an OBDD and its root as long as it introduces no ambiguity. In this sense, the successors of a node are the nodes reachable via edges starting in the node as well as the subOBDDs rooted in these nodes. An OBDD (respectively, a πOBDD) for a function f is denoted by OBDD(f) (respectively, πOBDD(f)).

The defined OBDD model slightly differs from the one usually used. The introduction of negated nodes is motivated by existing OBDD-implementations [e.g., BRB90, Lon93, Som96], where the use of negated edges allows to save up to half the size of the representation.

If P is an OBDD, then $\overline{P}$ denotes the OBDD obtained from P by multiplying the negation mark of the root by -1. Two OBDDs are *(functionally) equivalent* (denoted by $\equiv$) if they represent the same function. An OBDD is called *reduced* if all nodes have negation mark 1 and no two subgraphs represent equivalent OBDDs. We remark that OBDDs can be reduced in linear time [SW93]. For a fixed variable ordering π, the representation of a Boolean function in terms of reduced πOBDD is uniquely determined [Bry86].

The subject of this paper is the development of a reducibility concept usable for complexity investigation of OBDDs. Unfortunately, in the context of OBDDs, the notions 'reduce' and 'reduction' have a fixed meaning in the sense mentioned above. Speaking about reductions in the complexity theoretical sense, we avoid the terminological ambiguity by using the term 'OBDD-transformation' instead of 'OBDD-reduction'.

The non-uniformity of OBDDs and the sensitivity of the structure and the size of an OBDD to a variable ordering have to be taken into account in the definition of a 'problem' in this context. Due to the former attribute, we work with a *family of functions*. The latter makes differences among representations of a function with respect to different orderings. Indeed, the differences in sizes of the respective OBDDs may be exponential.

Definition.
A problem (f, π) *is a sequence of pairs* $((f_n, \pi_n))_{n=1}^{\infty}$*, where* $f_n : \{\mathit{false}, \mathit{true}\}^n \longrightarrow \{\mathit{false}, \mathit{true}\}$ *is a Boolean function and* π_n *is a permutation which defines a variable ordering for* f_n*. For each* n*,* (f_n, π_n) *is called an* instance *of the problem.*

For each problem (f, π), there is a uniquely determined sequence $(\pi_n \mathrm{OBDD}(f_n))_{n=1}^{\infty}$ of reduced OBDDs.

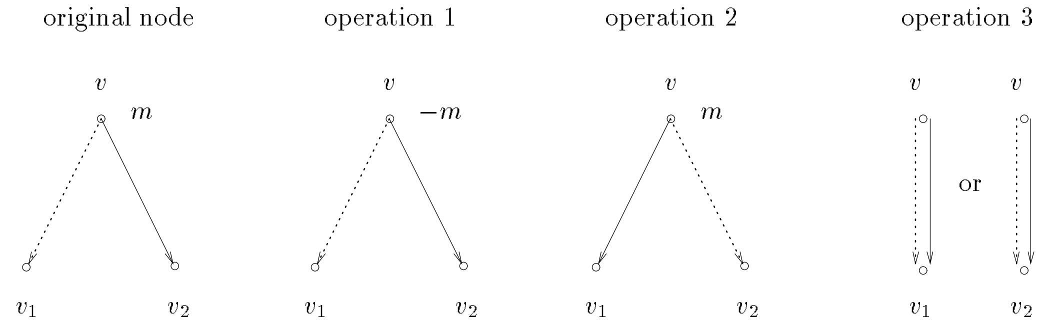
3 OBDD-Transformations

Our aim is to introduce a reducibility concept for complexity classes defined by OBDDs which reflects the expense that arises if one constructs an OBDD P' for one problem from a given OBDD P for another problem. This expense is measured in the number of 'elementary' operations that are necessary for constructing P' from P. Here, an operation is considered 'elementary' if it can be performed in constant time (under the unit cost measure).

Operation 1. Setting/deletion of the negation mark of a node, i.e. negating the subfunction computed in this node.

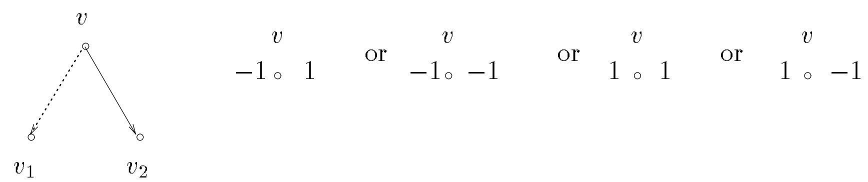
Operation 2. Exchange of outgoing edges of a node, i.e. negation of the label.

Operation 3. Redirection of one outgoing edge towards the second one, i.e. replacing the label of this node by the corresponding constant value.

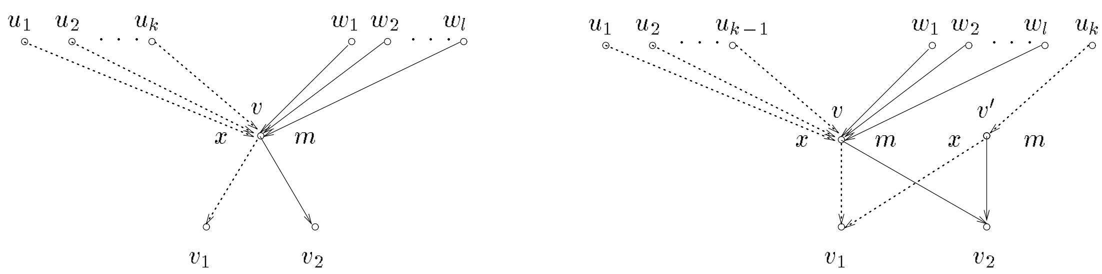


Application of the operations 1, 2 and 3.

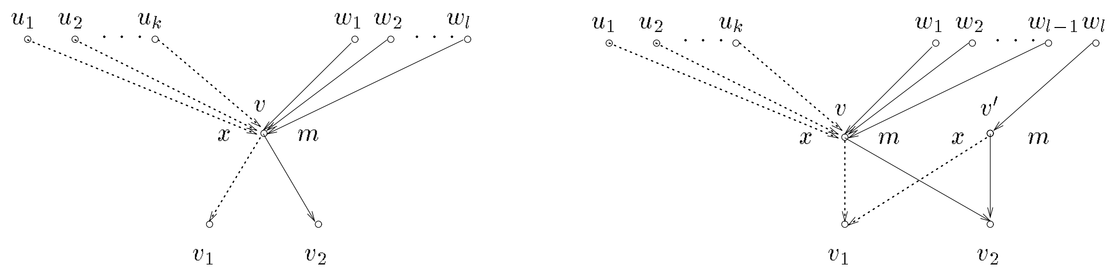
Operation 4. Replacing a node by a sink, i.e. replacing the subfunction computed in this node by a trivial constant function.



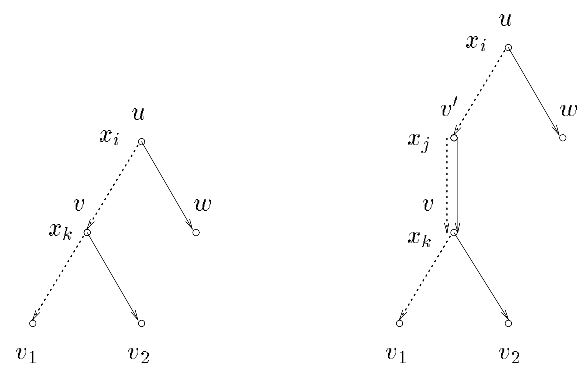
Operation 5. Node splitting, i.e. redirection of an edge to a newly created equivalent node.



or



Operation 6. Introduction of a dummy node labeled with a variable that is consistent with the given variable ordering (let $x_i < x_j < x_k$ in the picture).



Considering the usual representation of an OBDD, where each node stores its label, its mark and two pointers to its sons all operations are performable in constant time.

The operations defined above have clear semantics. The first operation negates the subfunction defined in a node, the second operation negates the evaluation of the checked variable, the third one corresponds to the restriction, and the fourth operation provides a replacement of a subfunction by a trivial function. Unlike the first four operations which may change the functionality of an OBDD, the last two allow to turn to unreduced OBDDs and to perform the subsequent operations merely on some distinguished subgraphs/subfunctions.

It is quite natural to have the possibility to rename variables. Among other things, it allows to move from one variable ordering to another. In order to preserve the 'read-once property', we have to insist that renaming does not identify any two different variables.

Definition.
A problem (f, π) is an (OBDD-)transformation *of a problem (g, σ) (written as $(f, \pi) \leq_{OBDD} (g, \sigma)$), if for each n, there is an m such that the reduced π_n OBDD(f_n) can be obtained from the reduced σ_m OBDD(g_m) by a sequence of elementary operations completed by reduction and a renaming of variables (i.e., application of a bijective mapping on the set of variables that occur in the reduced OBDD).*

Example A. The following example illustrates an OBDD-transformation defined by means of the application of a sequence of elementary operations. We show that, for any variable ordering π,

$$\left(\bigvee_{i=1}^{n} x_i, \pi\right) \quad \leq_{OBDD} \quad \left(\bigwedge_{i=1}^{n} x_i, \pi\right) .$$

We start with an OBDD that computes $\bigwedge_{i=1}^{n} x_i$. First, applying Operation 2 on each node, we obtain an OBDD that computes $\bigwedge_{i=1}^{n} \overline{x_i}$. Then we apply Operation 1 to the root of this OBDD. We obtain an OBDD that computes $\overline{\bigwedge_{i=1}^{n} \overline{x_i}}$. Since, by DeMorgan's rules, $\overline{\bigwedge_{i=1}^{n} \overline{x_i}} = \bigvee_{i=1}^{n} x_i$ the constructed OBDD is the desired one. For case $n = 2$ and natural variable ordering, the transformation is shown in Fig. 1.

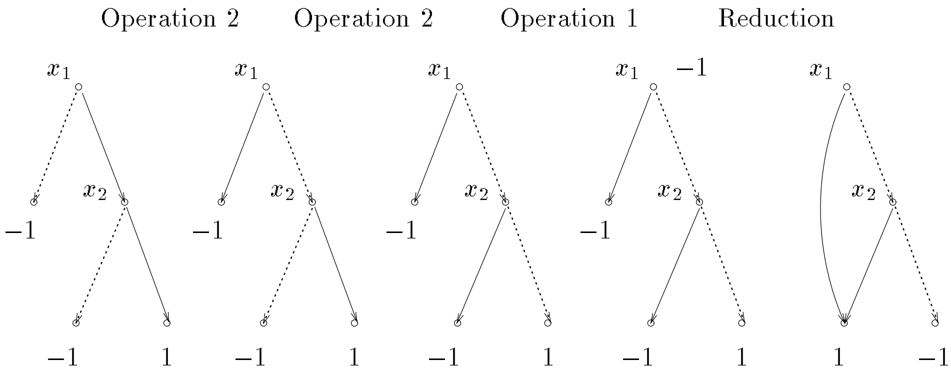


Fig. 1

It is interesting to remark, that the functions considered in the example are not reducible by means of read-once projections [BW96].

The desciption of more complex and interesting (OBDD) transformation in terms of sequences of elementary operations which have to be applied on permanently changing OBDDs may be quite cumbersome. In order to overcome these difficulties, we develop a formalism which allows a more 'static' and easier manipulatable desciption of transformations. The basic idea of this formalism is to encode a sequence of elementary operations by a certain OBDD-like graph structure, the so-called transformer. Then the application of a sequence of elementary operations to a given OBDD is realized by means of an algorithm *Derive* which computes the desired OBDD from the given one and a corresponding transformer.

Using this formalism it becomes much easier to get the transformations like *(MOD-2,π)* $\leq_{OBDD}$ *(MOD-2^i,π)*, *(MOD-2^i,π)* $\leq_{OBDD}$ *(MOD-2,π)*, or *(SQU,π)* $\leq_{OBDD}$ *(MUL,π')* which show the power of the introduced reduction concept in comparison with the formerly mentioned read-once-projections.

Moreover, since we prove that the size of a transformed OBDD can be estimated in terms of the sizes of the original OBDD and of the transformer, our formalism open a way for complexity theoretic investigations.

4 Alternative Description of OBDD-Transformations

In the following we develop an alternative more 'static' description of OBDD-transformations.

4.1 OBDD-Transformer

An OBBD-transformer is an OBDD-like structure with an extended labeling of the nodes.

Definition.
For any variable set X_n we define $\tilde{X_n} = X_n \cup \{\bar{x} | x \in X_n\} \cup \{1_x | x \in X_n\} \cup \{-1_x | x \in X_n\} \cup \{-1, 1, \perp\}$. A mapping from $\tilde{X_n}$ to $X_n \cup \{-1, 1\}$ is defined by $x \mapsto |x|$, where

(i) $|x_i| = |\bar{x}_i| = |1_{x_i}| = |-1_{x_i}| = x_i$

(ii) $|-1| = -1$

(iii) $|1| = 1$

(iv) $|\perp| = 1$

An OBDD-transformer *(or simply* transformer*) T over X_n is a graph whose nodes are labeled by elements from $\tilde{X}_n$ such that the graph obtained from T by replacing each node label x by $|x|$ is an OBDD over X_n.*

A πOBDD-transformer is an OBDD-transformer where the orderings of the variables associated with the labels of the nodes on the root-to-sink paths are consistent with π. Particularly, each πOBDD is a πOBDD-transformer. The *size* of a transformer T is defined as the number of its nodes and is denoted by $size(T)$.

4.2 The Algorithm *Derive*

The algorithm *Derive* realizes OBDD-transformation defined in terms of OBDD-transformer. Let P be a πOBDD and T a πOBDD-transformer. The result of $Derive(P, T)$ is denoted by $P \diamond T$. The algorithm starts in the roots of P and T, scans the graphs in parallel, and creates the result recursively. The information in the root of T describes the required changes in the root of P. Instead of modifying P, we create a new graph as the result. The mark -1 corresponds to Operation 1, the negative variable to Operation 2, labels 1_x and -1_x to Operation 3, and the sink node to Operation 4.

We explain this idea in detail. If x is a label (of the root) of T, then, for each constant $\delta \in \{-1, 1\}$, (the root of) $T_{|x|=\delta}$ denotes the δ-successor (of the root) of T. Similarly, $P_{x=\delta}$ is the δ-successor of P.

Trivial case:

- *(i)* If T is a sink labeled by $\perp$, then $P \diamond T$ is a graph isomorphic to P but its mark equals to product the marks of P and T.
- *(ii)* If T is a sink labeled by a constant, then $P \diamond T$ is isomorphic to T.

Nontrivial case:

Let x be the label of P and y the label of T. The program continues according to the ordering relation of x and $|y|$.

- *(i)* If $x < |y|$, then a node is created with the same label and mark as P has and with true-successor $P_{x=1} \diamond T$ and false-successor $P_{x=-1} \diamond T$.
- *(ii)* If $x > |y|$, then a node is created, labeled by $|y|$, with the same mark as T and with the true-successor $P \diamond T_{|y|=1}$ and false-successor $P \diamond T_{|y|=-1}$.
- *(iii)* If $x = |y|$, then a node labeled by x is created. The mark of the node will be the product of the marks of P and T. The true- and false-successor of the node depend on y:

		true-successor	false-successor
$y = x$	:	$P_{x=1} \diamond T_{x=1}$	$P_{x=-1} \diamond T_{x=-1}$
$y = \bar{x}$	:	$P_{x=-1} \diamond T_{x=1}$	$P_{x=1} \diamond T_{x=-1}$
$y = 1_x$	:	$P_{x=1} \diamond T_{x=1}$	$P_{x=1} \diamond T_{x=-1}$
$y = -1_x$	:	$P_{x=-1} \diamond T_{x=1}$	$P_{x=-1} \diamond T_{x=-1}$

Particularly, if T is an OBDD, then for each OBDD P it holds $P \diamond T = T$.

The reduction running in parallel is easy to implement in the algorithm without changing its asymptotic time and space performance. W.l.o.g. we assume that the result of *Derive* is a reduced OBDD.

Proposition 1.
Let P be a πOBDD and T a πOBDD-transformer. The time as well as the space complexity of the algorithm Derive *on an input (P, T) is bounded by $\mathcal{O}(size(P) \cdot size(T))$. The size of the output is bounded by $size(P) \cdot size(T)$.*

Sketch of the proof. The main observation in the complexity analysis of the algorithm is that each pair of nodes (u, v), where $u \in P$ and $v \in T$, generates at most two recursive calls. $\square$

Derive is extendable on any pair consisting of a π_1OBDD and a π_2OBDD-transformer as long as π_1 and π_2 are consistent. In that case, there is a common order that is defined over all variables that appear in the OBDD and the transformer, and is consistent with both orders. Since this ordering is not uniquely defined, but has substantial influence on the result it should be explicitly given as the third input parameter. If the variable orderings in T and P are not consistent, $P \diamond T$ remains undefined.

4.3 The Algorithm *Compose*

The algorithm *Compose* computes the composition of two transformers that reflects the 'concatenation' of two OBDD-transformations. The reduced result of *Compose*(T_1, T_2) is denoted by $T_1 \circ T_2$. Like in the case of *Derive*, an implementation of the reduction within *Compose* is straightforward and causes no additional costs. *Compose* is closely related to *Derive*. It is designed in such a way that $(P \diamond T_1) \diamond T_2$ is isomorphic to $P \diamond (T_1 \circ T_2)$ which corresponds to the idea of the composition of two transformations. he complexity analysis of *Compose* is based on the same arguments as the analysis of *Derive*.

Proposition 2.
Let T_1 and T_2 be πOBDD-transformers. The time as well as the space complexity of the algorithm Compose *on an input (T_1, T_2) is bounded by $\mathcal{O}(size(T_1) \cdot size(T_2))$. The size of the output is bounded by $size(T_1) \cdot size(T_2)$.* □

Proposition 3.
Let P be a πOBDD P and T_1, T_2 πOBDD-transformers. It holds: $P \diamond (T_1 \circ T_2) \cong (P \diamond T_1) \diamond T_2$.

Sketch of the proof: The design of *Compose* was exactly aimed to fulfill the property in the proposition. Indeed, the correctness of $P \diamond (T_1 \circ T_2) \cong (P \diamond T_1) \diamond T_2$ can be easily shown by induction on the priority of the top variable of the triple (P, T_1, T_2) (i.e., on the position of the smallest variable associated with the labels of the roots of $P, T_1,$ and T_2 with respect to π). Recursive calls to *Derive* and *Compose* are always generate for arguments with smaller top variable that allows to use the inductive hypothesis. The proof is then reduced to the consideration of all possible cases which are characterized by the different relative positions of the top variables of arguments. □

Like in the case of *Derive*, *Compose* is extendable to any pair of OBDD-transformers as long as their variable orderings are consistent. Otherwise, the result of $\circ$ is undefined.

4.4 Transformers vs. Sequences of Elementary Operations

Now we prove that any sequence of elementary operations performed on a given OBDD P can be simulated by applying the algorithm *Derive* on P and a suitable transformer T. Moreover, we show that also the reverse is true: any transformer describes a particular OBDD-transformation. Hence, transformer gives an equivalent 'static' description of the 'dynamical' transformation process.

Theorem 4.
$(f, \pi) = (f_n, \pi_n)_{n=1}^{\infty}$ is an OBDD-transformation of $(g, \sigma) = (g_n, \sigma_n)_{n=1}^{\infty}$, if and only if, for every n, there is an m and an OBDD-transformer T_n such that π_nOBDD(f_n) and σ_mOBDD$(g_m) \diamond T_n$ are equivalent up to a renaming of the variables.

We say that the sequence of OBDD-transformers $(T_n)_{n=1}^{\infty}$ of the theorem *realizes* an OBDD-transformation $(f, \pi) \leq_{\text{OBDD}} (g, \sigma)$. We remark that there is nothing said about the number of variables in T_n yet.

The **proof** of Theorem 4 is the consequence of the following two lemmas.

Lemma 5.
Let T be a πOBDD-transformer and let P be a πOBDD. Then $P \diamond T$ is a transformation of P, i.e., it can be obtained from P by applying a suitable sequence of elementary operations.

Idea of the proof.

In order to prove the statement, we show that, for each P and T, there is a P' derivable from P by a sequence of elementary operations such that $P' \equiv P \Diamond T$. The proof is done by induction on the depth of T (i.e., on the length of the longest path in T).

Lemma 6.
Let P be a πOBDD. If P' is a πOBDD obtained from P by applying a sequence of elementary operations, then there is a πOBDD-transformer T such that P' is equivalent to $P \Diamond T$.

Idea of the proof.
The corresponding transformer is constructed inductively, with respect to the number of elementary operations that are applied to transform P into P'. $\Box$

Example B: A transformer for the transformation described in Fig. 1 has three nodes. The root is labeled by $\overline{x_1}$ and has negation mark -1. Its both outgoing edges enter the same node labeled by $\overline{x_2}$ with the negation mark 1. The outgoing edges of this second node enter the same sink labeled by $\perp$.

5 Examples of OBDD-Transformations

According to Theorem 4 and Proposition 1 and 2, transformers provide an alternative formalism for OBDD-transformations which is supported by the output-efficient algorithms *Derive* and *Compose*. The effect of this formalism can be illustrated by several interesting examples. Due to a lack of space the examples are moved to Appendix I.

6 Complexity-Bounded OBDD-Transformations

It is easy to see that the relation $\leq_{OBDD}$ defined via OBDD-transformations is *reflexive* and *transitive*. Even more, we can show that $\leq_{OBDD}$ is *symmetric*, too, and, hence, that all problems are equivalent with respect to *unbounded* OBDD-transformations. This is neither surprising nor disturbing, as long as we allow to apply any sequence of elementary operations, no matter how long it is.

Proposition 7.
Each problem (f, π) is transformable to $(1, \sigma)$, where σ is any variable ordering and, vice versa, $(1, \sigma)$ is transformable to any given problem (f, π).

Proof. The proof is based on the application of algorithm *Derive*. Each OBDD P can be seen as an OBDD-transformer that fulfills the relation $\mathbf{1} \Diamond P = P$ and $P \Diamond \mathbf{1} = \mathbf{1}$, where $\mathbf{1}$ denotes the reduced OBDD for the constant function 1. $\Box$

Corollary 8.
Any two problems are OBDD-transformable to each other. $\Box$

6.1 Complexity Measure

In order to use OBDD-transformations for the investigation of complexity theoretic properties of OBDDs, we have to restrict their computational resources in an appropriate manner. As already mentioned, the complexity measure of interest is the OBDD-size. Since, due to Propositions 3 and 4 together with Theorem 4, the size of the OBDD constructed by the transformation can be estimated in terms of the size of the applied transformer, it suffices to concentrate consideration on the size-bounded transformers in order to obtain results of complexity theoretic relevance.

Definition.
Let $r(n)$ be a function on the natural numbers. A problem (f, π) is called an $r(n)$-bounded OBDD-transformation *of (g, σ), denoted by $(f, \pi) \leq^{r(n)}_{OBDD} (g, \sigma)$, if there is a sequence of OBDD-transformers (T_n) such that*

(i) for each n there is an m such that π_n OBDD(f_n) and σ_m OBDD(g_m)$\diamond T_n$ are equivalent up to a renaming of the variables, and

(ii) the sequence of transformers $(T_n)^{\infty}_{n=1}$ is $r(n)$-bounded, *i.e., it holds size(T_n) $\leq r(n)$ for each n.*

If $\mathcal{C}$ is a class of functions, then the problem (f, π) is a $\mathcal{C}$–OBDD-transformation of the problem (g, σ), denoted by $(f, \pi) \leq^{C}_{OBDD} (g, \sigma)$, if there is a function $r \in \mathcal{C}$ such that $(f, \pi) \leq^{r(n)}_{OBDD} (g, \sigma)$.

6.2 Basic Properties and Examples

The transformations described in Examples C, D and E of Section 5 are linearly bounded. Unlike that, the operation described in Example F provides a transformation that may be exponential with respect to the number of variables, depending on the position of the redirected edge. The observation that the size of P is not increased by the application of the transformer T on it, may be a reason for the question whether the bounded transformation is well-defined. The explanation is very simple, T is applicable on many other OBDDs (e.g., **1**) whose sizes grow exponentially by application of T. The size of the transformer gives us only an upper bound on the growth of the resulting OBDD.

No matter how much we restrict the size of the transformers, the corresponded transformations remain reflexive in any case. The transitivity is a more sensitive property. According to Propositions 1 and 2, transitivity is fulfilled in the cases of e.g., constantly, polylogarithmically, or polynomially bounded transformations. The transformation might not be transitive e.g. in the case of linearly bounded ones. Generally, we have

Proposition 9.
For every class of functions $\mathcal{C}$, $\mathcal{C}$–OBDD-transformations are transitive if and only if, for every $r_1, r_2 \in \mathcal{C}$, there is an $r \in \mathcal{C}$, such that $r \geq r_1 r_2$. $\square$

A noteworthy phenomenon is that the symmetry property of unbounded transformations of Proposition 7 is no more true if one considers bounded transformations. For example, if one cuts out an exponentially large subOBDD by Operation 4, then an exponentially large transformer is needed to 'repair' this.

6.3 Polynomial OBDD-Transformations

Size bounded transformers provide a relevant reduction tool for the investigation of complexity classes defined in terms of OBDDs. Remember, if $\mathcal{C}$ is a class of functions, then the complexity class $\mathcal{C}_{OBDD}$ is defined to consist of all problems (f, π) for which, for every n, the size of the reduced π_nOBDD(f_n) is bounded by $r(n)$ for some $r \in \mathcal{C}$. Particularly, $\mathcal{P}_{OBDD}$ denotes the set of problems representable by means of polynomially bounded OBDDs, i.e.,

$$\mathcal{P}_{OBDD} = \{(f, \pi) \,|\, \text{there is a polynomial } p(n) \text{ with } size(\pi_n \mathrm{OBDD}(f_n)) \leq p(n) \text{ for all } n\}.$$

Now, let $\leq^{p}_{OBDD}$ denote the relation defined by means of polynomially bounded OBDD-transformations. Two problems Π and Σ are said to be *equivalent w.r.t. polynomial OBDD-transformation* (denoted by $\Pi \equiv^{p}_{OBDD} \Sigma$) if $\Pi \leq^{p}_{OBDD} \Sigma$ and $\Sigma \leq^{p}_{OBDD} \Pi$.

Proposition 10.
The relation $\leq^{p}_{OBDD}$ is transitive.

Proof. The transitivity follows from Propositions 1 and 2 combined with renaming of variables. $\square$

$\mathcal{P}_{OBDD}$ is the basic class in the hierarchy of complexity classes defined by polynomially bounded OBDD-transformation.

Proposition 11.
Let Π *be a problem from* $\mathcal{P}_{OBDD}$ *and* σ *any variable ordering. Then* $\Pi \equiv^p_{OBDD} (1, \sigma)$.

Proof. The construction of the transformers is the same as in the proof of Proposition 7. □

Corollary 12.
For any two problems Π *and* Σ *from* $\mathcal{P}_{OBDD}$ *it holds* $\Pi \equiv^p_{OBDD} \Sigma$.

Polynomially bounded OBDD-transformations provide an adequate reduction tool for the investigation of the membership in the class $\mathcal{P}_{OBDD}$, which can be very useful in practical applications of OBDDs.

Corollary 13.
If $\Pi \leq^p_{OBDD} \Sigma$ *then* $\Sigma \in \mathcal{P}_{OBDD}$ *implies* $\Pi \in \mathcal{P}_{OBDD}$ *and* $\Pi \notin \mathcal{P}_{OBDD}$ *implies* $\Sigma \notin \mathcal{P}_{OBDD}$. □

6.4 Polynomial OBDD-Transformations vs. Read-once Projections

According to the definition in [BW96], $f = (f_n)$ is a read-once projection of $g = (g_n)$ if there is a polynomially bounded sequence (p_n) such that for each n:

$$f_n(x_1, ..., x_n) = g_{p_n}(y_1, ..., y_{p_n})$$

where

(i) $y_i \in \{-1, 1\} \cup X_n \cup \{\overline{x_j} \mid x_j \in X_n\}$ for each $1 \leq i \leq p_n$, and

(ii) for any i and j, $i \neq j$, $y_i \in \{x_r, \overline{x_r}\}$ implies $y_j \notin \{x_r, \overline{x_r}\}$. (the so-called read-once property)

The problems considered in [BW96] are defined in terms of sequences of Boolean functions, unlike to this paper, where the variable ordering is taken in consideration as an important attribute of the problem. Roughly spoken, f is a read-once projection of g means that for each sequence $\pi = (\pi_n)$ of variable orderings for f there is a sequence $\sigma = (\sigma_n)$ of variable orderings for g such that π_nOBDD(f_n) and σ_mOBDD(g_m) are related as it is described in the definition. Hence, fine relations between the functions that are visible only with respect to some subset of orderings cannot be expressed in terms of read-once projections (see Example H in Appendix I). On the other hand, the extension of the notion of OBDD-transformations to problems defined as in [BW96] is possible, but it exceeds the aim of this paper.

Proposition 14.
If $f = (f_n)_{n=1}^{\infty}$ *is a read-once projection of* $g = (g_n)_{n=1}^{\infty}$, *then for each sequence* $\pi = (\pi_n)_{n=1}^{\infty}$ *of orderings on variables in the support of* f *and* g, *it holds* $(f, \pi) \leq^{p(n)}_{OBDD} (g, \pi)$, *where* $p(n)$ *is the polynom used in the read-once projection.*

Proof. Omitted.

Corollary 15.
A read-once projection defined by a sequence of equalities $f_n(x_1, ..., x_n) = g_{p_n}(y_1, ..., y_{p_n})$ *can be realized by means of a sequence of OBDD-transformers* (T_n) *with size*$(T_n) = p_n + 1$.

Proof. Omitted.

Examples C, D and E are witnesses that there are linearly bounded OBDD-transformations that are not read-once projections. There are even constantly bounded OBDD-transformations that can not be obtained via any read-once projection. Hence, polynomially bounded OBDD-transformations are definitely stronger than read-once-projections.

Proposition 16.
There are OBDD-transformations which can be realized by means of a single elementary operation, but

cannot be obtained via any read-once projection.

Sketch of the proof. Let us consider the constant functions -1 and 1. Obviously, none of the functions is a read-once projection of the other. On the other side, independently from the variable ordering, the OBDD for -1 can be obtained from an OBDD for 1 (and vice versa) by applying Operation 1. $\square$

Generalizing the OBDDs to multi-rooted OBDDs, we obtain an appropriate representation of multi-valued Boolean functions, i.e., the mappings $\{true, false\}^n \mapsto \{true, false\}^m$, where all output bits are represented at once. OBDD-transformations for this generalized type of OBDDs can be defined in the similar way as in the case of single-rooted OBDDs. A problem Π is reducible to problem Σ if, for each instance Π_n, its OBDD-representation can be obtained from the OBDD-representation of some instance Σ_m. Formally, $(f,\pi) \leq_{OBDD} (g,\sigma)$ if and only if, for each n, there is an m and a multirooted transformer $T_m = (T_m^1, \ldots, T_m^k)$ such that for all i there are j, k with

$$F_n^i \;\equiv\; T_n^k \diamond G_m^j$$

(up to a renaming of the variables), where F_n^i is the πOBDD for the i-th output bit of f_n, $T_{m'}^k$ is the k-th root of the transformer T, and G_m^j is the σOBDD of the $j-$th output bit of g_m.

The most surprising result obtained for read-once projections in [BW96] was the non-reducibility of SQUaring to MULtiplication. We show that this result is not a witness of the higher complexity of the SQUaring. In contrary, it is an additional argument for the need of a more adequate notion of reducibility in the context of restricted complexity classes.

Example H: (SQU,π) $\leq_{OBDD}^p$ (MUL,π'), where $\pi'_{2n} = (\pi(1), \pi(1)+n, \pi(2), \pi(2)+n, ..., \pi(n), \pi(n)+n)$.

For every n, let $\mathrm{SQU}_n^i(x_1, x_2, ..., x_n)$ be the i-th output-bit of the square of the binary number $x_1x_2...x_n$ and $\mathrm{MUL}_n^i(x_1, x_2, ..., x_{2n})$ be the i-th output-bit of the multiplication of the binary numbers $x_1x_2...x_n$ and $x_{n+1}x_{n+2}...x_{2n}$.

We can construct a π'_{2n}OBDD-transformer that transforms a π'_{2n}OBDD for MUL_{2n}^i into a πOBDD for SQU_n^i, for each $i, 1 \leq i \leq n$. For each j, $1 \leq j \leq n$, the π'_{2n}OBDD-transformer contains one node labeled by x_{2j-1}. Each node labeled by x_{2j-1} has two successor nodes labeled by x_{2j}. The true-successor (respectively, the false-successor) has one outgoing edge labeled by 1 (resp., by -1). See Fig. 7 in Appendix I for the corresponding transformer. $\square$

6.5 Deriving Exponential Lower Bounds by Polynomial OBDD-Transformation

In the following, we sketch an example which shows how OBDD-transformations can be used for deriving exponential lower bounds on the OBDD-size.

Let $PERM_n$ be the test whether a given $n \times n$ matrix M with coefficients from $\{0,1\}$ is a permutation matrix, i.e., if there is exactly one 1 in each row and in each column. Let $MAG\text{-}2_n$ denote the test whether a given $n \times n$ matrix M over Z_2 is a magic square, i.e. whether the sum (XOR) of the elements in each row and the sum of each column agree. It holds

$$PERM \leq_{OBDD}^p MAG\text{-}2$$

The transformation is based on the observation that a permutation matrix is a magic matrix whose row sums over Z_2 equal 1 and which overall sum of ones equals the number of rows. For every n, from the input we construct a transformer T_n^1 that excludes from the the inputs the matrices with more or less than n ones. Transformer T_n^2 assures that the sum in a row is equal to 1 in the accepted inputs. It looks like an OBDD for the parity of the variables in one (any chosen) row, where the 1-sink is replaced by the $\perp$-sink. The entire transformer is obtained as $T_n^1 \circ T_n^2$. For details see Appendix IV.

In [KMW88], it was proven that the function $PERM$ requires exponentially large OBDDs for any variable ordering. (This lower bound holds even for a more general model). Due to this fact, the described OBDD-transformation proves an exponential lower bound for the magic square problem.

Corollary 17.
For any variable ordering π, πOBDDs for the functions MAG-2_n have size $2^{\Omega(n)}$.

References

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

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

[BW96] B. Bollig, I. Wegener: Read-once Projections and Formal Circuit Verification with Binary Decision Diagrams. Proc. STACS'96 (1996), 491–502.

[BDG88] J. L. Balcázar, J. Díaz, J. Gabarró: Structural Complexity I., Springer Verlag, 1988.

[Lee90] J. van Leeuwen (edit.): Handbook of Theoretical Computer Science. The MIT Press, 1990.

[KW87] K. Kriegel, S. Waack: Exponential Lower Bounds for Real-time Branching Programs. Proc. FCT'87, LNCS 278 (1987), 263–367.

[KMW88] M. Krause, Ch. Meinel, S. Waack: Separating the Eraser Turing Machine Classes $\mathcal{L}_\epsilon, \mathcal{NL}_\epsilon, co-\mathcal{L}_\epsilon$ and $\mathcal{P}_\epsilon$. Proc. MFCS'88, LNCS 324 (1988), 405-413.

[Lon93] D. Long: BDD-Package, CMU.

[Mei89] Ch. Meinel: Modified Branching Programs and Their Computational Power. LNCS 370, Springer Verlag, 1989.

[Som96] Somenzi: CUDD: CU Decision Diagram Package (Release 1.1.1). University of Colorado at Boulder. June 1996.

[SV81] S. Skyum, L. G. Valiant: A Complexity Theory Based on Boolean Algebra. Proc. 22^{nd}IEEE FOCS (1981), 244-253.

[SW93] D. Sieling, I. Wegener: Reduction of BDDs in linear time. Information Processing Letters 48 (1993) 139-144.

Appendix I

Examples of OBDD-Transformations Defined via OBDD-Transformers

First, we construct transformers for OBDD-transformations between MOD2^i and MOD2 functions. As observed in [BW96] MOD-2 is not reducible to MOD-4 via read-once projections.
Formally, MOD-$2^i_n(x_1, ..., x_n) = 1$ iff $\sum_{j=1}^n x_j$ MOD $2^i = 0$. The presented transformations between MOD-2^i and MOD-2 are constructed with respect to the natural variable ordering. The change to any other ordering is straightforward.

Example C: $\boxed{(\text{MOD-}2, \pi) \leq_{OBDD} (\text{MOD-}4, \pi)}$

The idea behind the construction of the transformer for the transformation is based on the observation that MOD-$2_n(x_1, ..., x_n) = 1$ iff $\sum_{i=1}^n x_i$ MOD 4 equals 0 or 2. The corresponded transformer should force the paths for which $\sum_{i=1}^n x_i$ MOD 4 equals 2 to terminate in a sink labeled by 1. The transformer for $n = 6$ is shown in Fig. 2. $P4$ is an OBDD for MOD-4_6 and $P2$ is an OBDD for MOD-2_6. We leave the OBDDs unreduced since it is easier to see what happens.

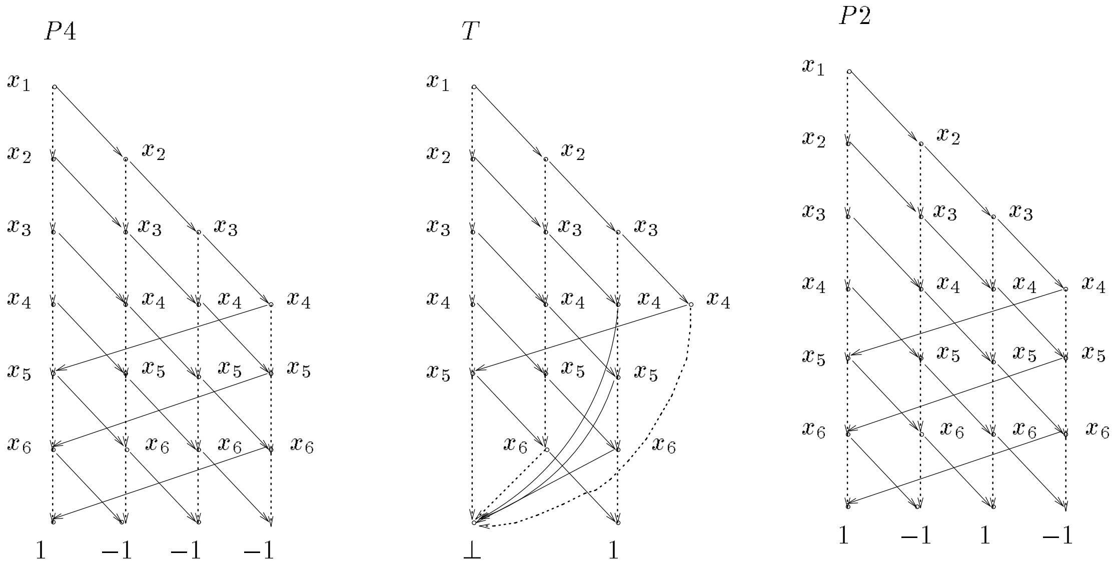


Fig.2 $P4 \diamond T = P2$

Example D: $\boxed{(\text{MOD-2}, \pi) \leq_{OBDD} (\text{MOD-4}, \pi)}$

Another idea how to realize an OBDD-transformation of $(\text{MOD-2}, \pi)$ to $(\text{MOD-4}, \pi)$ is based on the observation that $\text{MOD-2}_n(a_1, a_2, ..., a_n) = \text{MOD-4}_{2n}(a_1, a_1, a_2, a_2, ..., a_n, a_n)$ holds for all $a \in \{0,1\}^n$, $a = (a_1, a_2, ..., a_n)$.

For each n we can construct a transformer with $2n$ variables, where each variable with even index is forced to be set to the same value as the previous variable. The implementation of this idea for $n = 3$ and natural variable ordering is shown in Fig. 6. Transformer S applied on an OBDD for MOD-4_6 yields an OBDD for MOD-2_3, e.g., $Q4 \diamond S = Q2$. After reduction of redundant nodes from $Q2$ (i.e., those nodes that have two equal successors), we obtain an OBDD for MOD-2 over the variables $\{x_1, x_3, x_6\}$. This construction can be easily generalized to transformers that realize the transformation MOD-2 $\leq_{OBDD}$ MOD-2^i, for any i.

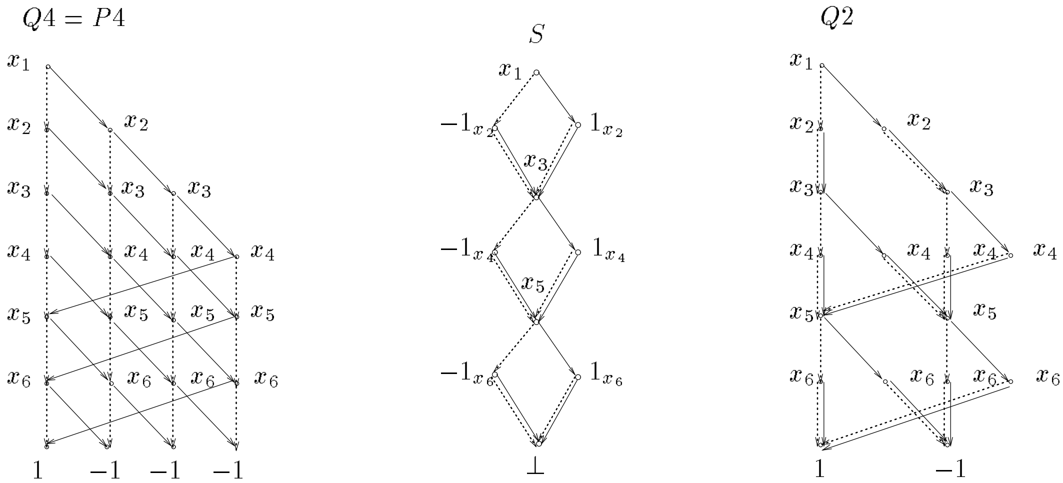


Fig. 6 $Q4 \diamond S = Q2$

Example E: $\boxed{(\text{MOD-4}, \pi) \leq_{OBDD} (\text{MOD-2}, \pi)}$

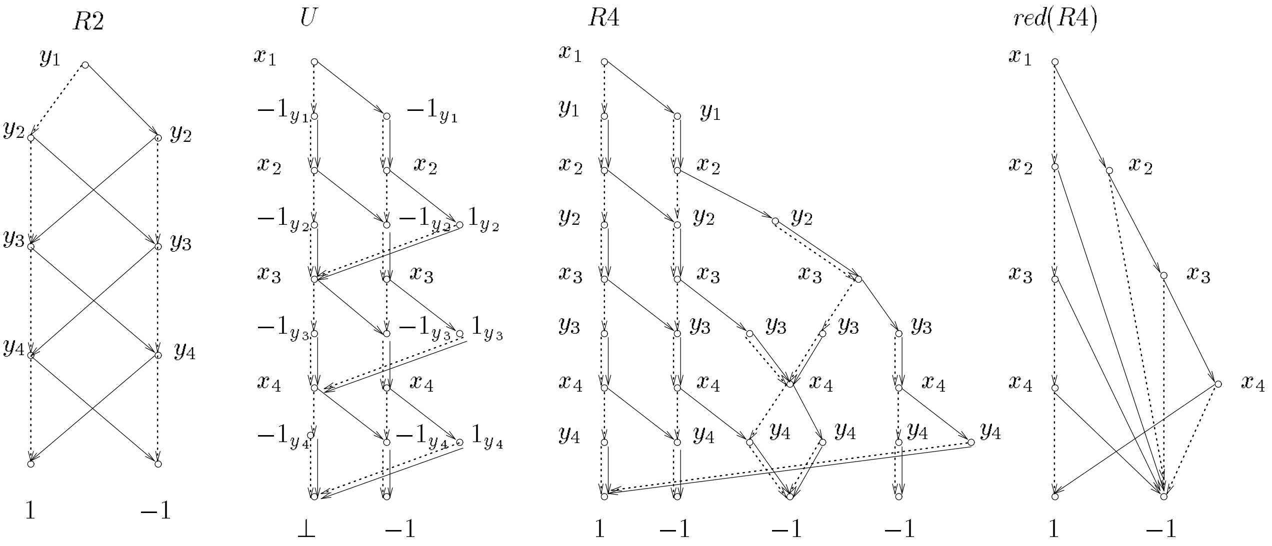


Fig.3 $R4 = R2 \Diamond U$

$R2$ is an OBDD for MOD-2_4, U is a transformer and $R4 = R2 \Diamond U$. red($R4$) is obtained from $R4$ by elimination of the redundant nodes. It is easy to see that MOD-$2^i \leq_{OBDD}$ MOD2, for any i.

Example F: $\boxed{\textit{Redirection of an Edge.}}$

Another example is related to the problem of redirecting an edge of an OBDD P to another node of P. Althought it looks like an elementary operation, it is not. In order to illustrate this observation, let us consider the situation sketched on Fig. 4. Let u, v and w be nodes of P, v be δ-successor of u. We want to redirect the δ-edge of u to w what is shown by the dotted line. Neither u nor v has information about the position of w. A transformer T that simulates this operation on P is constructed as follows. In order to describe the operation, we have to specify which edge should be redirected and towards which node. The idea is the same as in the basis of the proof of Lemma 6. The top part of T is constructed as an isomorphic copy of all root-to-u paths of P. All other edges enter a sink labeled by $\perp$ with negation mark 1. The δ-edge of u enters a root of an OBDD that is isomorphic with the OBDD rooted in w.

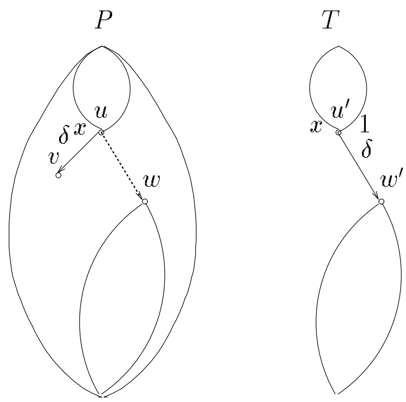


Fig.4

Example H:

$\boxed{(\mathbf{SQU},\pi) \leq^p_{OBDD} (\mathbf{MUL},\pi'), \textbf{ where } \pi'_{2n} = (\pi(1), \pi(1)+n, \pi(2), \pi(2)+n, ..., \pi(n), \pi(n)+n)}$

Transformers for this transformation can be built up on the same principle as in the previous example.

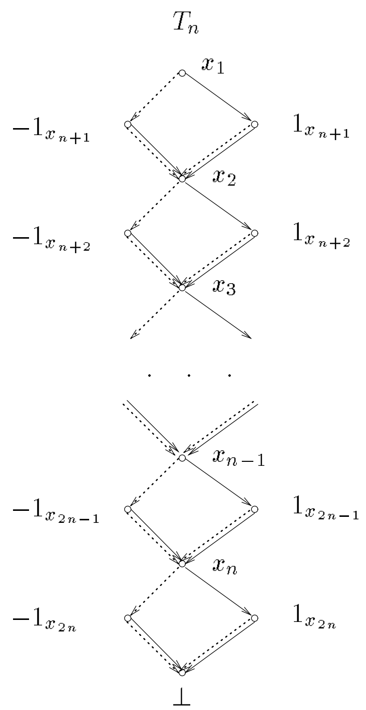


Fig. 7 T_n realizes (SQU,π) $\leq^p_{OBDD}$ (MUL,π')

Appendix II

Derivation of Exponential Lower Bounds

In Section 6.5, we discussed the possibility of deriving new lower bounds on the size of OBDDs from known ones by using OBDD-transformations. Here, we describe the transformation from *PERM* to *MAG-2* in detail.

W.l.o.g, the functions $PERM_n$ and $MAG\text{-}2_n$ are assumed to be defined over the same set of variables $\{x_i \,|\, 1 \leq i \leq m = n^2\}$. Zero is encoded by -1. The transformation is based on the following observation: A given matrix is a permutation matrix if and only if it is a magic matrix with n ones whose row sum and column sum over Z_2 (i.e. parity) is equal to 1. The left-to-right implication of this statement is clear. Let us consider a magic matrix with the given properties. Since the parity of each row and column coincide and is equal to 1, there is no row or column consisting merely of zeros. On the other hand, the existence of exactly n ones assures that no row and/or column has more than one 1. Otherwise there would be at least one row or column with zeros only.

In order to illustrate a comfortable work with transformers, we describe the desired transformation via the composition of two transformers which filter out those inputs from *MAG-2* that have not the desired properties mentioned above.

The transformer T_n^1 allows those inputs from $MAG\text{-}2_n$, where exactly n variables are 1. Transformer T_n^2 checks the parity in one chosen row or column. (Since the inputs accepted by *MAG-2* have equal row and column sums, any row or column can be chosen.) The transformers are shown on Fig. 8. Since the labels of variables in one level coincide, only the leftmost nodes are labeled in the figure. Variables $x_{i\,1}, x_{i\,2}, \ldots, x_{i\,n}$ in T_n^2 that correspond to the coeficients in the chosen row/column occur consistently with π_n.

T_n^1 and T_n^2 are both π_n-transformers and we can build a transformer $T_n = T_n^1 \circ T_n^2$ which excludes all matrices from the consideration that have either more or less than n ones, or have a row whose sum is distinct from 1. The size of T_n is bounded by the product of the sizes of T_n^1 and T_n^2 which is $O(n^3)$. Due to the observation that the application of *Compose* on (T_n^1, T_n^2) at most doubled the number of nodes on each level of T_n^1 we can improve the estimation of the $size(T_n)$ to $O(n^2)$. The resulting transformer T_n can be constructed directly by the following top-down procedure: The nodes of T_n form $(m+1)$ levels.

Ech node corresponds to some triple (i,j,k), where $0 \le i \le n$, $j \in \{0,1\}$, and $1 \le k \le m$. The meaning of the first element of the triple is the sum of ones so long read, the second element represents the parity of the coeficients in the chosen row that have been already read, and the third element represents the level. Level 0 of T_n consists of one node – the root that corresponds to a triple $(0,0,1)$. The nodes on level k, $1 \le k \le m$, are labeled by $x_{\pi(k)}$. If $x_{\pi(k)}$ is a variable from the chosen row, then, for the node that corresponds to (i,j,k), two nodes in level $(k+1)$ are constructed. The one that corresponds to $(i,j,k+1)$ becomes its false-son, the other that corresponds to $(i+1, j \oplus 1, k+1)$ becomes its true-son. If $x_{\pi(k)}$ does not belong to the chosen row, then the sons of the node that corresponds to (i,j,k) are the nodes $(i,j,k+1)$ as then-son, and $(i+1,j,k+1)$ as the false-son. All nodes on the bottom level are sinks labeled by -1 with exception of the node that correspond to $(n,1,m+1)$ which is labeled by $\perp$.

Applying T_n on an π_nOBDD(*MAG-2$_n$*) P_n we obtain an π_nOBDD(*PERM$_n$*) S_n. Since T_n has no sinks labeled by 1, the inputs accepted by P_n could be accepted by S_n, too.

On the other hand, all inputs, with the sum of ones distinct from n are forced to terminates in a sink labeled by -1. The remaining inputs not considered yet are the inputs from *MAG-2$_n$* with exactly n ones. They are accepted only in the case that the row sum is equal to 1. Those inputs are in *PERM$_n$*, too.

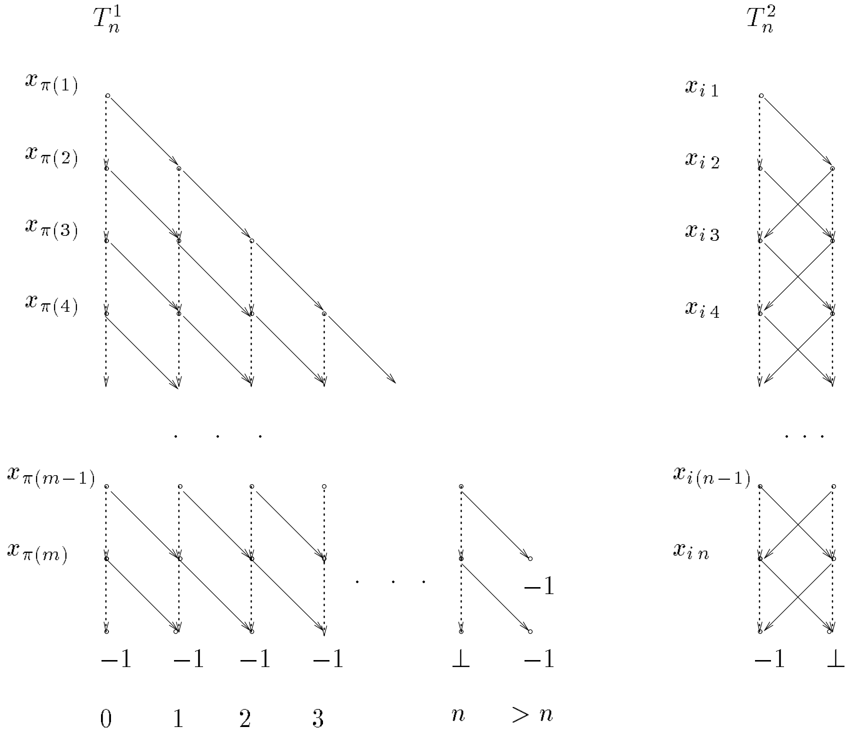


Fig. 8