Index-Resilient Zero-Suppressed BDDs: Definition and Operations

ANNA BERNASCONI, Università di Pisa
VALENTINA CIRIANI, Università degli Studi di Milano

Zero-Suppressed Binary Decision Diagrams (ZDDs) are widely used data structures for representing and handling combination sets and Boolean functions. In particular, ZDDs are commonly used in CAD for the synthesis and verification of integrated circuits. The purpose of this paper is to design an error resilient version of this data structure, i.e., a self-repairing ZDD. More precisely, we design a new ZDD canonical form, called index-resilient reduced ZDD, such that a faulty index can be reconstruct in time $O(k)$, where k is the number of nodes with a corrupted index. Moreover, we propose new versions of the standard algorithms for ZDD manipulation and construction, which are error resilient during their execution and produce an index-resilient ZDD as output. The experimental results validate the proposed approach.

CCS Concepts:•**Hardware** → **Error detection and error correction;** *Fault tolerance;*

Additional Key Words and Phrases: Data structures for logic synthesis, Error Resilient Data Structures, Binary Decision Diagrams, Zero-Suppressed Binary Decision Diagrams

1. INTRODUCTION

Due to the growing density of components on chips, even circuits and memories that are properly designed and produced may suffer from errors. For example, background radiation can induce single bit errors in the circuits that cause incorrect output behavior [Taylor 1990]; or small imperfections in the chip manufacturing can remain undetected in the production test.

Fast, large, and cheap memories in today's computer platforms are characterized by non-negligible error rates, which cannot be underestimated as the memory size becomes larger [Jacob et al. 2008]. Since the correctness of the underlying algorithms may be jeopardized by even very few memory faults, computing in the presence of memory errors is a fundamental task in many applications. Therefore, the resiliency of a data structure to memory faults has become a very important issue.

The scientific community has studied the problem in two different frameworks: *(i)* fault tolerant hardware design and *(ii)* development of error resilient algorithms and data structures. While fault tolerant hardware has been widely studied even in the past, the design of algorithms and data structures resilient to memory faults, i.e., that are able to perform the tasks they were designed for even in the presence of unreliable or corrupted information, has become much more attractive only recently (for a survey on the subject refer to [Finocchi et al. 2007; 2009; Italiano 2010]).

This work was supported in part by the Italian Ministry of Education, University, and Research (MIUR) under PRIN 2012C4E3KT national research project "AMANDA" Algorithmics for MAssive and Networked DAta.

A preliminary version of this paper has been presented in *TCS* (2014) [Bernasconi and Ciriani 2014].

Author's addresses: A. Bernasconi, Department of Computer Science, Università di Pisa, Italy, E-mail: anna.bernasconi@unipi.it; V. Ciriani, Department of Computer Science, Università degli Studi di Milano, Italy, E-mail: valentina.ciriani@unimi.it.

DOI: http://dx.doi.org/10.1145/0000000.0000000

Several error models have been proposed for designing resilient data structures [Taylor 1990]. A fault model in which any error is detectable via an error message when the program tries to reach the faulty object is proposed in [Aumann and Bender 1996]. The authors assume that an error denies access to an entire node of the structure. A model with higher granularity, called *faulty-RAM*, is presented in [Finocchi et al. 2005; Finocchi and Italiano 2008; Italiano 2010]. In faulty-RAM an adversary can corrupt any memory word and it is impossible to determine a priori if a memory area is corrupted or not. Such a scenario is realistic since an error can be induced by an external source, perhaps temporary, which can change any memory location that can not be discovered a priori. Moreover, faulty-RAM model has an extreme granularity: any memory location (from a single bit, the single data, or an entire structure) can be affected by a fault. Another interesting error model is the *single-component model* described in [Taylor 1990], which focuses on single attributes of an item at a time and assumes that each error affects one component of one node of the storage structure, e.g., a pointer, a count, an identifier field.

The purpose of this paper is to discuss the error resilience of a data structure called *Zero-Suppressed Binary Decision Diagrams* (ZDDs) [Minato 1993; Mishchenko 2001] and design an error-resilient version of it. ZDDs are a particular type of Ordered Binary Decision Diagrams (OBDDs), a fundamental family of data structures for Boolean function representation and manipulation [Bryant 1986]. OBDDs are DAG representations of Boolean functions, where each internal node N is labeled by a Boolean variable x_i and has exactly two outgoing edges: 0-edge and 1-edge. Terminal nodes (leaves) are labeled 0 or 1.

OBDDs can be constructed by applying some reduction rules to a Binary Decision Tree, and depending on the set of reduction rules, different representations can be derived. For example, Figures 1(b), 1(c), and 1(d) are different decision diagrams, derived by the decision tree in Figure 1(a), representing the same Boolean function. In particular, Reduced OBDDs (ROBDDs) [Bryant 1986] are typically used for the representation of general Boolean functions, while *Zero-Suppressed* BDDs (ZDDs) are used for representing family of subsets of combination sets [Minato 1993; Mishchenko 2001; Knuth 2009]. Indeed, ZDDs can be used to describe and manipulate solutions to combinatorial problems as, in this framework, they are much more compact and therefore more efficient than the traditional ROBDDs. For instance, the family of subsets $\{\{x_1, x_2\}, \{x_3, x_4\}, \{x_1\}\}$ of the set $\{x_1, x_2, \ldots, x_{10}\}$ needs a ROBDD representation with 10 variables, while the ZDD representation uses only the four variables included in the subsets.

Even if ZDDs were originally introduced and studied for circuit design and formal verification, their areas of application have widened and today ZDDs have developed into a fundamental research tool for combinatorial algorithms, including representation and manipulation of combination sets in different research fields as data mining [Minato 1993; 2010; 2013], bioinformatics [Minato and Kimihito 2007; Requeno and Colom 2012; Yoon et al. 2005], symbolic logic, probability theory, and string matching for computer virus detection and text retrieval. In a 2011 talk, Donald Knuth referred to ZDD as *the most beautiful construct in computer science*[1]. We refer the reader to [Sasao and Butler 2014] for a recent survey on Zero-Suppressed Decision Diagrams and their applications.

Due to the growing interest in this particular data structure and to its many important applications, in this paper we focus on the error resilience of ZDDs, mirroring what has already been done for standard ROBDDs, for which security aspects of implementation techniques have been discussed in [Drechsler 1998], and an error re-

[1]"All Questions Answered" by Donald Knuth, Google Tech Talk, March 24, 2011.

silient version has been proposed in [Bernasconi et al. 2013; 2015]. We exploit the single-component error model and we assume perfect error detection capabilities: errors are immediately reported when the program tries to use the fault component of a node. Thus, our analysis only deals with the problem of error correction; in fact, as in [Bernasconi et al. 2013; 2015], our main goal is to study the capability of ZDDs to restore corrupted data, not to detect them. However, it is worth mentioning that the peculiar structure of ZDDs, and in particular of the new model of ZDDs that we design and analyze in this paper, could be exploited for error detection too, as discussed later in the paper.

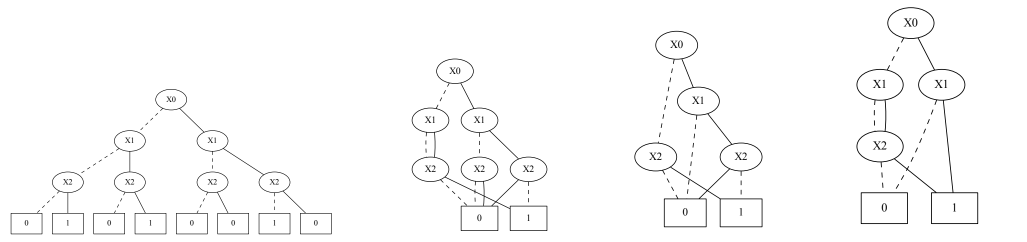
We consider, as component of a node N in a ZDD, the index i of the variable x_i associated to the node. In particular, we show that similarly to OBDDs, ZDDs can be made resilient to errors on indexes, and we design a new canonical ZDD form, called *index-resilient reduced ZDD*, where a faulty index can be reconstruct in time $O(k)$, where k is the number of nodes with a corrupted index. We then propose an index-resilient version of the standard algorithms for ZDD manipulation and construction, such as subset, union, intersection, and set difference. The main characteristic of these new versions is that they are error resilient during their execution and produce an index-resilient ZDD as output.

The experimental results show that the number of nodes in a index-resilient reduced ZDD is not too much greater that the number of nodes of a standard reduced ZDD; indeed, standard reduced ZDDs are only 4% more compact than index-resilient reduced ZDDs.

Besides indexes, it is important to correct errors in pointers to the children of a node. Pointer based data structures are not, in general, error resilient, since the loss of a pointer can imply the loss of an important part of the data structure. Different strategies could be adopted to handle edge errors. In particular, we can use error resilient linked data structures [Aumann and Bender 1996] or the ad hoc solution already proposed for OBDDs [Bernasconi et al. 2015]. As the solution studied for OBDDs can be immediately applied to correct pointer errors in ZDDs, without any substantial modification, in this paper we only focus on index errors.

We finally observe that, even if the techniques applied to obtain index-resilient reduced ZDDs and OBDDs are similar, the construction, reduction and dynamical computation of ZDDs are not immediate generalizations of those defined in [Bernasconi et al. 2015] for OBDDs. Indeed, due to the different reduction rules applied to ZDDs, the methods discussed in [Bernasconi et al. 2015] for OBDDs cannot be directly applied to these diagrams. In particular, the concept of *removable chains of nodes* [Bernasconi et al. 2015] can be generalized to ZDDs only after a preprocessing phase during which a particular sequence of nodes, called *zr-chain*, has been identified and removed, if present, from the decision diagram (as explained in Section 4). Moreover, since the basic operations for the manipulation of ZDDs are different from those typically applied to OBDDs, their error resilient versions had to be designed from scratch.

The paper is an extended version of the conference paper [Bernasconi and Ciriani 2014], where only static construction of index-resilient ZDDs was considered, and is organized as follows. Preliminaries on OBDDs and ZDDs are described in Section 2. In Section 3 we discuss the error resilience of the standard ZDD structure, and in Section 4 we introduce and study index-resilient ZDDs. Section 5 discusses how index-resilient ZDDs can be dynamically computed through a sequence of set operations applied to other index-resilient ZDDs using index-resilient versions of the standard ZDD procedures. Experimental results for validating the proposed strategies are reported in Section 6. Section 7 concludes the paper.



(a) BDT (b) QR-BDD (m-rule) (c) ROBDD (m-rule and r-rule) (d) ZDD (m-rule and z-rule)

Fig. 1. Example of transformations of a BDT using the reduction rules.

2. ZERO-SUPPRESSED BDDS

2.1. Definitions and Reduction Rules

A *Binary Decision Tree* (BDT) on a set of Boolean variables $\{x_0, x_1, \ldots x_{n-1}\}$ is a rooted binary tree, where each non-terminal (internal) node N is labeled by a Boolean variable x_i and has exactly two outgoing edges: 0-edge and 1-edge. Terminal nodes (leaves) are labeled 0 or 1 (e.g., see Figure 1(a) where dashed, rep., solid, lines represent 0-edges, resp., 1-edges). Without loss of generality, we can assume that each node containing the variable x_i (with $0 \leq i \leq n-1$) lyes on the i-th level of the tree. Thus, the variable x_0 is the root of the BDT and the leaves are on level n (see for example the BDT in Figure 1(a)).

BDTs are typically used for representing completely specified Boolean functions (i.e., any function $f : \{0,1\}^n \rightarrow \{0,1\}$). The leaves represent the constants 0 and 1 and the root represents the entire Boolean function f. The value of f on the input $x_0, \ldots, x_{n-1}$ is found by following the path indicated in the BDT by the values of $x_0, \ldots, x_{n-1}$ on the edges: the value of $f(x_0, \ldots, x_{n-1})$ is the label of the reached leaf. For example, the BDT in Figure 1(a) represents the Boolean function $f : \{0,1\}^3 \rightarrow \{0,1\}$ such that $f(0,0,0) = 0, f(0,0,1) = 1, \ldots, f(1,1,1) = 0$.

In order to give a more compact description of Boolean functions, a BDT can be compressed in an acyclic graph (called BDD) that represents the same function. In particular, a *Binary Decision Diagram* (BDD) on a set of Boolean variables $X = \{x_0, x_1, \ldots x_{n-1}\}$ is a rooted, connected direct acyclic graph, where each non-terminal node N is labeled by a Boolean variable x_i, and has exactly two outgoing edges, 0-edge and 1-edge, pointing to two nodes called 0-child and 1-child of node N, respectively. Terminal nodes (leaves) are labeled 0 or 1. A 0-*parent* (resp., 1-*parent*) of a node N is a node M such that N is a 0-child (resp, 1-child) of M. For instance, the decision diagrams in Figures 1(b), 1(c), and 1(d) are examples of BDDs.

A BDD is *ordered* (OBDD) if there exists a total order $<$ over the set X of variables such that if an internal node is labeled by x_i, and its 0-child and 1-child have labels x_{i_0} and x_{i_1}, respectively, then $x_i < x_{i_0}$ and $x_i < x_{i_1}$. Hereafter, we will consider ordered BDDs only.

In order to obtain an OBDD starting from a BDT we can apply several *reduction rules*:

— **m-rule:** (or merge rule) if M and N are two distinct nodes that are roots of isomorphic subgraphs, then N is deleted, and all the incoming edges of N are redirected to M (N and M are called *mergeable*);

— **r-rule:** (or redundant rule) a node N that has both edges pointing to the same node M is deleted and all its incoming edges are redirected to M (N is called *redundant node* or *r-node*);
— **z-rule:** (or zero-suppress rule) a node N that has the 1-edge pointing to the constant leaf 0 is deleted and all its incoming edges are redirected to the subgraph pointed by the 0-edge (N is called *z-node*).

A *zr-node* is a redundant z-node, i.e., is a node with both edges pointing to the constant leaf 0.

There are different *reduced BDD forms* that derive from the use of one or two reduction rules:

— **QR-BDD:** (*Quasi-Reduced BDD*) [Liaw and Lin 1992] is the OBDD derived from a BDT repeatedly applying the m-rule until it is no longer applicable (see Figure 1(b));
— **ROBDD:** (*Reduced Ordered BDD*) [Akers 1978; Bryant 1986; Ebendt et al. 2005; Knuth 2009] is the OBDD derived from a BDT repeatedly applying the m-rule and r-rule until they are no longer applicable (see Figure 1(c));
— **ZDD:** (*Zero-suppressed BDD*) [Knuth 2009; Minato 1993] is the OBDD derived from a BDT repeatedly applying the m-rule and z-rule until they are no longer applicable (see Figure 1(d)).

QR-BDDs, ROBDDs and ZDDs are *canonical forms*. In particular, given a function f and a variable ordering $<$, there is exactly one QR-BDD, one OBDD, and one ZDD with variable ordering $<$ that represent f. Thus, once we have fixed the variable ordering, we can compute the QR-BDD, the ROBDD and the ZDD starting from a BDT repeatedly applying the corresponding reduction rules in any order. Moreover, it is possible to first build a QR-BDD (applying the m-rule) and then transform it in a ROBDD (resp., ZDD) using the r-rule (resp., z-rule) on it. In fact, starting from a QR-BDD, the r-rule and the z-rule cannot create new mergeable nodes (as shown in [Bernasconi et al. 2015] for the r-rule, and in Section 4 for the z-rule). The interpretation of a QR-BDD as a Boolean function is equivalent to the interpretation of a BDT since the m-rule simply merges isomorphic subgraphs resulting in an OBDD that has all the paths from the root to the leaves containing *all* the variables in X. For ROBDDs and ZDDs we have to give a correct interpretation of possibly missing nodes (in a path), which have been deleted using the r-rule or the z-rule. In particular a missing variable in a path of a ROBDD means that the variable can have any value (0 or 1). For example, in Figure 1(c), the path "x_0 0-edge x_2 1-edge 1", where x_1 is missing, represents two possible input values (i.e., $001, 011$) on which the function takes the value 1. On the other hand, the interpretation of a missing variable x_i in a path of a ZDD means that if $x_i = 1$ the function outputs 0, otherwise (i.e., if $x_i = 0$) the function outputs the value obtained following the path. For example, in Figure 1(d), the path "x_0 1-edge x_1 1-edge 1", where x_2 is missing, means that $f(1, 1, 1) = 0$ and $f(1, 1, 0) = 1$.

In this paper we study ZDDs and their operations. In particular as these diagrams are mainly used for set representation (through the characteristic vectors), the standard operations on ZDDs consist in set manipulation as shown in the following section.

2.2. Operations on ZDDs

As already observed, ZDDs can be constructed applying the m-rule and z-rule to binary decision trees. However, in practical applications, ZDDs are dynamically constructed applying some basic operations to simple graphs representing the two terminal nodes and the single variables, or directly to other ZDDs already available. Thus, we now briefly review the following basic operations for the manipulation of ZDDs, for a more comprehensive treatment, see [Minato 1993]:

— SUBSET(Z, ind, val), $val \in \{0, 1\}$: returns the ZDD of the subset of the set represented by Z, where the value of the variable with index ind is equal to val;
— CHANGE(Z, ind): returns the ZDD obtained from Z complementing the variable with index ind;
— UNION(R, S): returns the ZDD representing the union of the two sets corresponding to R and S;
— INTERSEC(R, S): returns the ZDD representing the intersection of the two sets R and S;
— DIFF(R, S): returns the ZDD representing the set difference of R and S.

Without loss of generality, let us assume that the chosen variable ordering is $x_0 < x_1 < \ldots < x_{n-1}$, so that the index of a variable in a node is the *level* of the node in the corresponding ZDD.

All operations can be implemented by recursive algorithms that make use of a procedure called GETNODE(ind, Z_0, Z_1) which creates a node for the variable with index ind, with the 0 and 1-edges pointing to Z_0 and Z_1, respectively, or just returns a pointer to such a node, if it already exists. This procedure relies on the use of the so-called *unique table*, i.e., a hash table used in most software implementations to maintain the ZDD reduced: the lookup on the unique table is indeed used to determine whether it is necessary or not to create a new node. GETNODE also checks whether a node is actually a z-node and can be deleted: if Z_1 is the terminal node 0, GETNODE directly applies the z-rule and returns Z_0:

```
GETNODE($ind, Z_0, Z_1$)
  **if** ($Z_1$ == terminal 0) **return** $Z_0$ /* application of the z-rule */
  $Z$ = search of $(ind, Z_0, Z_1)$ in the unique table
  **if** ($Z \neq$ null) **return** $Z$ /* application of the m-rule */
  $Z$ = new node $(ind, Z_0, Z_1)$
  **insert** $Z$ in the unique table
  **return** $Z$
```

The recursive implementations of the operations have a similar structure. The first two procedures, SUBSET and CHANGE, are called on a node Z of a ZDD and a variable of index ind, and depending on ind and on the index $Z.index$ of the variable associated to Z, three cases are considered: $Z.index > ind$, $Z.index = ind$, $Z.index < ind$. Observe that, because of the application of the z-rule, $Z.ind > ind$ means that a node with index ind and with the terminal 0 as 1-child has been deleted from the ZDD. This must be taken into account in the implementation of the two operations. For instance, the procedure SUBSET

```
SUBSET($Z, ind, val$)
  **if** ($Z.index > ind$)
      **if** ($val == 1$) **return** the terminal $0$
      **else return** $Z$
  **if** ($Z.index == ind$) **return** $Z_{val}$
  **if** ($Z.index < ind$)
      **return** GETNODE($Z.index$, SUBSET$(Z_0, ind, val)$, SUBSET$(Z_1, ind, val)$)
```

returns the terminal 0 if $Z.index > ind$ and $val = 1$, that is if we are computing the subset of Z where the variable of index ind is equal to 1, whereas it returns Z if $val = 0$, since Z is the 0-child of the missing z-node associated to the variable of index ind. For $Z.index = ind$, the subgraph corresponding to val is returned (i.e., the 0-child Z_0 of Z if $val = 0$, and the 1-child Z_1 otherwise), while for $Z.index < ind$ the algorithm is applied recursively on the 0 and 1-child of Z.

The procedure CHANGE returns a node with index ind, with the terminal 0 (i.e., the 1-child of the missing z-node of index ind) as 0-child and Z (the 0-child of the z-node) as 1-child for $Z.index > ind$, it exchanges the two children Z_0 and Z_1 of Z is $Z.index = ind$, and it proceeds recursively on Z_0 and Z_1 if $Z.index < ind$:

```
CHANGE($Z, ind$)
  if ($Z.index > ind$) return GETNODE($ind$, terminal 0, $Z$)
  if ($Z.index == ind$) return GETNODE($ind$, $Z_1$, $Z_0$)
  if ($Z.index < ind$) return GETNODE($Z.index$, CHANGE($Z_0, ind$), CHANGE($Z_1, ind$))
```

The last three procedures (UNION, INTERSEC and DIFF) are called on the roots R and S of two ZDDs, and depending on the indexes $R.index$ and $S.index$, three cases are considered:

— if $R.index = S.index$, a new node U with the same index is created (or returned if it already exists) and the algorithms are recursively executed on the 0-children of R and S, to generate the ZDD whose root becomes the 0-child of U, and on their 1-children to generate the 1-child of U;
— if $R.index < S.index$, the algorithms proceed recursively by pairing the 0 and 1-child of the node R with lowest index with the 0 and 1-child of the missing z-node above S, that are S and the terminal 0;
— an analogous procedure is used in the reverse case, where $R.index > S.index$.

Observe that when the procedures are called on the terminal 0 node the recursion stops returning the appropriate result. For instance, the algorithm UNION

```
UNION($R, S$)
  if ($R ==$ terminal 0) return $S$
  if ($S ==$ terminal 0) return $R$
  if ($R == S$) return $R$
  if ($R.index < S.index$) return GETNODE($R.index$, UNION($R_0$, $S$), $R_1$)
  if ($R.index > S.index$) return GETNODE($S.index$, UNION($R$, $S_0$), $S_1$)
  if ($R.index == S.index$) return GETNODE($R.index$, UNION($R_0, S_0$), UNION($R_1, S_1$))
```

returns directly S (resp. R) if R (resp. S) is the terminal 0, R_1 as a result of UNION(R_1, terminal 0) when $R.index < S.index$, and S_1 as UNION(terminal 0, S_1) when $R.index > S.index$. Analogously, the algorithm INTERSEC

```
INTERSEC($R, S$)
  if ($R ==$ terminal 0) return the terminal $0$
  if ($S ==$ terminal 0) return the terminal $0$
  if ($R == S$) return $R$
  if ($R.index < S.index$) return INTERSEC($R_0$, $S$)
  if ($R.index > S.index$) return INTERSEC($R$, $S_0$)
  if ($R.index == S.index$)
      return GETNODE($R.index$, INTERSEC($R_0, S_0$), INTERSEC($R_1, S_1$))
```

returns the terminal 0 if one of the two nodes in input is the terminal 0, while for $R.index < S.index$ it applies the z-rule returning directly the ZDD representing the intersection between R_0 and S since in this case the new node ($R.index$, INTERSEC(R_0, S), INTERSEC(R_1, terminal 0)) would be a z-node. The behavior of the algorithm is analogous when $R.index > S.index$. Finally, the DIFF algorithm

```
DIFF($R, S$)
  if ($R ==$ terminal 0) return the terminal $0$
  if ($S ==$ terminal 0) return $R$
if ($R == S$) return the terminal 0
if ($R.index < S.index$) return GETNODE($R.index$, DIFF($R_0$, $S$), $R_1$)
if ($R.index > S.index$) return DIFF($R$, $S_0$)
if ($R.index == S.index$) return GETNODE($R.index$, DIFF($R_0, S_0$), DIFF($R_1, S_1$))
```

exploits the fact that the difference between the terminal 0 and a ZDD Z is the terminal 0 and directly applies the z-rule whenever the terminal 0 becomes the 1-child of a node.

To implement each of these algorithms efficiently, avoiding an exponential blow-up of the recursive calls, a table M of results of the form $M[N, N'] = U$ is used, indicating that the result of applying the algorithm to the ZDDs with roots N and N' is the ZDD with root U. Then, before executing the algorithm on a pair of nodes, we first check whether the table contains an entry for these two nodes. If so, the results is returned without any further computation. Otherwise, we compute the result of the algorithm on N and N', and add a new entry on the table M before returning the result. If the table M is implemented with constant look-up and insertion time (e.g., as a two-dimensional array or as a dynamic hash table with a perfect hashing function producing no collisions), the complexity of the procedures is proportional to the product of the size of the two ZDDs in input.

3. INDEX RECONSTRUCTION COST

In this section we discuss error resilient indexes in ZDDs, we analyze the cost of the reconstruction of a corrupted index, and study the impact of the ZDD reduction rules on this cost. This study gives us the knowledge to describe in Section 4 a new index resilient version of ZDDs.

Monitoring the work on error resilient OBDDs [Bernasconi et al. 2013; 2015], we give some definitions useful to describe error resilient ZDDs. As before, we assume that the chosen variable ordering is $x_0 < x_1 < \ldots < x_{n-1}$. In order to facilitate the index reconstruction of a faulty node N we define the range of indexes that contains the original index of the node. Let N be an internal node in a ZDD Z, the *node range* $I_N = [i_P + 1, i_C - 1]$ is the range containing all the possible levels for N in Z, where i_P is the maximum index of N's parents in Z, and i_C is the minimum index of its children, where the leaves have “index” n, and if N is the root, i.e., N has no parent, $i_P = -1$.

Obviously, by definition of ZDD, the index of node N belongs to its range I_N. Thus, in presence of an error in the index i of N we have a lower and an upper bound for the reconstruction of i given by $i_P + 1$ and $i_C - 1$, respectively. In particular, if $i_P + 1 = i_C - 1$, then i is $i_C - 1$.

Let us now examine which characteristics make a ZDD more suitable to the reconstruction of a corrupted index. To this aim, we introduce a metric to measure the cost of the reconstruction of a corrupted index of a ZDD node in the worst case. The *index reconstruction cost* $C(N)$ of the faulty index i in the node N is given by the number of indexes that are candidate to be the correct one in N.

If we consider the case of one fault only in node N, we have that $C(N)$ is at most $|I_N|$. In particular, $C(N) = |I_N|$ whenever there is no additional knowledge on the structure of the ZDD. In the rest of this section, we therefore assume that $C(N) = |I_N|$. Instead, in Section 4 we will study ZDDs with a particular structure implying that $C(N) \leq |I_N|$.

In the best case, for each node N of a ZDD we have that $C(N)$ is 1, meaning that any index can be reconstructed in constant time (considering one single error). This condition is obviously satisfied by BDTs. In fact, in a BDT, all paths from the root to the terminal nodes contain exactly n nodes, where n is the number of input variables. Thus, for each node N, $C(N) = |I_N| = 1$. It is interesting to notice that the optimal

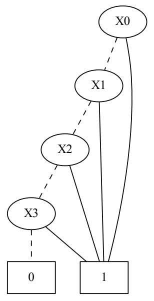


Fig. 2. A reduced ZDD where each internal node has cost 1.

cost $C(N) = 1$ can also be reached by reduced ZDDs, such as, e.g., the ZDD in Figure 2 where, even if the diagram is very compact, each internal node has cost equal to 1.

Recalling that a reduced ZDD can be constructed from a BDT by iteratively applying the reduction rules (m-rule and z-rule) and noticing that a BDT has optimal cost, we can study how the node range can increase using the two reduction rules.

We first consider the merge rule, i.e., the rule that is also used for the reduction of OBDDs. In the OBDD context, Theorem 1 in [Bernasconi et al. 2013] shows that this rule does not increase the index reconstruction cost of the nodes. In fact, the merge of isomorphic subgraphs does not change the node range of the involved nodes. In other words, each node N in a QR-BDDs is such that $C(N) = |I_N| = 1$.

On the other hand, the second reduction rule (z-rule), that distinguishes ZDDs from OBDDs, can increase the index reconstruction costs. For example, consider the ZDD in Figure 1(d), that can be obtained applying the z-rule to the QR-BDD in Figure 1(b). While each node of the QR-BDD has cost 1, node x_1 on the right of the ZDD has cost 2, since its range is increased by the z-rule. Note that not always the z-rule increases the node cost as already observed for the ZDD in Figure 2.

In [Bernasconi et al. 2013; 2015] we proposed an algorithm that reconstructs the index of a faulty node, restoring exactly the original OBDD and the associated function f. The algorithm is based on the use of the unique table. Once we have defined the range I_N containing the possible indices for a faulty node N, we can use the unique table to find the correct index of N in this range as follows: for each possible value l in the range I_N, the algorithm examines all nodes with index l and pointers equals to the ones of the faulty node N in the unique subtable associated to l, until it finds a node with the same memory address of N.

This algorithm could be applied and used to restore a faulty index in a ZDD as well, without any modification. However, the algorithm presents a major drawback: it assumes that the unique table is fault free, i.e., it is either implemented using error resilient linked lists [Aumann and Bender 1996] or it is stored in a safe memory area not affected by errors. This is a quite strong requirement, and for this reason, we do not propose such an algorithm as a practical solution for the reconstruction of a faulty index in a ZDD.

4. INDEX-RESILIENT REDUCED ZDDS

The analysis of the previous section shows that, while the merge rule never increases the overall index reconstruction cost, the application of the z-rule could increase it. In this section, we describe a new reduced ZDD model where we maintain some z-nodes in the diagram, in order to guarantee a constant index reconstruction cost for each node. In particular we will define a ZDD, called *index-resilient reduced ZDD*, satisfying the following properties:

(1) the index reconstruction cost of each node N is $C(N) = 1$;
(2) in presence of k nodes with a corrupted index, in an index-resilient ZDD, the cost needed to reconstruct a faulty index is $O(k)$;
(3) starting from a QR-BDD, the construction of the corresponding index-resilient reduced ZDD is linear in time;
(4) the index-resilient reduced ZDD is canonical.

We note that, since the z-rule can increase the index reconstruction cost, we could decide not to apply this rule during the reduction of a ZDD. In this way, we only use the m-rule and obtain a QR-BDD that has a cost $C(N) = 1$ for each node N. Recall that an important property of QR-BDD is that each node at level i has all parents at level $i-1$ and all children on level $i+1$. QR-BDDs are still a compact representation and could represent a convenient and canonical trade-off between memory saving, reduction time and error reconstruction time.

However, as we have already observed for the ZDD in Figure 2, the use of the z-rule does not always increase the index reconstruction cost. In other words, it is still possible to delete some z-nodes in a QR-BDD guaranteeing that, in the final OBDD, the index reconstruction cost of each node N is still $C(N) = 1$. Most importantly, as we will show in this section, it is possible to apply the z-rule to some z-nodes, and derive a *canonical* OBDD, *more compact* than a quasi-reduced one, and with a cost $C(N) = 1$ for each node N. We will call these OBDDs *index-resilient ZDDs*.

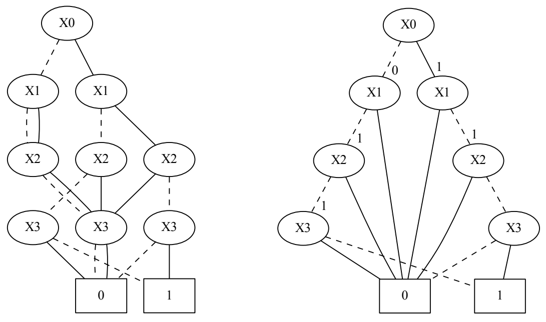
Definition 4.1 (*Index-Resilient ZDD*). An *Index-Resilient ZDD* is an OBDD obtained from a QR-BDD applying several times, possibly never, the z-rule guaranteeing that each internal node N on level i has at least one child on level $i+1$, for any level of the OBDD.

In particular, a QR-BDD is an index-resilient ZDD where each node on level i has *all* parents on level $i-1$ and *all* children on level $i+1$.

Observe that the index reconstruction cost for any node N in an index-resilient ZDD is $C(N) = 1$, since the variable index of a node N is directly given by $i = \min\{i_0, i_1\} - 1$, where i_0 and i_1 are the indexes of the 0- and 1-child of N. Note that, for any internal node N in a ZDD, the number of children of N is 2, but the number of parents of N can be $O(m)$, where m is the total number of nodes in the ZDD, and, in the worst case, $m \in \Theta(2^n/n)$ [Liaw and Lin 1992]. Note also that for the reconstruction of the index of N we do not need to know the indexes of its parents (whose number can be exponential in the number of variables), but only the indexes of its children. In fact, as shown below, we can define a structure where any node containing a variable x_i must have at least one child containing the variable x_{i+1}.

To compute a compact index-resilient ZDD, we start from a QR-BDD deleting some z-nodes while preserving the index-resilient property. For this purpose we first observe that in a QR-BDD there are at most one zr-node and one non-redundant z-node. These nodes are on level $n-1$. See, for example, the QR-BDD depicted in Figure 1(b).

We first consider the zr-node N_{zr}, if existing, in the QR-BDD. We can note that the removal of N_{zr} can generate new z-nodes, i.e., the 1-parents of N_{zr} (see Proposition 4.3). In particular, if N_{zr} has an r-node parent M, the removal of N_{zr} transforms M in a zr-node but the ZDD is not index-resilient since M is at level $n-2$ and has both children (the 0 constant) at level n. If we remove the zr-node M, we can generate again new z-nodes and one possible zr-node that has reconstruction cost greater than 1. We can,



(a) A zr-chain (b) Two z-chains

Fig. 3. Examples of zr-chain and z-chains. The value $P_C(N)$ is depicted over each node N in the z-chains. The z-chain on the left is removable.

therefore, consider the entire chain of r-nodes that ends with a zr-node defined as follows:

Definition 4.2 (*zr-chain*). A *zr-chain* in an index-resilient ZDD is a chain $C = N_1, N_2, \ldots, N_k$ (with $k \geq 1$) of nodes such that:

(1) N_1 has no redundant parents,
(2) N_i, with $i \in [1, \ldots, k-1]$, is an r-node and its unique child is N_{i+1},
(3) N_k is a zr-node.

The node N_1 is called *head* of the chain, and the leaf 0 is the *child* of the chain.

When $k = 1$ the chain corresponds to the zr-node N_1. As already observed, the deletion of a zr-chain generates new z-nodes in the obtained index-resilient ZDD:

PROPOSITION 4.3. *Let C be a zr-chain in an index-resilient ZDD Z. If C is removed from Z, then any node in Z that is a 1-parent of a node in C becomes a z-node.*

PROOF. Observe that when the chain $C = N_1, N_2, \ldots, N_k$ is deleted from Z, all edges pointing to its nodes are redirected to the subgraph pointed by the 0-edge of N_k, that is exactly the leaf 0. Thus, all nodes in Z whose 1-edge points to a node in the zr-chain C, i.e., all 1-parents of nodes in C, will end up with their 1-edge pointing to the leaf 0. □

For example, consider the QR-BDD in Figure 3(a) that contains a zr-chain of three nodes. The removal of the zr-chain will produce two new z-nodes (the nodes with indexes x_2 that are not part of the chain).

If we remove the entire zr-chain, the resulting OBDD is still an index-resilient ZDD, as proved in the following proposition. Moreover, in a QR-BDD the zr-chain is unique and, once deleted, the resulting index-resilient ZDD does not contain a new zr-chain.

PROPOSITION 4.4. *Let B be a QR-BDD containing a zr-chain C. We have that:*

(*1*) *C is the unique zr-chain in B,*
(*2*) *the OBDD B' derived by deleting C from B is an index-resilient ZDD,*
(*3*) *B' does not contain any zr-chain,*
(*4*) *B' does not contain any mergeable node.*

PROOF.

(1) First of all, note that in a QR-BDD B there can be at most one zr-node N, since N must have index $n-1$ and there are no isomorphic subgraphs in B. Moreover, any node in B cannot have two or more r-nodes as parents, either on the same level (they would be mergeable) or on different levels (otherwise, the most distant redundant parent would have an index reconstruction cost greater than 1, while B is a QR-BDD). Thus, any node in C has at most one redundant parent, and this implies that the zr-chain C is unique.
(2) The only nodes affected by the removal of the chain C are the parents of the nodes in C that do not belong to C, and therefore are not deleted. Thus, to prove that B' is still an index-resilient ZDD we must show that the index reconstruction cost of these nodes does not change. This property follows immediately from the fact that each parent $P \notin C$ of a node in C cannot be redundant and therefore has another child on the level immediately below, that can keep its reconstruction cost equal to 1 after the removal of C.
(3) Observe that the deletion of the zr-chain C cannot generate a new zr-node, and therefore a new zr-chain. Indeed, a new zr-node could be generated only by a non redundant node $N \notin C$ with the 0-edge pointing to the leaf 0, and the 1-edge pointing to a node in C. Now, since B is a QR-BDD, N must be on the level $n-1$ of B, but a node on this level cannot be connected to any node of C.
(4) Suppose by contradiction that N and N' are two mergeable nodes (i.e., roots of isomorphic subgraphs) at level i of B'. Since B is a QR-BDD, at least one of them must be a new node generated by the deletion of C. Thus, N and N' have a child at level $i+1$, and a child at level n (i.e., the leaf 0). If $i = n-1$, i.e., N and N' are on the level immediately above the leaves, neither N nor N' are generated by the deletion of C because the nodes with highest level affected by this deletion are the parents of the zr-node (i.e., are in level $n-2$). The fact that N and N' are mergeable is then in contradiction with the fact that B is a QR-BDD. If $i < n-1$, then both N and N' had a child on level $i+1$ in the zr-chain C in the original OBDD B. Since C is a chain and contains only one node per level, N and N' were parents of the same node of C, and were therefore mergeable in B. Thus we reach a contradiction with the fact that B is quasi-reduced.

□

We can observe that, after the deletion of a zr-chain in a QR-BDD, each node N at level i in the resulting index-resilient ZDD has both children at level $i+1$, or one child at level n (the 0 leaf) and a child at level $i+1$. Moreover, at level $n-1$ there exists at most one single z-node (i.e., the node that have the terminal 0 as 1-child and the terminal 1 as 0-child). The index-resilient ZDD obtained after the deletion of the zr-chain, can still contain z-nodes that can be removable. In order to efficiently test whether we can delete a z-node N, we first need the following parameter that counts the number of parents of N whose index reconstruction cost is affected by the deletion of N.

Definition 4.5 (P_C). Let N be a z-node in an index-resilient ZDD resulting by the deletion of the zr-chain from a QR-BDD. $P_C(N)$ is the number of parents P of N satisfying *at least one* of the following properties:

(1) Both children of P are z-nodes (possibly, the same z-node if P is redundant) and N is the 1-child of P;
(2) P has another child $N' \neq N$ on a level strictly greater than $i+1$, where i is the level of P (i.e., N' is the terminal node 0).

Note that if N is the root, then $P_C(N) = 0$. We can observe that Definition 4.5 derives from the fact that the cost $C(P) = 1$, of a node P at level i, is not increased by the deletion of its z-node child N if P has the other child $N' \neq N$, on level $i+1$ and N' cannot be removed. The child N' is not removed in two possible cases: 1) N' is not a z-node; 2) N' is a z-node (like N) but is the 1-child of P. This second criterion is an arbitrary choice due to the necessity of deleting one of the two z-nodes that are children of P while maintaining the index reconstruction cost and the canonicity of the representation. More precisely, when a node P has two children that are z-nodes, one of them can be removed without increasing the cost of P. In this paper we always remove the 0-child of P in order to guarantee that the resulting index-resilient ZDD is canonical (see Theorem 4.11). The choice of removing the 1-children is similar.

For example, see the index-resilient ZDD in Figure 3(b). Each z-node N in the figure has a value that corresponds to $P_C(N)$. We note that $P_C(N)$ can be efficiently computed with a simple visit of a index-resilient ZDD obtained after the deletion of the zr-chain from the QR-BDD.

When the QR-OBDD is constructed, the zr-chain (if existing) deleted, and P_C is computed, we can define chains of z-nodes (*z-chains*) and we can characterize the z-chains that can be removed, maintaining equal to 1 the index reconstruction cost of each remaining node. We therefore introduce the concept of *removable z-chain*.

Definition 4.6 (*Removable z-chain*). A *removable z- chain* in an index-resilient ZDD, which does not contain zr-chains, is a chain $C = N_1, N_2, \ldots, N_k$ (with $k \geq 1$) of z-nodes such that:

(1) N_i, with $i \in [2, \ldots, k]$, is the 0-child of N_{i-1},
(2) $P_C(N_1) = 0$,
(3) $\forall i \in [2, \ldots, k]$, $P_C(N_i) = 1$,
(4) if M is a z-node then $P_C(M) > 1$, where M is the the 0-child of N_k.

The node N_1 is called *head* of the chain, and M is called *child* of the chain.

The second requirement states that the head of the chain N_1 can be removed without affecting the reconstruction cost of its parents, as detailed in Proposition 4.8. Note that this requirement implies that all parents of N_1 are not z-nodes or r-nodes. The third requirement states that any other node N_i ($1 < i \leq k$) of the chain affects only the reconstruction cost of its parent in the chain. The last requirement guarantees that the removable z-chain is maximal. When the chain is composed by a single z-node node N, we have that N is removable when $P_C(N) = 0$. Consider, for example, the index-resilient ZDD in Figure 3(b). While, the z-chain on the left is removable, the z-chain on the right is not removable since the first node N_1 has $P_C(N_1) = 1$.

In an index-resilient ZDD there are no "crossing" z-chains, i.e., a node cannot be part of two different z-chains. This is a direct consequence of the following property.

PROPOSITION 4.7. *In an index-resilient ZDDs there are no nodes with two or more z-nodes as parents.*

PROOF. The property follows immediately from the fact that a node cannot have two parents that are z-nodes either on the same level (they would be mergeable) or on different levels (otherwise, the most distant parent would have an index reconstruction cost greater than 1). □

The following proposition shows that the deletion of all removable z-chains in an index-resilient ZDD Z does not change the overall index reconstruction cost, i.e., after the removal of the chains, each internal node on level i still has at least a child on level $i+1$, for any level i in Z.

PROPOSITION 4.8. *Let $C = N_1, N_2, \ldots, N_k$, $k \geq 1$, be a removable z-chain in an index-resilient ZDD, which does not contain a zr-chain. The OBDD resulting from the deletion of C is still an index-resilient ZDD.*

PROOF. The only nodes affected by the removal of C are the parents of the nodes in C that do not belong to C, and therefore are not deleted. Thus, to prove that Z is still an index-resilient ZDD we must show that the index reconstruction cost of these nodes does not change.

The second requirement in Definition 4.6 states that the head of the chain N_1 can only have non-removable siblings (as they are not z-nodes, or they are z-nodes but 1-child of their parents) that lye on the level immediately below the level of their parents, thus the deletion of N_1 does not affect the index reconstruction cost of its parents. The third requirement states that any other node N in the chain C can have only one parent without another child that can keep its reconstruction cost equal to 1: its z-node parent in the chain C, that will also be removed together with N; while all other parents P of N must have another child that does not belong to C and that can guarantee that $C(P) = 1$. □

Observe that once removable z-chains have been deleted, we are left with an index-resilient ZDD that can still contain some z-nodes: those that do not form a removable chains.

We can now propose a new OBDD reduction algorithm that, starting from a QR-BDD, deletes first the zr-chain and then all the removable z-chains. The following Theorem 4.9 shows that the deletion of removable z-chains does not construct new removable z-chains. Therefore, after the removal of the zr-chain, we can detect (and then delete) all the removable chains at the same time.

The reduction algorithm is based on a constant number of visits starting from a QR-BDD. The first visit is a breadth first search used to detect and remove the zr-chain, if it exists. Another visit is used to compute the parameter P_C for each z-node; then with a breadth first visit, all removable z-chains are identified and their nodes are removed with a final visit of the OBDD, executed by a simple recursive depth first visit that deletes from the OBDD all nodes identified as removable.

The correctness of the new reduction algorithm is proved in the following theorem.

THEOREM 4.9. *Let B be a QR-BDD. The reduction algorithm computes an index-resilient ZDD Z equivalent to B that contains neither removable z-chains nor a zr-chain.*

PROOF. First observe that the new reduction algorithm modifies the input OBDD B only applying the z-rule to a subset of its z-nodes (possibly a redundant z-node). Thus, the resulting ZDD Z is equivalent to B.

Now recall that the z-chains are deleted only after the removal of the zr-chain, that is when the ZDD does not contain any zr-node (the removal of the zr-chain does not generate new zr-nodes, as proved in Proposition 4.4).

Finally, observe that the deletion of the z-chains cannot introduce new removable z-nodes (or zr-nodes) in Z, as each 1-parent P of a node N in a chain cannot become a z-node. Indeed the edge of P pointing to N is redirected to the subgraph pointed by the 0-edge of the last z-node of the chain, and this subgraph cannot be the leaf 0 as the last z-node in the chain is not redundant (after the removal of the zr-chain the OBDD does not contain any zr-node). □

The cost of the algorithm is linear in the size of the QR-BDD in input, as it consists in a constant number of visits of the data structure.

We now formally introduce the concept of *Index-Resilient Reduced ZDD*.

Definition 4.10 (*Index-Resilient Reduced ZDD*). An index-resilient ZDD is *reduced* if it contains neither a zr-chain nor removable z-chains.

THEOREM 4.11. *Let Z be an index-resilient reduced ZDD obtained with the reduction algorithm. Then*

(*1*) *for each node N in Z, $C(N) = 1$;*
(*2*) *Z does not contain mergeable nodes;*
(*3*) *Z is canonical, i.e., given a function f and a variable ordering $<$, Z is the only index-resilient reduced ZDD with variable ordering $<$ that represents f.*

PROOF.

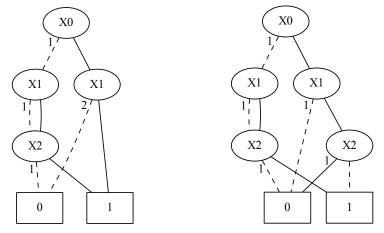
(1) Since the reduction algorithm starts with a QR-BDD, which is in particular index-resilient, Propositions 4.4 and 4.8 guarantee that the deletion of the zr-chain and of the removable z-chains maintains Z index-resilient, i.e., for each remaining internal node on level i there exists at least a child on level $i + 1$, for any level of Z.
(2) We have already proved in Proposition 4.4 that after the deletion of the zr-chain there are no mergeable nodes in the obtained OBDD B'. We must then show that the deletion of the z-chains does not generate mergeable nodes. Suppose, by contradiction, that N and N' are two mergeable nodes at level i of Z. Since B' does not contain mergeable nodes, at least one of them must be a new node (i.e., a node with different children) generated by the deletion of the z-chains. N and N' have a child at level $i+1$ (otherwise the removed chains were not removable), and a child N_j at level $j > i$. If $j = i + 1$, N and N' were not parents of a node in a z-chain and the fact that they are mergeable is in contradiction with the fact that the algorithm started with a QR-BDD. If $j > i+1$, then the original QR-BDD contained a removable z-chain between N and N_j and one between N' and N_j. This in turns implies that N_j had two z-node parents on the same level in the original QR-BDD. Thus, we reach a contradiction with the fact that the starting OBDD is quasi-reduced, as the two z-node parents of N_j are mergeable.
(3) Given a variable ordering $<$ and a QR-BDD (which is canonical once fixed $<$), there is an unique way to delete the zr-chain and the removable z-chains, since each node cannot be part of two different chains and the removal of the chains does not produce new chains or mergeable nodes. Here it is important to recall that there is no ambiguity in the definition of removable z-chains: given any node P with two different z-nodes as children, only the 0-child can be head or part of a removable z-chain. The index-resilient reduced ZDD is then a canonical form.

□

For example, Figure 4 shows the index reconstruction cost of each node in the ZDD of Figure 1(d) (Figure 4(a)), and the index reconstruction cost of each node in the corresponding index-resilient reduced ZDD, where for each node N we have that $C(N) = 1$ (Figure 4(b)). We can note that the standard ZDD contains a node N with index reconstruction cost equal to 2 (the second x_1), this implies that if an index error occurs in N, the variable reconstruction is not unique since N could contain x_1 or x_2.

In index-resilient ZDDs, the index reconstruction cost remains limited even in presence of more than one error on the indexes, as stated and proved in the following theorem, that shows a result similar to the one obtained for OBDDs in [Bernasconi et al. 2015].

THEOREM 4.12. *The reconstruction cost of a node N on level i in an index-resilient reduced ZDD Z affected by k errors on the indexes is $O(\min(k, 2^{n-i}))$.*



(a) ZDD with index reconstruction costs (b) index-resilient ZDD with index reconstruction costs

Fig. 4. A reduced ZDD and the corresponding index-resilient reduced ZDD where, for each node, the index reconstruction cost is 1.

PROOF. First, we note that, starting from the node N on level i, there is always a complete path (i.e., a path containing the variable $x_i, x_{i+1}, \ldots, x_n$) that ends on a leaf. This path can be exploited to reconstruct the index of N. In fact, the index of N can be computed using the indexes of its children in the following way. Let j_0 and j_1 be the levels of the 0-child and of the 1-child of N. If both children are not affected by errors, then the index of N is $i = \min\{j_0, j_1\} - 1$. Otherwise, we recursively proceed on the index-resilient ZDD rooted in any corrupted child of N, and we will restore the index of N when both the indexes of its children will be corrected. The recursion stops on corrupted nodes with two uncorrupted children. Note that we can consider the two terminal nodes (the leaves of the OBDD) uncorrupted, as they could be memorized in a safe memory, or duplicated. In the worst case, the reconstruction cost is the minimum between the dimension of the OBDD rooted in N (i.e., $O(2^{n-i})$) and the total number of corrupted nodes in Z (i.e., $O(k)$). The number of visited nodes is then $O(\min(k, 2^{n-i}))$. $\square$

Let us now compare the number of nodes of index-resilient reduced ZDDs with the size of quasi-reduced BDDs and standard ZDDs. Consider a QR-BDD, an index resilient reduced ZDD, and a standard ZDD representing the same function with n variables, and denote by n_Q, n_{IR}, and n_Z, respectively, their size. Since the index-resilient reduced ZDD is derived from the corresponding QR-BDD deleting some, possibly none, z-nodes, while a standard ZDD is derived deleting *all* z-nodes from the QR-BDD, we have that $n_Z \leq n_{IR} \leq n_Q$.

It is also possible to give an upper bound on n_{IR} in terms of n_Z. Indeed, we can observe that in the worst case, in order to make index-resilient a standard ZDD, we would need to introduce a chain of at most n variables for each node. E.g., the chain $i+1, i+2, \ldots j-1$ is inserted between the node with index i and its child with index j with $j \neq i+1$, if it exists. Therefore, we can bound n_{IR} as follows: $n_{IR} \leq n \cdot n_Z$. Putting all together, we then get the bounds:

$$n_Z \leq n_{IR} \leq \min\{n \cdot n_Z, n_Q\}\,.$$

Notice that if the size of the ZDD is polynomial in the number of variable, then the same holds for the size of the index-resilient version of the decision diagram.

The limited size increase of index-resilient ZDDs with respect to ZDDs has been verified even experimentally. Indeed, the experimental results presented in Section 6 show that index-resilient reduced ZDDs are 14% more compact than quasi reduced BDDs, and only 4% less compact than standard ZDDs. This result, in turns, is in line with previous studies (for instance, see [Liaw and Lin 1992]) showing how the merge rule makes a much more significant contribution to memory saving than other reduction rules, and in general it is the rule allowing the exponential size reduction that is obtained, in some cases, transforming a binary decision tree into a ZDD or a BDD.

To conclude this section, we would like to discuss two important issues concerning the proposed index-resilient ZDD model. First of all, as we have just observed, an index-resilient reduced ZDD usually contains more nodes than a standard reduced ZDD, as some z-nodes are kept in the diagram. However, recall that index resilient ZDDs (reduced or not) do not need the memorization of the unique table. Moreover, they guarantee a better and more efficient reconstruction of faulty indexes. Secondly, even if similar results were already known for standard OBDDs, in our opinion the growing interest in this particular data structure justifies the extension of the idea of index-resilient BDDs to the family of zero-suppressed decision diagrams. In particular, even if index resilient BDDs and index-resilient ZDDs have compatible sizes on average (see Section 6), the areas of application of the two data structures are different, and the use of BDDs in applications more suitable for ZDDs could lead to an overall loss of efficiency. For this reason it is important to have index-resilient versions of both families of decision diagrams, guaranteeing a constant reconstruction cost for each faulty index. We finally recall that an additional advantage of index-resilient ZDDs is that the property that each node has at least one child on the level immediately below could be exploited for error detection, too. For instance, the presence of faulty indexes can be reported any time the index of a node and those of its children in the diagram are not consistent with the fixed variable ordering.

5. OPERATIONS ON INDEX-RESILIENT ZDDS

As we have seen in the previous section, it is possible to construct index-resilient reduced ZDDs starting from QR-BDDs. Moreover, the whole process of construction of this data structure is index-resilient. Indeed, the construction of an index-resilient ZDD starts from a binary decision tree that is transformed into a QR-BDD applying the merge rule, and in both models each node has all children on the level immediately below. Therefore, during the execution of the reduction algorithm on a QR-BDD, we can always guarantee that each node has at least one child on the level below, thus one or more faulty indexes can be immediately detected and restored as explained in the proof of Theorem 4.12.

However, ZDDs are not usually constructed from quasi-reduced ones, but instead they are dynamically constructed applying some basic operations to other ZDDs. Therefore, we now discuss how the basic recursive procedures reviewed in Section 2.2 can be modified in order to guarantee that the ZDD in output is resilient to index faults, even if some errors occur during the computation. In other words, we will show how to obtain error resilient algorithms for basic ZDD manipulation, that is, algorithms capable of dealing with errors (in the data structures) occurring during their execution.

First of all, an important change with respect to the standard algorithms is that our implementations do not use the procedure GETNODE. Recall that this procedure is used to maintain the ZDD reduced, avoiding the insertion of both mergeable and z-nodes in the ZDD under construction. To achieve this, GETNODE relies on the use of an additinal data structure: the unique table (see Section 2.2). Thus, in order to design error resilient algorithms, we should make the unique table resilient to errors as well, for instance assuming that it is implemented using error resilient linked lists [Aumann and Bender 1996] or that it is stored in a safe memory area not affected by errors. Since these are strong requirements, we prefer to avoid the use of the unique table or other data structures, at the expense of a possibly higher time complexity, that is however still polynomial. We only require that the two terminal nodes (the leaves of the ZDD) are always uncorrupted, and therefore that they are memorized in a safe memory or duplicated.

Secondly, we modify the implementation of the algorithms in order to guarantee the following invariant property.

PROPERTY 1 (**Invariant**). *At each step of the execution of the operations, for any ZDD involved, each internal node on level i has at least one child on level $i+1$.*

The invariant guarantees that during the execution of the operations it is always possible to detect and reconstruct faulty indexes as described in the proof of Theorem 4.12. As we will see, to maintain this invariant we will have to keep some z-nodes, or even insert chains of z-nodes, in the diagram. The final ZDD will then be transformed into an index-resilient reduced ZDD through a specific procedure, that we call REDUCTION.

This second reduction phase might be avoided, or performed only at the end of the overall computation, if time efficiency is a major concern for the current application, while space efficiency and canonicity of the decision diagrams are not crucial issues. Indeed, as discussed in this section, the ZDDs computed by these algorithms satisfy by construction Property 1 and this property guarantees the index-resiliency of the decision diagrams, even when they contain mergeable nodes and z-nodes that could be eliminated executing the REDUCTION procedure. Otherwise, if the application requires the use of reduced and canonical forms, we can execute the reduction procedure after each operation. This is still a feasible solution, as the reduction can be performed in polynomial time.

In the rest of the section, with a slight abuse of notation, we will call index-resilient ZDD any ZDD that satisfies Property 1, but may contain some mergeable nodes. Recall from Section 4, that the absence of mergeable nodes is crucial for the correctness of the reduction algorithm for the computation of index-resilient *reduced* and *canonical* ZDDs, while it is not a necessary condition for making the diagrams resilient to errors on indexes. In particular, Theorem 4.12 holds for any ZDD satisfying Property 1 only.

When performing operations on ZDDs, we always suppose that the ZDDs in input are index-resilient ZDDs, and we will show that the ZDD in output is index-resilient (i.e., it satisfies Property 1) and, after execution of the REDUCTION procedure, becomes an index-resilient reduced ZDD.

In summary, we present new operations, on index-resilient ZDDs, that are error resilient during their execution and produce an index-resilient ZDD as output. For any presented operation the overall strategy is depicted in Figure 5. In particular, we apply the index-resilient version of the operations (shown in Section 5) and we perform a reduction of the obtained index-resilient ZDD (as shown in Section 5.2). In the reduction phase we first reconstruct the QR-BDD, inserting a polynomial number of new nodes and merging all mergeable nodes that could have been inserted in the previous phases. We then remove the zr-chain (if any) and all the removable z-chains (as described in Section 4).

5.1. Index Resilient Procedures

We now describe the new index-resilient implementations of the standard operations on ZDDs. Let us first consider the standard set operations union, intersection and difference applied to two index-resilient ZDDs rooted in the nodes R and S.

5.1.1. Procedures with two Input ZDDs. As already mentioned, we do not use the procedure GETNODE and we do not delete z-nodes during the execution of the algorithms. As a consequence, in the index-resilient versions of the three procedures UNION, INTERSEC, and DIFF the recursion stops only when R and S are both terminal nodes,

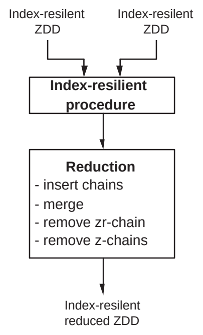


Fig. 5. General execution-flow for operations on index-resilient ZDDs.

without exploiting the possibility of an earlier termination when one of the two nodes is the terminal 0 (possibility that, however, would not improve the worst-case complexity of the algorithms). Thus, in the following implementations, if R and S are terminal nodes, a new terminal node is returned having the appropriate value 0 or 1, depending on the operation. Otherwise, if at least one node is non-terminal we proceed according to the index of the nodes. If the two nodes R and S have different indexes, we proceed by pairing the 0 and 1-child of the node with lowest index with the 0 and 1-child of the z-node that has been removed from the ZDD rooted in the node with higher index. Suppose for instance that $R.index < S.index$. This means that a z-node N, with index equal to $R.index$ and with S as 0-child and the terminal 0 as 1-child, has been removed from the ZDD rooted in S. Thus we create a new node with index $R.index$, and recursively apply the algorithms on R_0 and N_0 (i.e., S) to generate the ZDD whose root becomes the 0-child of the new node, and on R_1 and N_1 (i.e., the terminal 0) to generate the 1-child, without applying the z-rule when possible (as for the intersection and set difference operations). A similar procedure is used in the reverse case, where $S.index < R.index$. Finally, if the two nodes have the same index i, we create a new node with index i, and we apply the algorithms recursively on R_0 and S_0 to generate the ZDD whose root becomes the 0-child of the new node, and on R_1 and S_1 to generate the 1-child. In this way, we get the following versions of the procedures:

```
UNION($R, S$)
  **if** ($R$ and $S$ are terminal nodes and $R \neq S$) **return** the terminal 1
  **if** ($R == S$) **return** $R$
  **if** ($R.index < S.index$) **return** new node ($R.index$, UNION($R_0$, $S$), $R_1$)
  **if** ($R.index > S.index$) **return** new node ($S.index$, UNION($R$, $S_0$), $S_1$)
  **if** ($R.index == S.index$) **return** new node ($R.index$, UNION($R_0, S_0$), UNION($R_1, S_1$))

INTERSEC($R, S$)
  **if** ($R$ and $S$ are terminal nodes and $R \neq S$) **return** the terminal 0
  **if** ($R == S$) **return** $R$
  **if** ($R.index < S.index$)
      **return** new node ($R.index$, INTERSEC($R_0, S$), INTERSEC($R_1$, terminal 0))
  **if** ($R.index > S.index$)
      **return** new node ($S.index$, INTERSEC($R, S_0$), INTERSEC(terminal 0, $S_1$))
  **if** ($R.index == S.index$)
      **return** new node ($R.index$, INTERSEC($R_0, S_0$), INTERSEC($R_1, S_1$))

DIFF($R, S$)
  **if** ($R$ and $S$ are terminal nodes)
```

```
    if ($R == S$) return the terminal 0
    else return $R$
  if ($R.index < S.index$)
    return return new node ($R.index$, DIFF($R_0$, $S$), DIFF($R_1$, terminal 0))
  if ($R.index > S.index$) return new node ($S.index$, DIFF($R$, $S_0$), DIFF(terminal 0, $S_1$)
  if ($R.index == S.index$) return new node ($R.index$, DIFF$(R_0, S_0)$, DIFF$(R_1, S_1)$)
```

We now show that these three algorithms executed on two index-resilient ZDDs preserve by construction the invariant Property 1. Indeed, if the algorithms are called on two nodes R and S with the same index i, then a new node with index i is created, and the algorithms are recursively executed on the two 0-children and on the two 1-children of R and S. If instead R and S have different indexes, a new node with the *lowest* index between those of R and S is created, and the algorithms proceed recursively by pairing the 0 and 1-child of the node with lowest index with the other node and with the terminal 0, respectively. Now, without loss of generality, suppose that R.index $= i$ and S.index $\geq i$. Then the new node has index i and at least one of its children has index $i+1$, since *(i)* the procedures recur on the two children of R, one of which certainly has index $i+1$ as R is index-resilient, paired with S and with the terminal 0, whose indexes are in both cases greater or equal to $i+1$; *(ii)* the procedures always choose as index of the new node the lowest index between those of the two nodes in input.

As the standard implementations, these versions of the algorithms make use of the matrix M to avoid an exponential blow-up of the recursive calls. Instead, they do not exploit the unique table. Therefore, the ZDD in output can contain mergeable nodes. Still, the use of the matrix M is enough to guarantee that the worst-case complexity of the procedures is quadratic, and that the size of the resulting ZDD is bounded by the product of the size of the two ZDDs in input.

5.1.2. Procedures with one Input ZDD. We now consider the two procedures that take as input a single index-resilient ZDD, i.e., SUBSET and CHANGE.

The index-resilient version of the SUBSET procedure is quite different from the standard one. In fact, this operation deletes part of the ZDD, and we must preserve the invariant property in order to assure error correction during its execution. For this reason the recursion must be executed on the children of a node Z, instead of the node itself. In fact, if a child is changed we must insert a z-chain (between Z and the child) in order to guarantee the invariant property for Z. Thus, we have a main procedure SUBSET that solves the problem when the index ind corresponds to the one on the root of the input index-resilient ZDD, otherwise it calls a recursive procedure SUBRIC that solves the problem for the children of the input node.

```
SUBSET$(Z, ind, val)$
  if ($ind == Z.index$) return $Z_{val}$
  else return SUBRIC$(Z, ind, val)$

SUBRIC$(Z, ind, val)$
  for $i = 0$ to 1
    $C = Z_i$
    if ($C.index > ind$ and $val == 1$)
      construct a zr-chain from a new node $N$ of index $C.index$ to the terminal 0
      $Z_i = N$
    if ($C.index > ind$ and $val == 0$) $Z_i = C$
    if ($C.index == ind$)
      construct a z-chain from a new node $N$ of index $(ind + 1)$ to $C_{val}$
```

```
        $Z_i = N$
    if ($C.index < ind$)
        $Z_i$ = new node ($C.index$, SUBRIC($C_0, ind, val$), SUBRIC($C_1, ind, val$))
return $Z$
```

The procedure CHANGE simply swaps the 0-child with the 1-child, thus Property 1 is guaranteed and we have to be careful only when ind is the index of a removed z-node. In this case the node is reinserted, but it is no longer a z-node. We now have to add a zr-chain to the 0 terminal in order to maintain the invariant. Note that the canonicity of the ZDD is obviously lost (because of the swapping of some sub-ZDDs), but it will be restored during the reduction phase.

```
CHANGE($Z, ind$)
if ($Z.index > ind$)
    construct a zr-chain from a new node $N$ of index $ind + 1$ to the terminal 0
    return new node ($ind$, $N$, $Z$)
if ($Z.index == ind$) return new node ($ind$, $Z_1$, $Z_0$)
if ($Z.index < ind$) return new node ($Z.index$, CHANGE($Z_0, ind$), CHANGE($Z_1, ind$))
```

Even these last two procedures make use of the matrix M and do not exploit the unique table. Their worst-case complexity is bounded by the product between the size of the ZDD in input and the number of variables. The last factor of the product is due to the insertion of the z-chains.

5.1.3. Error Handling. During the execution of all the procedures described in Sections 5.1.1 and 5.1.2 some errors may occur, in particular we can have errors in the indexes of the nodes and errors in the recursive table M. We now discuss ho to handle these errors.

Recall that, for all the operations described, we have guaranteed the invariant property, i.e., that any internal node on level i of the computed ZDD has at least one child on level $i + 1$. Exploiting this property, we can always reconstruct the correct index, whenever an error occurs during the computation, using the strategy described in the proof of Theorem 4.12.

Moreover, following the strategy described in [Bernasconi et al. 2015] for index-resilient OBDDs, we can correct the errors occurring in the matrix M used for memorizing the output of the recursive calls in order to avoid an exponential number of re-computations. Position $M[N, N']$ of the matrix M contains NULL if the considered operation is never computed on the input (N, N'), otherwise, $M[N, N']$ contains the pointer to the root node of the sub-ZDD that is the solution of the operation. If $M[N, N']$ is corrupted, the algorithm simply computes (possibly, again) the operation on the input (N, N'). This means that, in the worst case, the number of re-computations of the same pointers in M is $O(r_M)$ where r_M is the total number of errors in M.

5.2. Reduction

As already observed, the result of the operations described in Section 5.1 is not usually an index-resilient reduced ZDD, since some mergeable nodes and some removable chains could have been inserted by the operations. This is due to the absence of the unique table. In order to obtain a reduced form, we now define a polynomial reduction strategy on the computed ZDD Z. In particular, Figure 5 shows the steps of this reduction.

As already pointed out, this reduction phase might be avoided, or performed only at the end of the overall computation, if time efficiency is a major concern for the current application, while space efficiency and canonicity of the decision diagrams are not crucial issues.

The reduction first transforms the given ZDD Z in a QR-BDD, first inserting the missing chains and then merging all mergeable nodes. This step is necessary since the operations can change the 0-child and the 1-child, possibly making non-removable a z-chain that has been removed. Therefore, any chain must be reconstruct and the ZDD reduced again in order to maintain the canonical form. This insertion of chains is polynomial in the number of nodes and in the number of variables n since, in the worst case, we introduce a chain of n variables for each node, i.e., the chain $i+1, i+2, \ldots j-1$ is inserted between the node with index i and its child with index j with $j \neq i+1$, if exists. Second, we must perform a merge step, since the obtained ZDD Z can contain mergeable nodes. This task can be done through a quadratic DFS visit of Z: when a node N is visited, we perform a second visit that identifies and merge all nodes mergeable with N. Finally we have a QR-BDD, and we can remove the zr-chain and all the the removable z-chains from it, as shown in Section 4.

During the execution of the reduction procedure, since the invariant Property 1 holds for any ZDD involved, we can dynamically correct errors.

6. EXPERIMENTAL RESULTS

This section shows the experiments we have run in order to evaluate size (i.e., number of nodes) and computational time for index-resilient ZDDs.

The algorithms have been implemented in C and the experiments have been run on a Linux Intel Core i7, 3.40 GHz CPU with 8 GB of main memory.

In order to show the gain and the loose, in number of nodes, of index-resilient reduced ZDDs, we compare these new forms with QR-BDDs and standard reduced ZDDs. Recall that, considering the same variable ordering, the number of node of an index-resilient ZDD (n_{IR-ZDD}) is less or equal to the number of nodes of the corresponding QR-BDD (n_{QR-BDD}), and it is greater or equal to the number of nodes of the ZDD (n_{ZDD}) for the same function: $n_{ZDD} \leq n_{IR-ZDD} \leq n_{QR-BDD}$.

The main aim is to determinate whether IR-ZDDs are compact as ZDDs, or not. Our second objective is to compare the index-resilient definition of ZDDs with the index-resilient definition of OBDDs described in [Bernasconi et al. 2015]. In particular, since the reduction rule for index-resilient OBDDs and index-resilient ZDDs are quite different, a purpose of the experiments is to determinate whether the given definition of index-resilient ZDDs is practical or not.

For the first set of experiments we have considered the benchmarks taken from the classical library LGSynth93 [Yang 1991]. The decision diagrams have been constructed considering the union of the on-set and dc-set. As some benchmarks have multiple outputs, we have constructed one ZDD for each output. In the following, we report a significant subset of the functions as representative indicator of our experiments.

In Table I we compare the number of nodes of index-resilient ZDDs with the size of QR-BDDs and standard ZDDs. The first column reports the name of the benchmarks, the second column contains the number of nodes in the QR-BDD. The following group of three columns reports the number of nodes in the index-resilient ZDD, the number of nodes in the standard ZDD and the percentage of outputs containing a zr-chain. The last group of two columns reports the number of nodes for index-resilient OBDDs and standard OBDDs [Bernasconi et al. 2015].

In our experiments we have also evaluated the computational times for constructing, starting from a QR-BDD, index resilient ZDDs (resp., BDDs) and the reduction times for standard ZDDs (resp., OBDDs). In both cases, the computational times were found to be negligible, at most a few milliseconds, for all the benchmarks considered showing that index-resilient ZDDs and standard ZDDs have limited and similar construction times.

Table I. Size (number of nodes) in the considered DDs.

Bench	QR	IRZDD	ZDD	zr-chain	IRBDD	BDD
add6	698	677	671	86%	547	372
addm4	495	428	420	100%	417	361
alu1	206	163	145	100%	109	31
alu2	455	432	426	75%	396	330
alu3	495	474	468	75%	429	358
apla	469	408	402	91%	384	331
b11	493	246	201	93%	294	160
bench1	1132	1102	1090	100%	1086	1048
br1	346	252	182	100%	265	242
br2	285	194	157	100%	190	174
clpl	140	115	115	80%	84	53
dc2	171	102	93	86%	129	103
dist	290	257	252	100%	266	247
dk17	422	383	377	100%	339	304
dk27	248	239	239	0%	185	185
ex1010	1773	1742	1729	100%	1718	1659
ex5	1188	833	739	94%	762	646
exp	858	601	470	100%	796	751
exps	1650	1213	1058	100%	1395	1278
fout	266	238	235	100%	238	222
inc	236	168	155	100%	186	126
lin.rom	1038	936	895	78%	894	849
luc	667	435	416	100%	543	375
m2	405	203	198	100%	282	229
m3	433	249	245	100%	284	244
m4	554	383	373	100%	440	388
max1024	512	407	407	100%	473	443
max128	507	345	345	88%	377	322
max512	329	259	259	100%	285	260
mp2d	413	304	290	100%	299	151
newapla	272	163	144	100%	134	78
newbyte	72	25	20	100%	40	40
newcpla1	352	212	186	100%	207	155
newtpla2	85	53	46	100%	49	43
newxcpla1	407	266	239	96%	255	176
opa	3091	1819	1519	100%	2315	1164
p1	688	601	587	94%	581	538
p3	504	442	429	93%	416	382
p82	182	97	83	100%	127	122
pope.rom	803	644	598	88%	603	537
prom1	4027	3566	3358	100%	3729	3577
prom2	1814	1546	1502	100%	1585	1501
t3	300	198	171	100%	227	111
t4	399	298	292	100%	320	213
test1	786	747	732	100%	744	711
test2	11678	11527	11482	100%	11431	11195
test3	6713	6581	6538	100%	6490	6302
test4	1512	1452	1441	77%	1361	1314
tms	438	261	224	100%	310	279

In order to have a statistical idea of the size of the diagrams, we have run a second set of experiments on all the Boolean functions with 2, 3 and 4 variables, which are 16, 256, and 65536 in number, respectively. Table II shows the results. The first column reports the number of variables. The second column shows the difference between the total number of nodes in the index-resilient ZDDs and the QR-BDDs, the third columns reports the difference between the total number of nodes in the ZDDs and the index-resilient ZDDs. The last two columns show the same difference for index-resilient BDDs with respect to QR-BDDs and reduced OBDDs.

Table II. Differences of size (number of nodes) of the DDs for all the functions with 2, 3 and 4 variables.

NV	IRZDD-QR	ZDD-IRZDD	IRBDD-QR	BDD-IRBDD
2	16	2	16	2
3	414	76	410	80
4	150956	35070	147086	38940

Table III. Index Reconstruction Cost for standard ZDDs.

Bench	Nodes	tot IRC	max IRC	Nodes with IRC > 1	%
alu1	145	171	2	26	18%
b11	201	320	6	64	32%
bench1	1090	1110	2	20	2%
br1	182	296	5	78	43%
br2	157	220	4	49	31%
clpl	115	115	1	0	0%
dc2	93	114	3	16	17%
exp	470	747	5	195	41%
exps	1058	1376	6	197	19%
inc	155	183	3	23	15%
lin.rom	895	951	3	52	6%
luc	416	475	3	49	12%
m2	198	246	8	31	16%
m3	245	286	6	29	12%
m4	373	417	7	32	9%
newapla	144	185	5	26	18%
newbyte	20	45	5	13	65%
newcpla1	186	246	5	41	22%
newtpla2	46	78	9	12	26%
newxcpla1	239	298	5	39	16%
opa	1519	2071	6	363	24%
p82	83	120	4	25	30%
pope.rom	598	672	5	52	9%
prom1	3358	3810	6	350	10%
t3	171	234	4	49	29%
t4	292	310	2	18	6%
tms	224	320	8	81	36%

The results in the two tables show that index-resilient reduced ZDDs are a good trade-off between QR-BDDs and reduced ZDDs. In fact, the reduction of a QR-BDD in an index-resilient reduced ZDD pays significantly, since index-resilient reduced ZDDs are more compact than QR-BDDs and the number of nodes in an index-resilient reduced ZDD is not too much greater that the number of nodes of a standard reduced ZDD. In practice, from the results on the benchmark functions, we have that, in average, index-resilient reduced ZDDs are 14% more compact than QR-BDDs, and that standard reduced ZDDs are 4% more compact than index-resilient reduced ZDDs.

Moreover, if we consider index-resilient reduced BDDs we have that index-resilient reduced BDDs are 13% more compact than QR-BDDs, and that standard reduced OBDDs are 10% more compact than index-resilient reduced BDDs. This result, experimentally shows that the definition of index resilient reduced ZDDs seems to give a more compact form that the one described in [Bernasconi et al. 2015] for OBDDs.

In order to understand how many errors cannot be handled in standard ZDDs, we have computed the index reconstruction cost for the considered benchmark functions. Table III shows the results of a significant subset of the benchmark functions. The first column reports the name of the benchmarks and the second contains the number of nodes in the classical ZDD representation. The following group of two columns reports the index reconstruction cost of the entire ZDD (i.e., the sum of the index reconstruction costs for each node) and the maximum value of the index reconstruction cost of a single node in the ZDD. The last group of two columns shows the number, and the percentage, of nodes with index reconstruction costs strictly greater than 1. This number gives us an indication about how many node errors cannot be handled in standard ZDDs. For example, consider the benchmark *exp*, the 41% of its nodes has a index reconstruction cost strictly greater than 1 (with maximum 6). In order to reconstruct the correct index, if we do not use the unrealistic unique table in a trusted memory, for any node with index reconstruction cost c, we can only guess the correct index value with probability $1/c$. In the example, if the node with index reconstruction cost 6 contains an index error, the reconstruction can randomly guess the correct index value with probability 17%. Recall that in index-resilient ZDDs this fact can never happen, since the index reconstruction cost for each node is always 1. In fact, by construction of index-resilient ZDDs, the index of each node can be simply reconstructed exploiting the indexes of its children.

7. CONCLUSION

This paper has proposed a new ZDD canonical form that is resilient to errors in indexes under the single-component error model. This form can be derived in linear time starting from a quasi-reduced BDD, or in polynomial time combining other index-resilient ZDDs. Even if similar results were known for standard OBDDs [Bernasconi et al. 2013; 2015], in our opinion the many applications of this data structure justify the extension of the study on error resiliency from standard BDDs to the family of ZDDs. In particular, we have focused on error on indexes only, as the ad hoc solution already proposed for OBDDs in [Bernasconi et al. 2015] to handle errors on pointers can be immediately applied to ZDDs, without modification. Instead, the techniques for the construction, reduction and dynamical computation of index-resilient ZDDs are not immediate generalizations of those defined in [Bernasconi et al. 2015] for OBDDs, as reduction rules and basic operations for the manipulation of ZDDs are different from those typically applied to OBDDs.

The proposed analysis completes the study on the error correction of the two families of binary decision diagrams that are mainly used in applications. The generalization of these results to other families of decision diagrams such as ordered functional decision diagrams (OFDDs) [Kebschull and Rosenstiel 1993] and ordered Kronecker functional decision diagrams (OKFDDs) [Becker et al. 1997] should be easily accomplished, as the reduction rules for OFDDs are syntactically identical to those applied to ZDDs, even if the semantic is different.

Error detection is another important issue, that we have not considered in the present analysis, where we assumed perfect error detection capabilities. Further work on this subject could therefore be a deeper study of error detection in binary decision diagrams.

REFERENCES

AKERS, S. 1978. Binary Decision Diagrams. *IEEE Transactions on Computers 27,* 6.

AUMANN, Y. AND BENDER, M. 1996. Fault Tolerant Data Structures. In *37th Annual Symposium on Foundations of Computer Science (FOCS)*.

BECKER, B., DRECHSLER, R., AND THEOBALD, M. 1997. On the expressive power of okfdds. *Formal Methods in System Design 11,* 1, 5–21.

BERNASCONI, A. AND CIRIANI, V. 2014. Zero-suppressed binary decision diagrams resilient to index faults. In *Theoretical Computer Science - 8th IFIP TC 1/WG 2.2 International Conference, TCS 2014, Rome, Italy, September 1-3, 2014. Proceedings*. 1–12.

BERNASCONI, A., CIRIANI, V., AND LAGO, L. 2013. Error Resilient OBDDs. In *IEEE Symposium on Design and Diagnostics of Electronic Circuits and Systems (DDECS)*. 246–249.

BERNASCONI, A., CIRIANI, V., AND LAGO, L. 2015. On the Error Resilience of Ordered Binary Decision Diagrams. *Theor. Comput. Sci. 595*, 11–33.

BRYANT, R. 1986. Graph Based Algorithm for Boolean Function Manipulation. *IEEE Transactions on Computers*.

DRECHSLER, R. 1998. Verifying integrity of decision diagrams. In *Computer Safety, Reliability and Security*.

EBENDT, R., FEY, G., AND DRECHSLER, R. 2005. *Advanced BDD Optimization*. Springer.

FINOCCHI, I., GRANDONI, F., AND ITALIANO, G. 2005. Designing reliable algorithms in unreliable memories. In *Algorithms ESA 2005*. Lecture Notes in Computer Science.

FINOCCHI, I., GRANDONI, F., AND ITALIANO, G. 2007. Designing reliable algorithms in unreliable memories. *Computer Science Review 1,* 2, 77–87.

FINOCCHI, I., GRANDONI, F., AND ITALIANO, G. F. 2009. Optimal resilient sorting and searching in the presence of memory faults. *Theor. Comput. Sci. 410,* 44, 4457–4470.

FINOCCHI, I. AND ITALIANO, G. 2008. Sorting and Searching in Faulty Memories. *Algorithmica*.

ITALIANO, G. 2010. Resilient Algorithms and Data Structures. In *Algorithms and Complexity*.

JACOB, B., NG, S., AND WANG, D. 2008. *Cache, DRAM, Disk*. Morgan Kaufmann.

KEBSCHULL, U. AND ROSENSTIEL, W. 1993. Efficient graph-based computation and manipulation of functional decision diagrams. In *Design Automation, 1993, with the European Event in ASIC Design. Proceedings. [4th] European Conference on*. 278–282.

KNUTH, D. 2009. *The Art of Computer Programming Volume 4, Fascicle 1: Bitwise Tricks & Techniques; Binary Decision Diagrams*. Addison-Wesley Professional.

LIAW, H.-T. AND LIN, C.-S. 1992. On the OBDD-representation of general Boolean functions. *IEEE Transactions on Computers*.

MINATO, S. 1993. Zero-Suppressed BDDs for Set Manipulation in Combinatorial Problems. In *ACM/IEEE 30th Design Automation Conference (DAC)*. 272–277.

MINATO, S. 2010. Data Mining Using Binary Decision Diagrams. In *Progress in Representation of Discrete Functions*. Morgan & Claypool, Chapter 5, 97–109.

MINATO, S. 2013. Techniques of bdd/zdd: Brief history and recent activity. *IEICE Transactions 96-D,* 7, 1419–1429.

MINATO, S. AND KIMIHITO, I. 2007. Symmetric item set mining method using zero-suppressed bdds and application to biological data. *Information and Media Technologies 2,* 1, 300–308.

MISHCHENKO, A. 2001. An introduction to zero-suppressed binary decision diagrams. In *in Proceedings of the 12th Symposium on the Integration of Symbolic Computation and Mechanized Reasoning*.

REQUENO, J. I. AND COLOM, J. M. 2012. Compact representation of biological sequences using set decision diagrams. In *6th International Conference on Practical Applications of Computational Biology & Bioinformatics*, M. P. Rocha, N. Luscombe, F. Fdez-Riverola, and J. M. C. Rodrguez, Eds. Advances in Intelligent and Soft Computing, vol. 154. Springer Berlin Heidelberg, 231–239.

SASAO, T. AND BUTLER, J. 2014. *Applications of Zero-Suppressed Decision Diagrams*. Morgan & Claypool.

TAYLOR, D. 1990. Error models for robust storage structures. In *20th International Symposium on Fault-Tolerant Computing*.

YANG, S. 1991. Logic Synthesis and Optimization Benchmarks User Guide Version 3.0. User guide, Microelectronic Center.

YOON, S., NARDINI, C., BENINI, L., AND DE MICHELI, G. 2005. Discovering coherent biclusters from gene expression data using zero-suppressed binary decision diagrams. *IEEE/ACM Trans. Comput. Biol. Bioinformatics 2,* 4 (Oct.), 339–354.