published in
Proceedings of IEEE/ACM Int. Workshop on Logic Synthesis (IWLS2000),
Dana Point, CA, USA, 2000, pp. 341-348.

Representation of Multiple-Valued Functions with Mod-p Decision Diagrams

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 logic allows us to formulate problems by using symbolic variables which are often more naturally associated with the problem specification than the variables obtained by a binary encoding. In this paper we present a data structure for representation and manipulation of multiple-valued functions - Mod-p Decision Diagrams (Mod-p-DDs). Mod-p-DDs differ from conventional Multiple-Valued Decision Diagrams (MDDs) in that they contain not only branching nodes but also functional nodes, labeled by addition modulo p operation, p - prime. Mod-p-DDs are potentially much more space-efficient than MDDs. However, they are not a canonical representation 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

In the last few years, major advances in integrated circuit technology made feasible fabrication of several commercial products benefiting from multiple-valued logic, such as 256-Mbit 4-valued flash memory [1] and 4-Gbit 4-valued DRAM [2]. These products can be seen as first steps toward recognition of the increasing role of multiple-valued logic in the next generation of electronic systems. However, for further practical utilization, efficient computer-aided tools for design, testing and verification of multiple-valued logic circuits are needed. Some existing tools, such as Berkeley's tool for verification and synthesis VIS [3], provide a solution for the special case of multiple-valued input binary-valued output functions, but the general problem is still open.

This paper focuses on the problem of efficient representation and manipulation of multiple-valued functions. For the case of Boolean functions, Reduced Ordered Binary Decision Diagrams (ROBDDs) [4] 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, [5] presented a generalization of ROBDDs into *Multiple-Valued Decision Diagrams* (MDDs), representing multiple-valued functions, $M^n \rightarrow M$, over a finite set of totally ordered values $M = \{0, 1, \ldots, m-1\}$. For this purpose the conventional ITE-algorithm [6] is extended into the CASE-algorithm, utilizing the *generalized Boole/Shannon decomposition* [7]:

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

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, $P_1 \times P_2 \times \dots P_n \to M$, where $P_i = \{0, 1, \dots, p_i - 1\}$ are sets of values that the variables x_i assume, has been presented in [8]. A survey of different multiple-valued decision diagrams is given in [9].

In this paper we introduce a new type of MDDs, extending the concept of Parity-OBDDs [10] to the multiple-valued case and representing functions, $f : M^n \to M$, with $M = \{0, 1, \dots, p-1\}$, p - prime. We call it *Mod-p Decision Diagram* (Mod-p-DD). Such decision diagrams have a potential of being more space-efficient than MDDs. 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.

For the Boolean case, the fastest known deterministic equivalence test for non-canonical Parity-OBDDs, presented in [11], requires time cubic in the number of nodes. Hence, it doesn't seem to be suitable for practical purposes. In [12], 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 (Mod-p-DD) 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 a value $value(v) \in M$ attributed. A *non-terminal* node has either a variable index $index(v) \in \{1, 2, \dots, n\}$ (*branching node*), or the p-ary operation addition modulo p, p - prime, (*$\oplus_p$-node, functional node*) attributed, and p children $child_i(v) \in V, i \in M$.

Definition 2 A Mod-p-DD 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-DD 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.

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

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

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

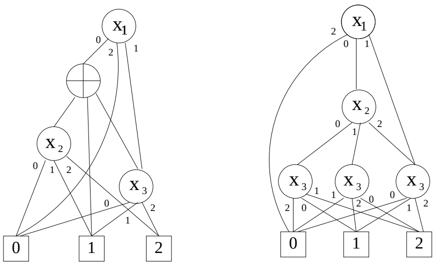
3. If v is a $\oplus_p$-node, then f_v is the function

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

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 an optimal Mod-p-DD for a given multiple-valued function f is not greater than the size of an optimal 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 new data structure is not canonical. As an illustration, consider a 3-variable 3-valued function defined by the table in Figure 1. 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.



$x_3 \backslash x_1x_2$	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 function from the example.

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 [12] to the multiple-valued case. Our extension employs the concept of multiple-valued signatures introduced in [13] for identifying the equivalence of two multiple-valued functions probabilistically.

The probabilistic equivalence test for Parity-OBDDs proposed in [12] 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. A detailed description of the transformation is given in [14]. By extending this test 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 [13].
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* " · " *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 [13], that this polynomial is unique for a given function. Let $|P|$ denote the number of nodes of a given Mod-p-DD 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. 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 $\lceil \log p \rceil k$ steps. Therefore, $p(P)$ can be computed in at most $p \cdot |P| + 2p^2 \cdot |P| \cdot \lceil \log p \rceil k + 2p^2 \cdot |P| \cdot \lceil \log p \rceil k$ steps. 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 a linear time algorithm for probabilistic equivalence test of two Mod-p-DDs:

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

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

Assumption: $GF(p^k)$, 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.

Next, we give an estimation of the probability of a 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 [16, 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 [15] 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 error probability 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

4.1 Reduction rules for Mod-p-DDs

Mod-p-DDs can be reduced in the same manner as MDDs [5], [17]. In a Mod-p-DD, a branching node is redundant if all p of its out-going 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*).

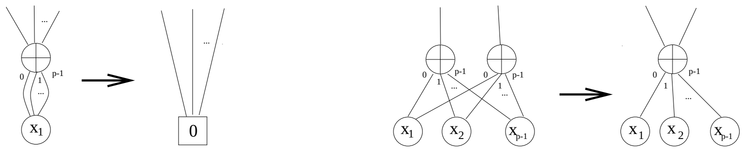


Figure 3: Deletion rule and merging rule for $\oplus$-nodes.

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).

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 [6]. 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 [18]. For addition modulo p we directly create a $\oplus_p$-node in the graph.

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.

```
procedure  CASE-⊕(f, g_0, g_1, ..., g_{p-1})
begin
  transform_to_standard_tuple(f, g_0, g_1, ..., g_{p-1});
  if  terminal_case(f, g_0, g_1, ..., g_{p-1}, res)
  then
    return res;
  reorder_tuple_acc_to_variable_order(f, g_0, g_1, ..., g_{p-1});
  if  in_computed_table(f, g_0, g_1, ..., g_{p-1}, res)
  then
    return  res;
  if f=⊕_p
  then
    res=new_node(lab=⊕_p,child_0 = g_0, ..., child_{p-1} = g_{p-1});
  else
    begin
      for j = 0 to (p - 1) do
        h|_{x=j} =CASE-⊕(f|_{x=j}, g_0|_{x=j}, ..., 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},...,child_{p-1} = h_{x=p-1});
      insert_in_computed_table(f, g_0, g_1, ..., g_{p-1}, res);
    end;
  find_or_add_in_unique_table(res);
  return res;
end.
```

Figure 4: CASE-$\oplus$ algorithm for Mod-p-DDs 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 $g_0, g_1, \ldots, g_{p-1}$ 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 $f|_{x=j}, g_0|_{x=j}, g_1|_{x=j}, \ldots, g_{p-1}|_{x=j}$ as its arguments. These are composed by using Boole/Shannon decomposition as $(x, h_0|_{x=0}, h_1|_{x=1}, \ldots,\ h_{p-1}|_{x=p-1})$.

The adoption 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, if this node does not already exist in the Mod-p-DD (see Figure 5).

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. Therefore, we have given 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, which necessitates the

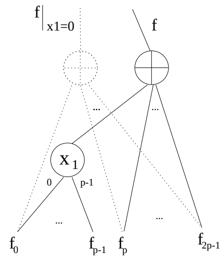


Figure 5: Cofactor creation $f|_{x_1=0}$ in Mod-p-DD.

development of a complete new package, or (2) implementing a Mod-p-DD by performing an arbitrary encoding of the multiple-valued function $M^n \rightarrow M$ represented by Mod-p-DD into the Boolean function, $B^{\lceil \log p \rceil \cdot n} \rightarrow B^{\lceil \log p \rceil}$. Thus, a multi-valued variable x_i can be encoded as $\lceil \log p \rceil$ binary Boolean variables $x_{i_1}, \ldots, x_{i_{\lceil \log p \rceil}}$. 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 existing Parity-OBDD package [18].

References

[1] A. Nozoe et al., A 256-Mb multilevel flash memory with 2 MB/s program rate for mass storage applications, *Proc. of 1999 IEEE Int. Solid-State Circuits Conference (ISSCC'99)*, (1999), 110-111.

[2] T. Okuda, T. Murotani, A four-level storage 4-Gb DRAM *IEEE Journal of Solid-State Circuits* **32**, 11, (1997), 1743 - 1747.

[3] The VIS Group, VIS: A system for verification and synthesis, *Proc. 8th Int. Conf. on Computer Aided Verification*, Springer Lecture Notes in Computer Science, **1102**, Edited by R. Alur and T. Henzinger, New Brunswick, NJ, (1996), 428-432.

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

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

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

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

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

[9] 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.

[10] 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.

[11] 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).

[12] 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.

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

[14] 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.

[15] 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.

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

[17] 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.

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