Efficient Minimization of Multiple-valued Decision Diagrams for Incompletely Specified Functions

Denis V. Popel
Computer Science Department
Baker University, Baldwin City, KS 66006
denis.popel@bakeru.edu

Rolf Drechsler
Institute of Computer Science
University of Bremen, 28359 Bremen, Germany
drechsle@informatik.uni-bremen.de

Abstract

This paper addresses the problem of finding a small size Multiple-Valued Decision Diagram (MDD) representation of an incompletely specified multiple-valued function. Optimal MDD representation improves performance and flexibility of many applications in logic design and multiple-valued circuit synthesis.

We introduce an algorithm which incorporates a new operation on incompletely specified MDDs, called fusion. *The diagram is optimized by dynamic variable ordering, graph compaction and minimization. During the optimization the structure of the underlying MDD is modified in a way that only specified values are represented while don't cares are ignored. The results on multiple-valued as well as binary benchmarks with don't cares are given to demonstrate the efficiency and robustness of the algorithm.*

1 Introduction

Several algorithms have been presented to represent and manipulate multiple-valued incompletely specified functions[1]. One major problem in the representation of incompletely specified functions is the size explosion through redefining don't cares. Recently, there is a renewed interest in solving this problem based on symbolic techniques, since existing techniques often do not guarantee sufficient quality. As one approach decision diagram have been successfully applied to many logic design and circuit synthesis domains [1, 5].

Different aspects of decision diagram representation for incompletely specified multiple-valued data have been extensively studied in the following areas: decomposition of multiple-valued functions using MDDs [7]; minimization of multiple-valued logic networks [6]; and synthesis of *Finite State Machines* (FSMs) [8, 12]. Further application domains are verification of large FSMs with many unreachable states, synthesis of *Field Programmable Gate Arrays* (FPGAs) derived from decision diagram representations, and data mining and decision making. Though decision diagrams have shown to be a practical tool for symbolic verification and logic function manipulation, they are not always efficient to deal with incompletely specified functions. As an alternative, new graph structures and novel principles might be exploited and validated for the representation and manipulation of incompletely specified functions.

There are two counteractive principles of dealing with don't cares of an incompletely specified function. The first (the traditional one) is based on redefining unspecified values of the function. This principle has been extensively exploited for several years [5, 10]. Another principle utilizes the idea of "don't care about don't cares" [14] which stands for excluding unspecified values from further analysis. So far there are no direct diagram related techniques to manipulate incompletely specified functions that follow this principle. Especially in the multiple-valued domain there is a large potential for optimization and construction of decision diagrams for incompletely specified functions, since the serach space is often much larger. E.g., many practical problems in data mining require up to 200 input variables with an enormous number of don't cares. Problems with an unusual high percentage of don't cares occur also in circuit design, e.g. when using VHDL compilers [1].

This paper proposes an approach to employ the idea of "don't care about don't cares" for MDD minimization. For this, a problem specific structural "extension" to MDDs, called *incompletely specified MDD* (*is*MDD), is introduced. For *is*MDD optimization algorithms which incorporates the new operation *fusion*, dynamic variable ordering, graph compaction and minimization are described. The experimental results demonstrate the efficiency and

[1]For an in-depth review see the special session of the Post-Binary Ultra Large Scale Integration (ULSI) Workshop, 1997

robustness of the algorithm especially for *weakly specified* functions, i.e. with a high percentage of don't cares.

2 Preliminaries and Definitions

2.1 Incompletely Speci£ed Functions

An *incompletely specified multiple-valued function* f, also known as a function with don't cares, is a relation where certain combinations of its variables cannot occur. Thus, the truth table of the function f does not generate output values for every possible combination of input values. In the following, let us consider the incompletely specified m-valued function f: $\mathbf{A}^n \rightarrow \mathbf{B}$ over the variable set $X = \{x_1, \cdots, x_n\}$, where $\mathbf{A}$=$\{0,\ldots,r-1\}$ and $\mathbf{B}$=$\{0,\ldots,m-1\}$, and n is the number of r-valued variables. More formally, the function f can be represented by sets: $\mathcal{C}_i = \{\epsilon \in \mathbf{A}^n | f(\epsilon) = i,\ i = 0, \ldots, m-1\}$; and $\mathcal{DC} = \{\epsilon \in \mathbf{A}^n | f(\epsilon) = d\}$, where d is an undefined value, and ϵ is a cube. $k = \sum_{i=0}^{m-1} |\mathcal{C}_i|$ is the number of cubes ϵ of the function f. It is obvious that for a completely specified function f the set $\mathcal{DC} = \oslash$. Following the idea of "don't care about don't cares" only the sets $\mathcal{C}$ are considered to specify the function f and manipulate with cubes. Note that the approach is not limited to single-output functions, it can also be applied to functions with several outputs.

2.2 Multiple-Valued Decision Diagrams

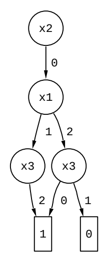
Graph-based data structures have become the advanced tools in logic design for representation and manipulation of discrete data because of their simplicity, canonical nature, and effective algorithms.

As well-known, each Boolean function f: $\{0,1\}^n \rightarrow \{0,1\}$ can be represented by a *Binary Decision Diagram* (BDD) [3], i.e. a directed acyclic graph with node (vertex) set V where: (i) each *non-terminal* node u is labeled by a variable x and assigned as a decision variable with two successors (children) $u.left$ and $u.right$; (ii) a *terminal* node is labeled with the leaf value and has no successors. Obviously, BDDs can be extended to represent functions f:$\{0,1\}^n \rightarrow \{0,\ldots,m-1\}$ and the resulting graphs are denoted as *Multi-Terminal BDDs* (MTBDDs). The operations on MTBDDs can be carried out as efficiently as in the case of two terminals [4].

It is straightforward to extend MTBDDs to *Multiple-Valued Decision Diagrams* (MDDs) [13] representing functions f: $\{0,\ldots,r-1\}^n \rightarrow \{0,\ldots,m-1\}$. For this, each internal node has r outgoing edges. In [13], it has been shown that the efficient operations known for BDDs can also be carried out on MDDs using a *case*-operator instead of the *ite*-operator [2]. A MDD is called *ordered* if the variables X appear in the same order $x_{j_1} \prec x_{j_2} \prec \ldots \prec x_{j_n}$ on each path from the root to a terminal node. In the following, only ordered MDDs are considered. It is known that the order of variables can be changed to reduce the MDD size. We employ this fact and integrate dynamic variable ordering techniques into the MDD minimization algorithm for incompletely specified multiple-valued functions. A MDD can be compacted by applying certain rules to eliminate redundant nodes. We assume that the common principles of decision diagram manipulations are known [3].

3 Incompletely Specified MDDs

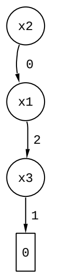
The basic problem of a decision diagram representation of an incompletely specified function explored by many researchers [5, 12] is how to assign the set $\mathcal{DC}$ so that the size of the diagram representing the corresponding completely specified function is minimized. This problem, known to be NP-complete [10], has been addressed through a variety of methods both heuristic [12] and exact [8]. In addition, the variable reordering technique utilizing don't cares has been presented in [11]. While the classical approaches assume that the specified parts and the don't care part are both given as functions, in the following we make use of don't cares by 'ignoring' them during the MDD construction. For this, MDDs are 'extended' to support the principle of "don't care about don't cares."



Definition 1 *An incompletely specified MDD (isMDD) Ξ is a directed acyclic graph with a node set in which each non-terminal node u has at least one successor.*

Shown on the left is an example of an *is*MDD. The properties of *is*MDDs are defined similarly to those for traditional MDDs.

Note that in contrast to MDD nodes, in *is*MDD nodes may have fewer outgoing edges. The compactness of *is*MDDs is guaranteed by two rules: (i) merging, which shares equal functions, and (ii) deletion, which deletes a node where all r children are equal. These rules are formulated analogously to the case of completely specified functions except the case of nodes with fewer successors than the radix of variables. We introduce a post-processing operation called *minimization* which eliminates nodes with a single outgoing edge. It can be considered as a transition from *is*MDD into traditional MDD.



Definition 2 *A chain ξ is a linear incompletely specified decision diagram composed of (i) non-terminal nodes associated with each variable x of the function f, and (ii) terminal nodes with the value $f(\epsilon)$. It represents a cube ϵ as a conjunction of a set of variable values: $\epsilon \Rightarrow \xi$. Note that don't cares are excluded from consideration.*

Depicted on the left is a chain for the cube $\{x_2^0 x_1^2 x_3^1\}$ from the set $\mathcal{C}_0$.
The variable ordering in the chain ξ is adjusted to the variable ordering in the existing diagram according to the output of dynamic variable reordering. To add the chain ξ into the current *is*MDD, graph structures are merged through a *fusion* operation defined below.

Operations on Chains and *is*MDDs

Definition 3 *A fusion operation $\coprod$ unites two isMDDs Ξ_1 and Ξ_2 with an identical variable ordering: $\Xi = \Xi_1 \coprod \Xi_2$.*

In the following, we consider the design of the *is*MDD Ξ for a given incompletely specified multiple-valued function f as a sequence of fusion operations on k chains ξ: $\Xi = \coprod_k \xi_k$. This process can be described iteratively as a sequence of snapshots: $\Xi_{t+1} = \Xi_t \coprod \xi_{t+1}$, where $t = 1, \ldots, k$ and $\Xi_1 = \xi_1$.

Lemma 1 *The computational complexity of the fusion operation $\coprod$ on the current isMDD Ξ and the chain ξ is in $O(n)$.*
Proof: According to the definition of the fusion operation, the order of variables in Ξ and ξ has to be the same. Let us assume that the order of variables in the chain ξ is different, it can be shown that the computational complexity of the chain reordering can be carried out in $O(n)$. The fusion operation has a recursive depth-first nature, where the complexity is estimated as the number of levels n in the *is*MDD Ξ. Both reordering of the chain ξ and the depth-first attaching of nodes into the *is*MDD Ξ take time $O(n)$.

Theorem 1 *The computational complexity of the design of the isMDD Ξ is $O(k \cdot n)$.*
Proof: As stated before, the design of the *is*MDD Ξ can be described as a sequence of fusion operations on k chains ξ: $\Xi = \coprod_k \xi_k$. Each fusion operation has the computational complexity $O(n)$, thus k operations take time $O(k \cdot n)$.

Lemma 2 *The design of the isMDD Ξ has space complexity in $O(k \cdot n)$.*
Proof: Turning to the space requirement, we state that for the representation of an incompletely specified function f as the set of k cubes, the number of nodes for the chain ξ is equal to n, and the space complexity for the entire set of chains is bounded by $k \cdot n$.

Theorem 2 *The fusion operation $\coprod$ reduces the space complexity which can be finally approximated by*

$$k \cdot (n - \log k + 1) - 1. \tag{1}$$

Proof: The formula is obtained by Stirling's approximation. The fusion operation between two chains reduces the space complexity by minimum $\log 2$ nodes. Attaching the third chain can result in minimum $\log 3$ reductions, and so on. k fusion operations reduce $\log 1 + \log 2 + \ldots + \log k = \log k!$ nodes. According to the Stirling's approximation, $\log k! \approx k \log k - k + 1$. Thus, the final complexity of the *is*MDD is $k \cdot n - k \log k + k - 1$.
The formal proof is done by induction. (1) If function f is given on a single cube ($k = 1$), the *is*MDD would be comprised of n nodes which is equal to the output of Equation (1) for $k = 1$. (2) If the function f is specified on $k = 2$ cubes, the worst case of the fusion operation results in $2n - 1$ nodes (the output of Equation (1) for $k = 2$). (3) Let us assume that the approximation presented above is correct for $k-1$ cubes. The number of nodes is equal to $(k-1) \cdot (n - \log(k+1) + 1) - 1$. Attaching a new chain with n nodes simplifies the *is*MDD by $\log k$ nodes: $(k-1) \cdot (n - \log(k+1) + 1) - 1 + (n - \log k) = k \cdot (n - \log(k-1) + 1) + \log(1 - 1/k) - 2 \approx k \cdot (n - \log k + 1) - 1$.

Observation 1 *The latter formula leads to the conclusion that the usability of this method is especially applicable to weakly specified functions, where k is significantly less than r^n: $k \ll r^n$.*

Observation 2 *For the function f where $k = r^n$ (completely specified function), the space complexity of the design of the isMDD Ξ is approximated by r^n in the worst case. Obviously, the minimization stage does not contribute any reduction in number of nodes.*

Many heuristics have been proposed for finding a good variable ordering. In this approach, we have chosen to use sifting, because of its dynamic nature. The basic idea of the sifting algorithm is selecting the best position for one variable assuming that the relative order of all others remains the same. This process is repeated for all variables, starting with variables situated in the level with the largest number of nodes. Dynamic variable ordering is integrated into the algorithm outlined below.

Example 1 *Let us consider the following incompletely specified 3-valued function $f = f(x_1, x_2, x_3)$ given on five combinations of variable values $\mathcal{C}_0 = \{x_1^2 x_2^0 x_3^1, x_1^2 x_2^1 x_3^2\}$, $\mathcal{C}_1 = \{x_1^1 x_2^0 x_3^2, x_1^2 x_2^0 x_3^0\}$, and $\mathcal{C}_2 = \{x_1^2 x_2^0 x_3^2\}$. The process of constructing an isMDD and its minimization is illustrated in Figure 1.*

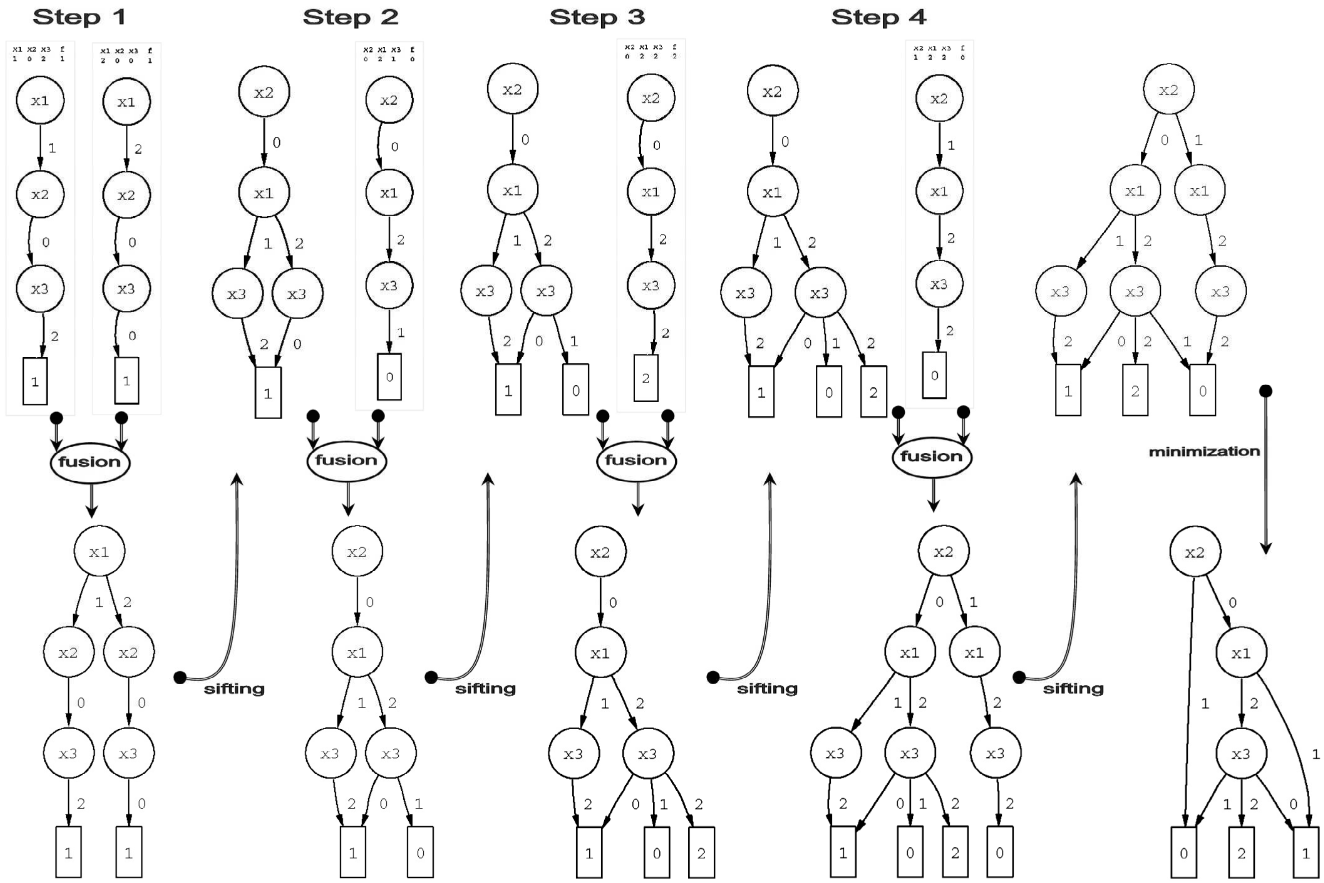


Figure 1. MDD minimization for the function f from Example 1

4 Algorithm and Experimental Results

A sketch of the MDD minimization algorithm for the function f given on the sets $\mathcal{C}$ is shown below:

Step 1. `Initialize the variable ordering lexicographically:` $x_1 \prec x_2 \prec \ldots \prec x_n$ `(`x_1 `is the topmost variable), and the` *is*`MDD` $\Xi = \oslash$.

Step 2. `Consider a cube` ϵ `from the given set` $\mathcal{C}$`:` $\epsilon \in \mathcal{C}$`. Build a corresponding chain` ξ `applying current variable ordering:` $\epsilon \Rightarrow \xi$.

Step 3. `Merge the chain` ξ `with the existing` *is*`MDD` Ξ `applying the fusion operation:` $\Xi = \Xi \coprod \xi$.

Step 4. `Reorder the obtained` *is*`MDD` Ξ `applying sifting.`

Step 5. `Perform the compaction of the` *is*`MDD` Ξ `to merge nodes which share equal functions, and delete parent nodes with equal children.`

Step 6. `If there are other cubes` $\mathcal{C} \neq \oslash$`, go to Step 2. Otherwise, terminate the algorithm and do the minimization of the` *is*`MDD` Ξ.

Step 7. `(Postprocessing) Perform the minimization of the` *is*`MDD` Ξ `to eliminate nodes with don't cares.`

The algorithm is coded in Java and the experimental results are obtained on a Pentium III workstation with 256Mb of memory. The Java program itself is designed specifically to process multiple-valued data (no other packages and simulation software have been used). It incorporates the subroutines for chain creation, graph compaction and minimization. As input the program accepts binary or multiple-valued and incompletely specified or completely specified functions, respectively. Comparison to previous approaches turned out to be difficult to perform, since many other authors performed their experiments on completely specified functions selecting don't cares randomly. Here, our experiments have been carried out on original incompletely specified functions taken from two different benchmark sets.

Table 1 shows the results of the experiments on the *Wright Lab* benchmark set with 70% and 90%

don't cares[2]. The table is divided into two parts: the left part for the benchmarks with 90% don't cares and the right one for the benchmarks with 70% don't cares. Each part is split into two parts: the part with the original binary benchmarks and the part with 4-valued benchmarks generated by pairing inputs of the original benchmarks. The columns labeled [F] list the number of *is*MDD nodes for the algorithm with fusion operation only (no MDD compaction or minimization is performed). The columns labeled [FC] give the number of nodes with *is*MDD compaction (the postprocessing stage of *is*MDD minimization is omitted). On average, the compaction step eliminates 46% of nodes. The *is*MDD minimization step combined with the compaction step (columns [FCM]) results in small representations. Encoding (pairing) inputs contributes on average 36% reduction counted in the number of nodes. The reduction observed suggests that the algorithm based on "don't care about don't cares" principle is viable to build an optimal *is*MDD for incompletely specified data, particularly for weakly specified functions. Indeed, the method outperforms approaches applying the principle of redefining don't cares, e.g. assigning 0's (the results are reported in columns [*]).

Table 2 shows some of the results on the *Machine Learning* benchmark set[3]. In addition to previously used labels [F], [FC], [FCM] and [*], r denotes the maximum radix of attributes, and m denotes the radix of a single-output multiple-valued function. Some continuous valued benchmarks have been pre-processed by the discretization program based on information density measures as reported in [9]. The table shows a significant reduction in the number of nodes while applying the minimization operation. Our approach outperforms the algorithm described in [7] which utilizes binary-encoded MDDs (BEMDDs and the corresponding column [BEMDDs]).

Table 2. Results on the UCI machine learning benchmark set (multiple-valued and continuous benchmarks)

Dataset	r	m	F	FC	FCM	*	BEMDDs
auto-mpg♯	14	5	864	301	121	270	-
balance	5	3	156	29	29	29	85
breastc	10	2	2592	994	39	800	3490
flare1	7	5	598	226	58	226	298
flare2	7	7	503	239	103	239	-
glass♯	7	7	494	224	62	125	-
hayes	4	3	72	27	14	24	-
ionosphere♯	7	2	2241	1338	89	827	-
iris♯	7	3	57	40	8	27	-
lensesmv	3	3	19	6	6	6	-
monks1te	4	2	301	5	5	5	-
monks1tr	4	2	176	52	12	32	118
monks2te	4	2	301	9	5	9	-
monks2tr	4	2	203	74	41	42	138
monks3te	4	2	329	3	3	3	-
monks3tr	4	2	191	58	19	38	119
post-operative♯	3	3	202	100	38	74	-
shuttle	4	2	581	10	7	7	-
yeast♯	7	10	67	61	30	31	-
zoo	9	7	343	105	14	61	216

♯Benchmarks with continuous attributes.

5 Conclusion

This paper presents an new approach to minimization of MDDs for incompletely specified multiple-valued functions. The main components of this approach are: (i) the novel graph structures, viz. *is*MDDs to represent incompletely specified functions and chains to represent cubes, (ii) the fusion operation to build *is*MDDs, and (iii) the new algorithm which incorporates the fusion operation on *is*MDDs. We studied complexity related aspects from a theoretical point of view. In particular, supplying weakly specified functions due to the computational complexity of the algorithm is recommended. Experimental results showed that the combination of dynamic variable ordering, graph compaction and minimization results in small sized MDDs.

[2] http://www.ece.pdx.edu/∼polo

[3] http://www.ics.uci.edu/mlearn/MLRepository.html

Acknowledgements

The authors would like to thank Christine Wacker, Baker student, for the data preparation.

References

[1] R. Bergamaschi, D. Brand, L. Stok, M. Berkelaar, and S. Prakash. Efficient use of large don't cares in high-level and logic synthesis. In *Proc. IEEE/ACM Int. Conference on CAD*, pages 272–278, 1995.

[2] K. Brace, R. Rudell, and R. Bryant. Efficient implementation of a BDD package. In *Proc. IEEE/ACM Int. Design Automation Conference*, pages 40–45, 1990.

[3] R. Bryant. Graph - based algorithm for Boolean function manipulation. *IEEE Trans. on Computers*, C-35(8):667–691, 1986.

[4] E. Clarke, M. Fujita, P. McGeer, K. McMillan, J. Yang, and X. Zhao. Multi terminal binary decision diagrams: an efficient data structure for matrix

Table 1. Results on the Wright Lab benchmark set (binary and multiple-valued benchmarks)

	90% don't cares								70% don't cares							
	binary				4-valued				binary				4-valued			
Dataset	F	FC	FCM	*	F	FC	FCM	*	F	FC	FCM	*	F	FC	FCM	*
add0	92	53	9	27	41	21	6	16	173	81	19	36	66	36	14	21
add2	91	56	14	27	41	22	10	14	173	81	19	36	66	36	14	21
add4	92	54	8	30	40	20	8	15	171	77	9	38	68	31	12	24
and_or_chain8	92	53	4	15	41	21	2	10	173	78	6	22	66	31	3	14
ch15f0	95	55	14	23	40	24	9	14	170	81	27	37	66	37	21	20
ch177f0	97	49	10	26	40	24	6	15	172	83	6	40	67	33	13	21
ch30f0	92	54	7	17	41	21	4	8	172	79	18	30	66	32	11	14
check_fail	93	53	14	23	44	25	10	16	174	87	30	44	70	41	23	25
contains_4_ones	88	53	14	21	40	23	7	10	173	88	36	45	66	40	27	23
greater_than	92	55	9	21	41	22	5	13	170	80	16	41	67	35	14	23
interval1	84	54	12	20	40	23	6	11	172	81	22	31	68	31	13	15
interval2	92	55	14	29	40	24	8	14	172	86	24	39	67	36	17	23
kdd1	92	52	3	24	41	21	2	13	173	78	3	37	66	32	2	21
kdd3	92	51	5	14	41	19	3	5	173	76	8	27	66	32	4	13
kdd6	92	50	4	33	41	19	2	18	174	73	8	49	68	29	7	27
majority_gate	86	54	10	23	40	24	8	16	170	82	21	28	66	36	15	17
modulus2	84	54	7	16	36	23	4	10	172	77	17	21	66	31	11	13
monkish1	92	54	5	23	41	21	3	13	173	82	8	40	67	32	7	22
monkish3	93	54	8	25	40	22	5	13	176	81	7	34	67	32	8	19
mux8	90	56	9	26	40	24	6	17	170	82	12	36	67	31	9	20
nnr1	95	54	10	22	40	22	9	14	172	84	31	40	67	36	21	25
nnr2	92	53	7	29	41	20	4	16	173	77	13	41	66	33	8	26
nnr3	83	54	7	25	40	21	4	12	169	82	26	37	67	37	18	21
or_and_chain8	92	52	3	25	41	19	2	15	173	76	5	39	66	30	3	21
pal_dbl_output	87	56	15	26	40	25	9	15	174	83	32	48	67	39	25	28
pal_output	93	54	13	30	40	23	8	13	173	90	36	47	66	39	26	24
parity	93	59	15	30	40	23	9	15	170	82	42	54	68	38	28	26
primes8	93	49	7	10	40	20	5	7	173	86	27	31	66	37	20	17
remainder2	101	49	13	24	40	24	8	14	171	82	21	40	67	36	15	20
rnd_m25	90	52	6	8	41	19	4	9	173	75	17	21	66	30	10	10
rnd_m50	83	52	10	16	40	24	5	9	172	83	25	35	66	32	16	18
rnd1	93	56	14	27	40	22	10	14	170	86	36	44	66	39	24	25
rnd2	89	52	15	26	40	23	9	14	171	82	38	46	67	38	27	26
rnd3	91	56	12	29	41	22	8	16	170	91	33	52	67	37	24	26
rndvv36	86	54	13	28	40	25	9	19	169	82	30	45	67	38	22	26
substr1	90	57	12	24	41	23	8	13	171	82	24	42	67	36	16	24
substr2	93	54	11	18	40	24	6	11	174	78	21	29	66	34	16	16
subtraction1	83	53	12	26	41	20	9	15	170	82	32	49	66	38	23	26
subtraction3	92	54	8	30	40	20	8	15	171	77	9	38	68	31	12	24

representation. In *Proc. Int. Workshop on Logic Synthesis*, pages P6a:1–15, 1993.

[5] Y. Hong, P. Beerel, J. Burch, and K. McMillan. Safe BDD minimization using don't cares. In *Proc. IEEE/ACM Int. Design Automation Conference*, pages 208–213, 1997.

[6] Y. Jiang and R. Brayton. Don't cares and multi-valued logic network minimization. In *Proc. IEEE/ACM Int. Conference on CAD*, pages 520–525, 2000.

[7] A. Mishchenko, C. Files, M. Perkowski, B. Steinbach, and C. Dorotska. Implicit algorithms for multi-valued input support minimization. In *Proc. Int. Workshop on Boolean Problems*, pages 9–20, 2000.

[8] A. Oliveira and S. Edwards. Limits of exact algorithms for inference of minimum size finite state machines. In *Algorithmic Learning Theory Workshop*, volume 1160, pages 59–66, 1996.

[9] D. Popel. From continuous to multiple-valued data. In *Proc. IEEE Int. Symposium on Multiple-Valued Logic*, 2003.

[10] M. Sauerhoff and I. Wegener. On the complexity of minimizing the OBDD size for incompletely specified functions. *In IEEE Trans. on CAD of Integrated Circuits and Systems*, 15:1435–1437, 1996.

[11] C. Scholl, D. Möller, P. Molitor, and R. Drechsler. BDD minimization using symmetries. *In IEEE Trans. on CAD of Integrated Circuits and Systems*, 18(2):81–100, 1999.

[12] T. Shiple, R. Hojati, A. Sangiovanni-Vincentelli, and R. Brayton. Heuristic minimization of BDDs using don't cares. In *Proc. IEEE/ACM Int. Design Automation Conference*, pages 225–231, 1994.

[13] A. Srinivasan, T. Kam, S. Malik, and R. Brayton. Algorithms for discrete function manipulation. In *Proc. IEEE/ACM Int. Conference on CAD*, pages 92–95, 1990.

[14] A. Zakrevskij. Optimizing polynomial implementation of incompletely specified Boolean functions. In *Proc. Workshop on Application of the Reed-Muller Expansions in Circuit Design*, pages 250–256, 1995.