published in
Proceedings of Int. Symposium on Multiple Valued Logic,
Portland, USA, 2000, pp. 233-238.

Mod-p Decision Diagrams: A Data Structure for Multiple-Valued Functions

Harald Sack
FB IV - Informatik
Universität Trier
D-54286 Trier, Germany
sack@uni-trier.de

Elena Dubrova
Department of Electronics
Royal Institute of Technology
S-164 40 Kista, Sweden
elena@ele.kth.se

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

Abstract

Multiple-valued decision diagrams (MDDs) give a way of approaching problems by using symbolic variables which are often more naturally associated with the problem statement than the variables obtained by a binary encoding. We present a more general class of MDDs, containing not only branching nodes but also functional nodes, labeled by addition modulo p operation, p - prime, and give algorithms for their manipulation. Such decision diagrams have a potential of being more space-efficient than MDDs. However, they are not a canonical representation of multiple-valued functions and thus the equivalence test of two Mod-p-DDs is more difficult then the test of two MDDs. To overcome this problem, we design a fast probabilistic equivalence test for Mod-p-DDs that requires time linear in the number of nodes.

1 Introduction

Many problems in digital logic design, combinatorial optimization and formal verification can be expressed as a sequence of operations on discrete functions. Such applications would benefit from a data structure, suitable for both compact representation for the computer's internal use and fast manipulation in algorithms. For the case of Boolean functions, Reduced Ordered Binary Decision Diagrams (ROBDDs) [1] have proved to be well qualified for this purpose. ROBDDs can be extended to discrete case in different ways, depending on the decomposition applied to the function in the nodes of the diagram. For example, [2] presented a generalization of ROBDDs into *Multiple-Valued Decision Diagrams* (MDDs), representing multiple-valued functions $M^n \to M$, over a finite set of totally ordered values $M = \{0, 1, \ldots, m-1\}$. The conventional ITE-algorithm is extended for this purpose into the CASE-algorithm, utilizing the *generalized Boole/Shannon decomposition* [3]:

$$f(x_1, \ldots, x_n) = \\ \overset{0}{x}_i \cdot f|_{x_i=0} + \overset{1}{x}_i \cdot f|_{x_i=1} + \ldots + \overset{m-1}{x_i} \cdot f|_{x_i=m-1}$$

where $f|_{x_i=j}$ are the cofactors of f defined by $f|_{x_i=j} = f(x_1, \ldots, x_{i-1}, j, x_{i+1}, \ldots, x_n)$ for all $i \in \{1, 2, \ldots, n\}$, $j \in M$, and "$+$", "$\cdot$" denote the multiple-valued operations MAX, MIN, correspondently. $\overset{i}{x}$ is a unary operation *literal* of x, defined by

$$\overset{i}{x} = \begin{cases} m-1 & \text{if } x = i, \\ 0 & \text{otherwise.} \end{cases} \tag{1}$$

A similar generalization, defined for the discrete functions of type $P_1 \times P_2 \times \ldots P_n \to M$, where $P_i = \{0, 1, \ldots, p_i - 1\}$ are sets of values the variables x_i assume, has been presented in [4]. A survey of different multiple-valued decision diagrams is given in [5].

In this paper we introduce a new type of MDDs, extending the concept of Parity-OBDDs [6] to the multiple-valued case and representing functions of type $f : M^n \to M$, with $M = \{0, 1, \ldots, p-1\}$, p - prime. We call them *Mod-p Decision Diagrams* (Mod-p-DDs). Such decision diagrams have a potential of being more space-efficient than MDDs. The size of a minimum size Mod-p-DD for a given multiple-valued function f is never greater than a minimum size MDD for f.

However, Mod-p-DDs do not provide a canonical representation of multiple-valued functions. For non-canonical representations, testing the equivalence of two graphs is much more difficult than for canonical ones (i.e. NP-complete). The speed of equivalence testing crucially affects the efficiency of synthesis of decision diagrams. Synthesis becomes an exponential operation if there is no cache available to look up, rather than re-compute, the result of operations that already occurred at a previous step of the computation. Looking up this cache involves checking the equivalence of the graphs of the current and the cached operation.

For the Boolean case, the fastest known deterministic equivalence test for non-canonical Parity-OBDDs, presented in [7], requires time cubic in the number of nodes. Hence, it doesn't seem to be suitable for practical purposes. In [8], a fast probabilistic equivalence test for Parity-OBDDs has been proposed that requires time at most linear in the number of nodes. In this paper we extend this algorithm to the multiple-valued case.

The paper is structured as follows. In Section 2, Mod-p-DDs are introduced. Section 3 describes the algorithm for deciding the equivalence of two Mod-p-DDs probabilistically. Section 4 presents the reduction and synthesis algorithms for Mod-p-DDs. Section 5 concludes the paper with an outlook of work to be done.

2 Definition of Mod-p-DDs

In this section we define Mod-p decision diagrams and show some of their properties.

Definition 1 A Mod-p Decision Diagram P is a rooted, directed acyclic graph $P = (V, E)$ with node set V containing two types of nodes: terminal and non-terminal. A *terminal* node v has as attribute a value $value(v) \in M$. A *non-terminal* node has as attributes either a variable index $index(v) \in \{1, 2, \ldots, n\}$ (*branching node*), or the p-ary operation addition modulo p, p - prime, (*$\oplus_p$-node, functional node*) and p children $child_i(v) \in V, i \in M$.

Definition 2 A Mod-p Decision Diagram is *ordered* if, for any non-terminal branching node v and for all $i \in M$, if $child_i(v)$ is also non-terminal, then it holds that $index(v) < index(child_i(v))$.

Definition 3 A Mod-p Decision Diagram is *reduced* if it contains no vertex v with $child_i(v) = child_j(v)$, for any $i, j \in M, i \neq j$, nor does it contain distinct vertices v and v' such that the subgraphs rooted by v and v' are isomorphic.

The function associated with the Mod-p-DDs is determined in the following way:

Definition 4 A Mod-p Decision Diagram P having root node v represents a function f_v defined recursively as follows

1. If v is a terminal node carrying the value $\delta_i \in M$, then $f_v = \delta_i$.

2. If v is a non-terminal branching node with $index(v) = i$, then f_v is the function

   $$f_v(x_1, \ldots, x_n) = \overset{0}{x}_i \cdot f_{child_0(v)} + \overset{1}{x}_i \cdot f_{child_1(v)} + \ldots + \overset{p-1}{x_i} \cdot f_{child_{p-1}(v)},$$

   where " $+$ " and " $\cdot$ " denote the multiple-valued operations MAX and MIN, and $\overset{i}{x}$, $i \in M$, is the *literal* defined by (1).

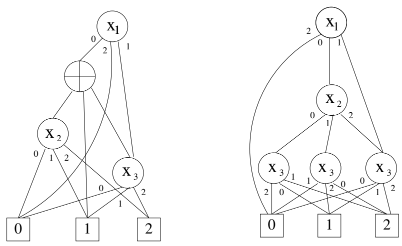
3. If v is a $\oplus_p$-node, then f_v is the function

   $$f_v(x_1, \ldots, x_n) = f_{child_0(v)} \oplus_p f_{child_1(v)} \oplus_p \ldots \oplus_p f_{child_{p-1}(v)},$$

   where " $\oplus_p$ " denotes the operation addition modulo p.

It is easy to see that MDDs are just a special case of Mod-p-DDs, namely Mod-p-DDs without $\oplus_p$-nodes. Therefore, the size of a minimum sized Mod-p-DD for a given multiple-valued function f is not greater than the size of a minimum size MDD for f.

But, for a fixed variable order, a function can be represented by several different Mod-p-DDs, with different $\oplus_p$-node placement. Thus, the data structure becomes non-canonical. As an illustration, consider a 3-variable 3-valued function defined by the table in Figure 1. There, two different Mod-p-DDs for this function, for the order $< x_1, x_2, x_3 >$, are shown. Functional nodes are represented " $\oplus$ ". The three children of non-terminal branching nodes are indicated by the edges labeled by $0, 1, 2$. The Mod-p-DD on the right does not contain any functional nodes, i.e. it is equivalent to the MDD of the function.



$x_3 \backslash x_1 x_2$	00	01	02	12	11	10	20	21	22
0	1	2	0	0	0	0	0	0	0
1	2	0	1	1	1	1	0	0	0
2	0	1	2	2	2	2	0	0	0

Figure 1. Two different Mod-p-DDs of the same given function.

3 Probabilistic Equivalence Test for Mod-p-DDs

Since Mod-p-DDs do not provide a canonical representation of multiple-valued functions, testing the equivalence of two graphs becomes an essential problem. In this section we show that the equivalence of Mod-p-DDs can be decided probabilistically in linear time, by extending the probabilistic equivalence test for Parity-OBDDs [8] to the multiple-valued case. Our extension employs the concept of multiple-valued signatures introduced in [9] for identifying the equivalence of two multiple-valued functions probabilistically.

In the Boolean case, Parity-OBDDs are a special case of Ω-OBDDs, allowing so called *functional nodes* labeled by an element of a basis Ω of binary Boolean functions. It was shown that while the equivalence test for all Ω-OBDDs, $\Omega \in \{\{\vee\}, \{\wedge\}, \{\vee, \wedge\}\}$ is **co-NP**-complete, for $\Omega \in \{\oplus\}$ it is within **co-R** [8]. The probabilistic equivalence test for Parity-OBDDs proposed in [8] needs only linearly many arithmetic operations in the number of nodes in the graph. Equivalence of two Parity-OBDDs is determined by an algebraic transformation of the functions represented by the graphs to polynomials over a finite field of integers modulo p. A detailed description of the transformation is given in [10].

We extend the probabilistic equivalence test for Parity-OBDDs [8] to the multiple-valued case. The equivalence of two Mod-p-DDs is determined by an algebraic transformation of the Mod-p-DDs in terms of polynomials over a finite field of integers modulo p. This algebraic transformation was introduced in [9].

Let $GF(p^k)$ be a Galois Field with p^k elements of characteristic p, p - prime, $k > 0$.

Definition 5 Let P be a Mod-p-DD representing a multiple-valued function $f : M^n \to M$. With each node $v \in P$ we associate the *polynomial* $p_v : (GF((p^k))^n \to GF(p^k)$ defined in the following way:

1. $p_v = \delta_i$, if v is a terminal node carrying the value $\delta_i \in M$,
2. $p_v = \sum_{j=0}^{p-1} \left(\prod_{\forall r \in M - \{j\}} \frac{r - x_i}{r - j} \right) \cdot p_{child_j(v)}$, if v is a non-terminal branching node with $index(v) = i$,
3. $p_v = \sum_{j=0}^{p-1} p_{child_j(v)}$, if v is a $\oplus_p$-node,

where the operations " $+$ ", " $-$ " and " $\cdot$ " are carried out in the field $GF(p^k)$.

The *polynomial of* P, $p(P)$, is the polynomial associated with the root node of P. It was shown in [9], that this polynomial is unique for a given function. Therefore, $p(P)$ remains unchanged for different Mod-p-DDs of the same function. Let $|P|$ denote the number of nodes of a given Mod-p-OBDD P. It is easy to see from Definition 5, that $p(P)$ can be computed with $p \cdot |P|$ many additions, at most $2p^2 \cdot |P|$ many subtractions and at most $2p^2 \cdot |P|$ multiplications. Using $GF(p^k)$ of characteristic p, p - prime, simplifies the polynomial for addition modulo p operation to $p_{f_1 \oplus f_2} = p_{f_1} + p_{f_2}$. If we consider the elements of $GF(p^k)$ as p-ary vectors of length k, then field addition can be performed in constant time by bitwise addition modulo p. Multiplication and subtraction of two p-ary vectors of length k can be carried out in time $\lceil \log p \rceil k$. Therefore, $p(P)$ can be computed in at most $p \cdot |P| + 2p^2 \cdot |P| \cdot \lceil \log p \rceil k + 2p^2 \cdot |P| \cdot \lceil \log p \rceil k$ time. Since p and k are constants, the complexity of computing $p(P)$ is bounded by $O(|P|)$. Note, that if the computation of $p(P)$ is performed on a Mod-p-DD bottom-up, then the complexity of computing the polynomial for a given node takes only constant time, because the polynomials for all its successor nodes have already been computed.

Now we present an algorithm for probabilistic equivalence test of two Mod-p-DDs:

Input: Mod-p-DDs P_1 and P_2 representing p-valued functions of type $M^n \to M$.

Output: If P_1 and P_2 are equivalent, then the algorithm always answers "yes". Otherwise the result is "no" with probability greater than $1/p$.

Assumption: $GF(p^k)$ is a finite field of integers modulo p, p - prime, with more than pn elements.

procedure *equivalence*(P_1, P_2);
begin
  choose independently and uniformly $x_1, x_2, \ldots, x_n$
    from $GF(p^k)$;
  compute $p(P_1)$ in $GF(p^k)$;
  compute $p(P_2)$ in $GF(p^k)$;
  if $(p(P_1) = p(P_2))$
  then
    return("yes"); $/*$ P_1 and P_2 are equivalent $*/$
  else
    return("no"); $/*$ P_1 and P_2 are not equivalent $*/$
end.

Figure 2. Algorithm for probabilistic equivalence of two Mod-p-DDs.

Theorem 1 *The Algorithm in Figure 2 probabilistically decides equivalence of two Mod-p-DDs in linear time.*

Proof: To check whether two Mod-p-DDs, P_1 and P_2, are equivalent, the algorithm computes polynomials for P_1 and P_2 and evaluates them on an independently and uniformly random assignment of values from $GF(p^k)$ for the variables $x_1, x_2, \ldots, x_n$. The resulting numbers are called *integer hash codes* (or *signatures*) of f. The hash codes for two equivalent functions are always the same [9]. Therefore, if P_1 and P_2 represent the same function, then

$p(P_1)(x_1, x_2, \ldots, x_n) = p(P_2)(x_1, x_2, \ldots, x_n)$ for any assignment of variables from $GF(p^k)$.

Consider the case when P_1 and P_2 are not equivalent. According to Schwartz-Zippel Theorem [12, p. 165], if the assignments of values of variables $x_1, \ldots, x_n$ are taken independently and uniformly at random from a field F of size $|F|$, then $p(P_1)$ and $p(P_2)$ can be distinguished with the probability at least $\frac{n}{|F|}$. So, if the size of the field GF is greater than pn, then the above algorithm distinguishes P_1 from P_2 with the probability at least $1/p$.

Since the complexity of computing $p(P)$ is bounded by $O(|P|)$, the worst-case complexity of the above algorithm is $O(max(|P_1|, |P_2|))$.

□

Next, we give an estimation of the probability of collision, i.e. the probability that during the synthesis on Mod-p-DDs the signatures for two nodes representing different multiple-valued functions are computed to be equal.

Lemma 1 *By using s different signatures per node, the probability of collision is at most*

$$\epsilon < \frac{|P|^2 \cdot n^s}{2 \cdot |GF(p^k)|^s}$$

Proof: According to Schwartz-Zippel Theorem [12, p. 165], if the assignments of values of variables $x_1, \ldots, x_n$ are taken independently and uniformly at random from a field F of size $|F|$, then the polynomials associated with two different nodes can be distinguished with the probability at least $\frac{n}{|F|}$. Blum [11] has shown that with s parallel signatures, the risk of pairwise collision is at most $\frac{n^s}{|F|^s}$. Among $|P|$ considered nodes, there are $\frac{|P|^2}{2}$ pairs of nodes, and therefore the chance of having at least one possible collision among them is less than $\frac{|P|^2 \cdot n^s}{2 \cdot |GF(p^k)|^s}$.

□

The probability of error depends on the number of elements in $GF(p^k)$, therefore it can be reduced by enlarging the size of the field. It can also be reduced by using several different signatures per node with different random assignment from $GF(p^k)$.

4 Operations on Mod-p-DDs

In this section we describe operations involved in reduction and synthesis of Mod-p-DDs.

4.1 Reduction rules for Mod-p-DDs

Mod-p-DDs can be reduced in the same manner as MDDs [2], [13]. In a Mod-p-DD, a branching node is redundant if all p of its out-coming edges point to the same node. Then, the node can be replaced by reconnecting all its incoming edges to its child (*simple reduction* or *deletion rule*). Identification of isomorphic subgraphs forms the second reduction rule (*algebraic reduction* or *merging rule*).

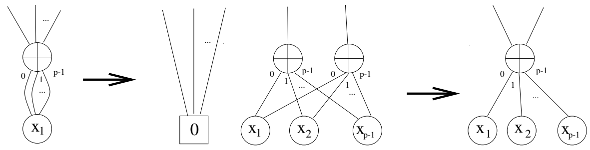


Figure 3. Deletion rule and merging rule for $\oplus$-nodes.

For a $\oplus_p$-node, the merging rule is applied in a similar way as for branching nodes (see Figure 3). The deletion rule differs in that a $\oplus_p$-node with p of its out-going edges pointing to the same node is substituted by the 0-terminal node (see Figure 3). This rule follows from the property of modulo p addition - the modulo p sum of p identical values is zero.

4.2 Synthesis of Mod-p-DDs

For describing the Mod-p-DDs synthesis algorithm we assume that the reader is familiar with standard BDD synthesis algorithms [14]. We implement all multiple-valued operations, except addition modulo p, by means of the CASE-$\oplus$ operator, which is an extension of ITE-$\oplus$ operator, used in case of Boolean Parity-OBDDs [15]. For addition modulo p we directly create a $\oplus_p$-node in the graph. To extend the ITE-$\oplus$ operator, the creation of the cofactors $f|_{x_i=j}, j \in M$ has to be adapted for Mod-p-DDs.

The input parameters of the CASE-$\oplus$ operator are, in general, multiple-valued functions given in the form of Mod-p-DDs. The task is to generate the resultant function $h =$CASE-$\oplus(f, g_0, g_1, \ldots, g_{p-1})$ recursively.

If f is a variable x, then the function returned by CASE-$\oplus$ corresponds to a branching node with a top variable x and with children functions $g_0, g_1, \ldots, g_{p-1}$:

$$\text{CASE-}\oplus\ (x, g_0, g_1, \ldots, g_{p-1}) = (x, g_0, g_1, \ldots, g_{p-1})$$

Moreover, it holds that

$$\text{CASE-}\oplus\ (f, 0, 1, \ldots, p-1) = f$$

If f is an $\oplus_p$-operation, then the function returned by CASE-$\oplus$ corresponds to a functional node with children $g_0, g_1, \ldots, g_{p-1}$:

$$\text{CASE-}\oplus\ (\oplus, g_0, g_1, \ldots, g_{p-1}) = (\oplus, g_0, g_1, \ldots, g_{p-1})$$

The above three equations form the terminal cases for our recursive algorithm.

If f is a complex function, then we first recursively compute the CASE of its cofactors, and then compose them using Boole/Shannon decomposition. To speed up the performance of the CASE-$\oplus$ operation, we are using a *computed table*, which is organized as a hash based cache, to store and recall the results. Before a new node is created, we always refer to a *unique table* organized as a hash table, to prevent the creation of already allocated nodes. In both, *computed table* and *unique table*, every reference is made by application of the probabilistic equivalence test to identify the underlying Mod-p-DDs. A node v with $index(v) = i$ is represented by an $p+1$-tuple $(x_i, v_0, v_1, \dots, v_{p-1})$, with v_j being the node connected to $child_j(v)$. To avoid redundant entries in the *computed table* we transform the $p+1$-tuple to a standard form by reordering it.

procedure *CASE-*$\oplus(f, g_0, g_1, \dots, g_{p-1})$
begin
  transform_to_standard_tuple$(f, g_0, g_1, \dots, g_{p-1})$;
  if *terminal_case*$(f, g_0, g_1, \dots, g_{p-1}, res)$
  then
    return *res;*
  reorder_tuple_acc_to_variable_order$(f, g_0, g_1, \dots, g_{p-1})$;
  if *in_computed_table*$(f, g_0, g_1, \dots, g_{p-1}, res)$
  then
    return *res;*
  if *f*=$\oplus_p$
  then
    res=new_node(*lab*=$\oplus_p$,*child*$_0 = g_0, \dots,$*child*$_{p-1} = g_{p-1}$);
  else
    begin
      for $j = 0$ **to** $(p-1)$ **do**
        $h|_{x=j}$=*CASE-*$\oplus(f|_{x=j}, g_0|_{x=j}, \dots, g_{p-1}|_{x=j})$;
      if *signature*$(h|_{x=0})$=*signature*$(h|_{x=2})$=...
        ...=*signature*$(h|_{x=p-1})$
      then
        res = $h|_{x=0}$;
      else
        res=new_node(*lab=x,child*$_0 = h_{x=0}, \dots$
        ...,*child*$_{p-1} = h_{x=p-1}$);
      insert_in_computed_table$(f, g_0, g_1, \dots, g_{p-1}, res)$;
    end;
  find_or_add_in_unique_table(*res*);
  return *res;*
end.

Figure 4. CASE-$\oplus$ algorithm for Mod-p-DD synthesis.

The pseudo-code of CASE-$\oplus$ is shown in Figure 4. First, the algorithm checks the terminal cases. If the resulting function has already been computed and stored in the unique table, then it is returned. Further, if $f = \oplus_p$, then a new functional node with children $g_0, g_1, \dots, g_{p-1}$ is created. If f is a branching node, then the cofactors $h|_{x=j}$ of the function h are computed by calling CASE-$\oplus$ recursively with the cofactors $f|_{x=j}, g_0|_{x=j}, g_1|_{x=j}, \dots, g_{p-1}|_{x=j}$ as its arguments. These are composed using Boole/Shannon decomposition as $(x, h_0|_{x=0}, h_1|_{x=1}, \dots, h_{p-1}|_{x=p-1})$.

The adaption of the algorithm involves the creation of Mod-p-DDs for cofactors $f|_{x_i=j}, j \in M$ of a multiple-valued function f associated with a node. For a branching node v with $index(v) = i$, the cofactors are derived by simply returning $child_j(v)$ of v. For an $\oplus_p$-node v, creating the cofactors with respect to a variable x_i necessitates the allocation of a new $\oplus_p$-node connected to the cofactors of the p children of v (see Figure 5), if this node does not already exist in the Mod-p-OBDD.

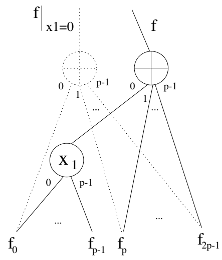


Figure 5. Cofactor creation $f|_{x_1=0}$ for Mod-p-DDs.

5 Conclusion

In this paper, a new data structure for representation and manipulation of multiple-valued logic functions - Mod-p-DDs - is introduced and algorithms for its manipulation are given. Mod-p-DDs have a potential of being more space-efficient than MDDs. However, they are not canonical and therefore the equivalence test of two Mod-p-DDs is more difficult then the test of two MDDs. To overcome this problem, we design a fast probabilistic equivalence test for Mod-p-DDs that requires time linear in the number of nodes.

An implementation of the Mod-p-DDs package is subject of currently ongoing research. We are working on two possibilities: (1) direct implementation of Mod-p-DD structure, and (2) implementing a Mod-p-DD by performing an arbitrary encoding the multiple-valued function $M^n \to M$ represented by Mod-p-DD into the Boolean function of type $B^{\lceil \log p \rceil \cdot n} \to B^{\lceil \log p \rceil}$. By manipulating the same ordering between the associated groups of Boolean variables, we can perform the same operations on the Parity-OBDD as on the Mod-p-DD. This allows us to implement Mod-p-DDs using the already developed Parity-OBDD package [15].

References

[1] R.E. Bryant, Graph-based algorithm for Boolean function manipulation, *IEEE Transactions on Computers* **C-35** No. 8 (1986), 677-691.

[2] D. M. Miller, Multiple-valued logic design tools, *Proc. 23rd Int. Symp. on MVL* (1993), 2-11.

[3] J. C. Muzio, T. C. Wesselkamper, *Multiple-Valued Switching Theory*, Adam Hilger Ltd Bristol and Boston, 1986.

[4] A. Srinivasan, T. Kam, S. Malik, R. Brayton, Algorithm for discrete function manipulation, *Proc. of Int. Conference on Computer-Aided Design* (1990), 92-95.

[5] R.S. Stankovic, T. Sasao, Decision Diagrams for Discrete Functions: classification and Unified Interpretation, *Proc. of Asia and South Pacific Design Automation Conference* (1998), 439-446.

[6] J. Gergov, C. Meinel, Mod2-OBDDs: A data structure that generalizes EXOR-sum-of-products and Ordered Binary Decision Diagrams, *in Formal Methods in System Design* **8** (1996), Kluwer Academic Publishers, 273-282.

[7] S. Waack, On the descriptive and algorithmic power of parity ordered binary decision diagrams, *Proc. 14th Symp. on Theoretical Aspects of Computer Science*, **1200** of LNCS, Springer (1997).

[8] J. Gergov, C. Meinel, Frontiers of feasible and probabilistic feasible Boolean manipulation with branching programs, *Proc. 10th Annual Symp. on Theoretical Aspects of Computer Science*, **665** of LNCS, Springer (1993), 576-585.

[9] E. Dubrova, H. Sack, Probabilistic verification of multiple-valued functions, Tech. Report 99-23, University of Trier, FB IV-Informatik, Nov. 1999.

[10] J. Jain, J. Bitner, D. S. Fussell, J. A. Abraham, Probabilistic verification of Boolean functions, *in Formal Methods in System Design* **1** (1992), Kluwer Academic Publishers, 65-116.

[11] M. Blum, A. K. Chandra, M. N. Wegman, Equivalence of free Boolean graphs can be decided probabilistically in polynomial time, *Information Processing letters* **10**, No. 2, (1980), 80-82.

[12] R. Motwani, P. Raghavan, *Randomized Algorithms*, Cambridge University Press, 1995.

[13] T. Kam, T. Villa, R. Brayton, A. Sangiovanni-Vincentelli, Multi-valued decision diagrams: Theory and applications, *Multiple-Valued Logic, An International Journal* **4** (1998), 9-62.

[14] K. S. Brace, R. L. Rudell, R.E. Bryant, Efficient Implementation of a BDD Package, *Proc. of the 27th Design Automation Conference* (1990), 40-45.

[15] C. Meinel, H. Sack, $\oplus$-OBDDs - a BDD structure for probabilistic verification, *in Proc. Workshop on Probabilistic methods in Verification* (1998) 141-151.