**Minimal Cut Set/Sequence Generation for Dynamic Fault Trees**

Zhihua Tang, University of Virginia, Charlottesville  
Joanne Bechta Dugan, University of Virginia, Charlottesville

Key Words: Minimal Cut Set/Sequence, Binary Decision Diagram, Zero-suppressed BDD, Static Fault Tree, Dynamic Fault Tree.

**SUMMARY & CONCLUSIONS**

This paper proposes a zero-suppressed binary decision diagrams (ZBDD) based solution for minimal cut set/sequence (MCS) generation of dynamic fault trees. ZBDD is an efficient data structure for combinational set representation and manipulation. Our solution is based on the basic ZBDD set manipulations (union, intersection, difference and product). Due to the nature of the ZBDD, our algorithm is more efficient than the algorithms based on the BDD, both in computation time and memory usage. In our solution, we also extend the concept of minimal cut set in static fault trees into minimal cut sequence (also with notation MCS) in dynamic fault trees. The minimal cut sequence generation is based on minimal cut set generation. It is also efficient compared with Markov model based methods. As an example, we apply our method to X2000 avionics architecture. The system is modeled using a dynamic fault tree and the minimal cut sets/sequences are generated and analyzed.

1. **INTRODUCTION**

Acronyms

<table>
<thead>
<tr>
<th>Acronym</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>BDD</td>
<td>Binary Decision Diagram</td>
</tr>
<tr>
<td>DFT</td>
<td>Dynamic Fault Tree</td>
</tr>
<tr>
<td>FDEP</td>
<td>Functional Dependent Gate</td>
</tr>
<tr>
<td>FT</td>
<td>Fault Tree</td>
</tr>
<tr>
<td>HSP, WSP, CSP</td>
<td>Hot, Warm, Cold Spare Gate</td>
</tr>
<tr>
<td>MCS</td>
<td>Minimal Cut Set/Sequence</td>
</tr>
<tr>
<td>PAND</td>
<td>Priority AND Gate</td>
</tr>
<tr>
<td>SEQ</td>
<td>Sequence Gate</td>
</tr>
<tr>
<td>ZBDD</td>
<td>Zero-suppressed BDD</td>
</tr>
</tbody>
</table>

The fault tree is one of the most commonly used models for dependability analysis of fault-tolerant systems. Since the first use of the fault tree model in 1960s (Ref. 1), many different methods (e.g. BDD (Ref. 2) and Markov Model (Ref. 3)) have been developed for the evaluation of fault trees.

Among various fault tree analysis methods, MCS analysis plays a central role in the assessment of fault trees. MCS are the basis for qualitative analysis since they represent the minimal sets of component failures that cause the failure of the whole system. MCS can also be used in the quantitative analysis which determines the probability of occurrence of the top event of a fault tree. Related probabilistic computations based on MCS appear in Refs.4, 5, 6.

The main difficulty in MCS-based fault tree analysis is finding the MCS of fault trees. The MCS may be generated by applying a top-down algorithm (Ref. 4), or by traversing the BDD representation of a fault tree (Refs. 7, 8, 9). Since a MCS is a minimal set of basic events of a fault tree, any computation of MCS involves combinational set representation and manipulation, which may be computationally intensive and require large amount of storage.

Dynamic fault trees provide more flexible and more powerful modeling in the analysis of dynamic fault-tolerant systems by introducing dynamic gates into traditional fault trees. The top event of a dynamic fault tree depends not only on the combinations of failure events but also on the failure sequences among basic events. In practice, people may ask the same questions in dynamic fault trees as in the static fault trees, such as, “what are the minimal sets of basic events occurring in a certain order that will cause the failure of a system?” and “how can we find those basic event sets in an efficient way?”

The purpose of this paper is to propose an efficient solution for MCS generation in dynamic fault trees. First we introduce an efficient data structure for MCS representation and manipulation. Then we present our MCS generation algorithm for static fault trees. By extending the concept of minimal cut set in static fault trees to minimal cut sequence in dynamic fault trees, we also expand our MCS generation algorithm to deal with the minimal cut sequence generation. Finally, the MCS generation of a real fault-tolerant computer system, X2000 avionics architecture (Ref. 10), is illustrated.

2. **PRELIMINARY CONCEPTS**

2.1 **BDD**

A Binary Decision Diagram (BDD) is a directed acyclic graph representation of a Boolean function. A BDD has two terminal nodes: 0 and 1 encoding the two corresponding constant functions. Each internal node has two edges, which are called 0-edge and 1-edge. A BDD is derived by reducing a binary decision tree, which represents the recursive execution of Shannon’s decomposition. The following reduction rules are used in the construction of a BDD from a binary decision tree.

1. Delete all the redundant nodes whose two edges point to the same node.
2. Merge all the isomorphic subgraphs.

With this two reduction rules, a BDD can represent a Boolean function efficiently. For details about BDD, the reader is referred to two papers by Bryant (Refs. 11, 12).

---

1The singular & plural of an acronym are always spelled the same.
2.2 ZBDD

In order to represent combination sets efficiently, Minato (Refs. 13, 14) proposes the following reduction rules for BDD.

1. Delete all the nodes whose 1-edge points to the 0-terminal node. Then connect the edge to the other subgraph directly, as shown in Fig. 1.
2. Share all isomorphic subgraphs.

The resulting BDD which are based on the above rules are called Zero-suppressed BDD (ZBDD).

![Fig.1: New reduction rule on ZBDD]

2.3 Combination Set and MCS

A combination on n components can be represented by an n bit binary vector, \((x_n x_{n-1} \ldots x_2 x_1)\), where each bit, \(x_k \in \{0, 1\}\), indicates whether or not the component is present in the combination. A set of combinations can be represented by a set of the n bit binary vectors. Such sets are called combination sets.

Combination sets can describe solutions to combinatorial problems. A kind of typical qualitative fault tree analysis is to determine all the possible combinations of basic events which can cause the occurrence of the top event. Such combinations of basic events are called cut sets.

Combination sets can describe solutions to combinatorial problems. A kind of typical qualitative fault tree analysis is to determine all the possible combinations of basic events which can cause the occurrence of the top event. Such combinations of basic events are called cut sets.

Informally, a MCS can be defined as a minimal set of basic events of the fault tree that induces its top event. In terms of combination set, a MCS is a minimal combination of basic events that induces the top event. Therefore, the MCS generation is a kind of combinatorial problems, which can be solved by manipulating combination sets.

3. MCS GENERATION OF STATIC FT

3.1 MCS Representations

MCS are sets of combinations in nature. A combination of n components can be represented by an n bit vector. Each bit has value of “1” or “0” which indicates whether one component is included in that combination or not. Furthermore, a set of combination can be mapped into Boolean space by using n-input variables for each bit of the combination vector. A set of combinations then can be represented by a Boolean function, which is called a characteristic function. Such characteristic function can be used to determine whether a given combination vector is included in a set of combinations.

To represent and manipulate MCS in computer is computing time and memory consuming. One characteristic function can represent a set of combinations. It is very efficient to represent and manipulate MCS by using the characteristic function. However, we need to find an efficient data structure for characteristic function representation and manipulation. One choice is the BDD, since characteristic function is a Boolean function in nature. In a BDD representation, the paths from the root node to 1-terminal node, which are called 1-paths, represent possible combinations in the set. As claimed in Refs. 15, 16, BDD can be used to encode characteristic functions and manipulate combinatorial sets efficiently.

However, BDD representation only considers the Boolean property of characteristic functions. It overlooks the combinatorial property. The BDD representation will generate many useless nodes when it is applied to very sparse combinations. This drawback comes from the reduction rules used in BDD. In combination sets, irrelevant variables are set to default value “0”, which means they never appear in any combination. Unfortunately, those irrelevant variables can not be deleted in the BDD representation, since the irrelevant variables in the BDD representation could be “1” or “0”.

The ZBDD representation of combination sets can solve the problem in the BDD representation. In a ZBDD representation, each 1-path also represents one possible combination. The reduction rule in a ZBDD deletes all nodes whose 1-edges point to the 0-terminal node. This rule suppresses all irrelevant variables in ZBDD representation of combination sets. In ZBDD, the default value of an irrelevant variable is set to zero.

Fig.2 illustrates the BDD and ZBDD representation of a combination set. The input variables are “a”, “b” and “c”. This example shows that the ZBDD may use fewer nodes to represent combination sets. In the following sections, we will illustrate how to generate MCS for static fault trees based on ZBDD representation.

![Fig.2: BDD & ZBDD representation of combination set \{a, b\}]

3.2 Basic Set Operations Used in MCS Generation

The basic set operations used in MCS generation deal with set union, set intersection, set difference and set product. Just like the manipulation of a BDD, a ZBDD can be recursively constructed from the trivial ZBDD by applying the basic set operations.
operations. The details of the algorithms of the basic set operations can be found in Ref. 13, 14. We list the brief description of basic operations used in MCS generation as follows:

- Union \((P, Q)\) returns \((P \cup Q)\);
- Intesec \((P, Q)\) returns \((P \cap Q)\);
- Diff \((P, Q)\) returns \((P - Q)\);
- Product \((P, Q)\) returns \((P \times Q)\).

(Returns the elements in \(P\) not in \(Q\)).

For example, if \(P = \{a, b\}\), \(Q = \{b, c\}\), then we have

- Union \((P, Q)\) = \(\{a, b, c\}\);
- Intesec \((P, Q)\) = \(\{b\}\);
- Diff \((P, Q)\) = \(\{a\}\);
- Product \((P, Q)\) = \(\{ab, ac, b, bc\}\).

These operations have an exponential time for the number of variables in the worst case; however, these operations can be more efficient by using hash table and computing table as used in the manipulation of BDD (Ref. 12). By referring the hash table and computing table, we can avoid performing the same operation and generating the same ZBDD nodes. With this improvement, these operations can be performed in a time that is almost proportional to the size of graphs.

3.3. MCS Generation for Static Fault Trees

A static fault tree consists of only static gates, including AND gate, OR gate and \(k\)-out-of-\(n\) gate (Refs. 3, 4). Since \(k\)-out-of-\(n\) gate can be expanded into the combination of the OR and AND expansions, we can consider that a static fault tree consists of only AND gates and OR gates. Our MCS generation algorithm for static fault trees is structured based on this static fault tree characteristic. Our algorithm includes three functional modules: MCS generation for basic events, MCS generation for AND connections and MCS generation for OR connections.

3.3.1. MCS Generation for Basic Events.

The MCS of a basic event is obviously the combination that comprises that basic event itself only. In ZBDD representation, the MCS of a basic event is a single ZBDD internal node, as shown in Fig.3.

![Fig.3: ZBDD representation of the MCS of a basic event](image)

3.3.2. MCS Generation for AND/OR Connection

The MCS generation algorithm is executed recursively during the depth-first left-most traversal of a fault tree. The algorithm first generates the MCS (encoded in ZBDD) of the inputs of a connection gate, and then performs a serial of set operations to combine the MCS of the inputs into the MCS of the output of the connection gate. Fig.4. shows the MCS generation of an OR connection. The operation MCS-OR combines the MCS (encoded in ZBDD) of the inputs of the OR gate. For an AND connection, the procedure is similar. The operation MCS-AND is performed to generate the MCS of the output of an AND gate.

![Fig.4: Generate the MCS from an OR gate](image)

The MCS-OR and MCS-AND are two key operations in our MCS generation algorithm. They generate a new MCS from several MCS according to the connection logic. They combine the input MCS using OR or AND logic by performing set union or set product operation. At the same time, reduction is taken to reduce the combined cut sets to MCS.

- **MCS-OR Operation**

If the top gate of a fault tree is an OR gate, then the cut sets of this fault tree are the union of the cut sets of the input subtrees of the OR gate. Following this fact, we can design MCS-OR operation based on set union operation. Since the set union of two or more MCS may not be minimal, we need extra reduction to get the MCS. Fig.5. shows the MCS-OR operation on two MCS inputs.

![Fig.5: MCS-OR operation](image)

Example: Let \(S_1 = \{a, b, c\}\), \(S_2 = \{b, d\}\). It is easy to generate \(MCS(S_1 \cup S_2) = \{a, b, d\}\) by hand. By applying MCS-OR operation, the MCS is generated as follows:
\[ S_c = S_1 \cap S_2 = \{ b \}; \quad D_1 = S_1 - S_c = \{ a, cd \}; \]
\[ D_2 = S_2 - S_c = \{ d \}; \quad U = D_1 \cup D_2 = \{ a, cd, d \}; \]
\[ P = D_1 \ast D_2 = \{ ad, cd \}; \quad D_3 = U - P = \{ a, d \}; \]
\[ MCS = S_c \cup D_3 = \{ a, b, d \}. \]

- **MCS-AND Operation**

If the top gate of a fault tree is an AND gate, then the cut sets of this fault tree are the product of the cut sets of the input subtrees of the AND gate. To get the minimal result, we need to do some reduction. The reduction is based on the fact: the product of a combination and MCS is still minimal if the combination does not appear in the given MCS. For example, let, \( M = \{ ab, c \}, C_1 = \{ e \}, \) and \( C_2 = \{ e \}. \)

\[ M \ast C_1 = \{ abc, c \} \text{ is not a MCS, while } M \ast C_2 = \{ abc, ce \} \text{ is a MCS. Fig.6 shows the MCS-AND operation.} \]

![MCS-AND Operation Diagram](image)

**Example:** Let \( S_1 = \{ a, b, cd \}; S_2 = \{ b, d \}. \) it is easy to get \( MCS(S_1 \text{ AND } S_2) = \{ b, ad, cd \} \) by hand. By applying MCS-AND operation, the MCS is generated as follows:

\[ S_c = S_1 \cap S_2 = \{ b \}; \quad D_1 = S_1 - S_c = \{ a, cd \}; \]
\[ D_2 = S_2 - S_c = \{ d \}; \quad P = D_1 \ast D_2 = \{ ad, cd \}; \]
\[ MCS = S_c \cup P = \{ b, ad, cd \}. \]

### 4. MCS GENERATION OF DFT

In a dynamic fault tree, dynamic gates, such as spare gates (HSP, WSP and CSP), sequence gate (SEQ), priority AND gate (PAND) and functional dependent gate (FDEP) (Ref. 7) may be present. The occurrence of the top event in a dynamic fault tree depends not only on the combination of the basic events, but also on the occurring order of the basic events. Therefore, the components in minimal cut sets are order sensitive. In order to reflect the order issue in DFT, we extend the concept of the "minimal cut set" in static fault trees to "minimal cut sequence". A "minimal cut sequence" is the minimal failure sequence that causes the occurrence of the top event of a DFT. The basic events in a minimal cut sequence should occur in the order that they appear in the minimal cut sequence. In this paper, the notation MCS can be "minimal cut set" or "minimal cut sequence". It depends on which kind of fault trees it is applied to.

DFT are usually evaluated using Markov model. We can generate the MCS of a DFT by extracting the failure sequences from the Markov chain of the DFT. However, to build the Markov model and find out the MCS from the Markov model are considerably time-consuming. Considering the relationship between the minimal cut set and minimal cut sequence, it is possible to provide a solution that more efficient than the solution based on Markov model.

The dynamic constraints of dynamic gates can be decomposed into logic constraints and timing constraints. The logic constraints specify the “AND” or “OR” relation among the failed inputs of a dynamic gate. The timing constraints specify the required failure sequences among the failed inputs of a dynamic gate. For instance, the logic constraint of a “PAND” gate is “AND” logic on all inputs. The timing constraint of a “PAND” gate is its inputs should fail in the order of “from left to right”. Based on this analysis of the dynamic constraints, we design the MCS generation algorithm for DFT as follows:

1. Replace dynamic gates in a DFT with the static gates corresponding to their logic constraints.
2. Generate the minimal cut sets of the resulting static fault tree.
3. Expand each minimal cut set to minimal cut sequences by considering the timing constraints.

In the step (3), we can first partition the minimal cut set into static subset and dynamic subset. The static subset consists of the inputs that are directly or indirectly connected to static gates only. The dynamic subset consists of the inputs that are directly or indirectly connected to any dynamic gate. We only need to expand the dynamic subset to set of failure sequences. Moreover, the implementation of the step (2) can shared the MCS generation of static fault trees, which is based on ZBDD. Therefore, the performance of our solution is expected to be better than that of Markov model based solution.

### 5. EXAMPLE

This section applies our solution to a fault-tolerant COTS-based bus architecture – X2000 avionics system architecture. The example is adapted from Ref. 10. We first build the fault tree model for the example system, and then the MCS are generated and analyzed.

Fig.7 shows the X2000 architecture. The architecture is a distributed, symmetric system of multiple computing-nodes and device-controllers that share common redundant bus architecture. The redundant bus consists of two different bus types – the IEEE 1394 bus and the I²C bus. Their different topologies and performance provide the design-diversity in X2000 system.
A node in X2000 architecture can be a flight computer, a global non-volatile mass memory, a subsystem microcontroller or a science instrument. A failed node may partition the bus network and cut off communication between the sub-networks. In X2000 architecture, the entire set of IEEE 1394 and I²C buses are duplicated to provide redundancy for fault recovery. In normal operation, only one set of the buses is activated. If one of the buses in the primary bus-set fails, the backup set of buses will be activated, and the system operations are transferred to the backup buses. In any bus-set, the two buses can help each other to isolate and recover the failed nodes and maintaining the communication of all nodes.

Fig. 8 shows the fault tree model for X2000 system. The top event is that the network is cut off due to the failure of bus set or computer nodes. For simplification, it is assumed that the PCI and other bus in subsystems of the computer nodes are reliable enough. The MCS of X2000 system are generated as listed in Table 1. They are sorted by the order (the number of components) ascendingly.
The probability of each MCS is the product of failure probabilities of all basic events in the MCS. In this example, all basic events are assigned an exponential distribution with 0.02 failure rate. The mission time is 100 hours. The probabilities of MCS can be used in the approximate quantitative reliability analysis. In the MCS result, the first two MCS are minimal cut sequences. The symbol “→” indicates there is failure order constraint between two basic events.

<table>
<thead>
<tr>
<th>Order</th>
<th>Probability</th>
<th>Minimal Cut Set/Sequence</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>0.748</td>
<td>[BS1→BS2]</td>
</tr>
<tr>
<td>2</td>
<td>0.748</td>
<td>[BS2→BS1]</td>
</tr>
<tr>
<td>6</td>
<td>0.418</td>
<td>[MC2, MC3, MC1, IOI, GMM, SI]</td>
</tr>
<tr>
<td>6</td>
<td>0.418</td>
<td>[MC2, MC3, MC1, NVM1, MC4, SI]</td>
</tr>
<tr>
<td>6</td>
<td>0.418</td>
<td>[MC2, MC3, MC1, FC, MC4, SI]</td>
</tr>
<tr>
<td>6</td>
<td>0.418</td>
<td>[MC2, MC3, TEL, IOI, MC4, SI]</td>
</tr>
<tr>
<td>6</td>
<td>0.418</td>
<td>[MC2, MC3, TEL, IOI, GMM, SI]</td>
</tr>
<tr>
<td>6</td>
<td>0.418</td>
<td>[MC2, MC3, TEL, IOI, MC4, SI]</td>
</tr>
<tr>
<td>7</td>
<td>0.361</td>
<td>[LSS, HSS, MC3, MC1, IOI, GMM, SI]</td>
</tr>
<tr>
<td>7</td>
<td>0.361</td>
<td>[LSS, HSS, MC3, MC1, NVM1, MC4, SI]</td>
</tr>
<tr>
<td>7</td>
<td>0.361</td>
<td>[LSS, HSS, MC3, MC1, FC, MC4, SI]</td>
</tr>
<tr>
<td>7</td>
<td>0.361</td>
<td>[LSS, HSS, MC3, TEL, IOI, MC4, SI]</td>
</tr>
<tr>
<td>7</td>
<td>0.361</td>
<td>[LSS, HSS, MC3, TEL, IOI, GMM, SI]</td>
</tr>
<tr>
<td>7</td>
<td>0.361</td>
<td>[LSS, HSS, MC3, TEL, IOI, MC4, SI]</td>
</tr>
</tbody>
</table>

**ACKNOWLEDGMENTS**

We would like to thank NASA Langley Research Center under NASA Contract NAS1-02076, by which the work reported in this paper was funded.

**REFERENCES**


BIOGRAPHIES

Zhihua Tang
Department of Electrical & Computer Engineering
351 McCormick Road; PO Box 400743
University of Virginia
Charlottesville VA 22904-4743 USA

e-mail: zt9m@virginia.edu

Zhihua Tang received his B.S. degree in Applied Physics from Northeastern University, Shenyang, China, in 1996, the M.S. degree in Computer Science from Shenyang Institute of Computing Technology, Chinese Academy of Science, China, in 1999, and the M.S. degree in Electrical Engineering from University of Virginia, Charlottesville, VA in 2002. He is now a Ph.D. student in the Department of Electrical & Computer Engineering at the University of Virginia. He is a student member of IEEE.

Joanne Bechta Dugan, Ph.D.
Department of Electrical & Computer Engineering
University of Virginia
351 McCormick Road PO Box 400743
Charlottesville, Virginia 22904-4743 USA

e-mail: jbd@virginia.edu

Joanne Bechta Dugan was awarded the B.A. degree in Mathematics and Computer Science from La Salle University, Philadelphia, PA in 1980, and the M.S. and Ph.D. degrees in Electrical Engineering from Duke University, Durham, NC in 1982 and 1984, respectively. Dr. Dugan is currently Professor of Electrical and Computer Engineering at the University of Virginia. She has performed and directed research on the development and application of techniques for the analysis of computer systems that are designed to tolerate hardware and software faults. Her research interests include hardware and software reliability engineering, fault tolerant computing, and mathematical modeling using dynamic fault trees, Markov models, Petri nets, and simulation. Professor Dugan is a member of Phi Beta Kappa, Eta Kappa Nu, Tau Beta Pi and IEEE; is an IEEE Fellow; was Associate Editor of the IEEE Transactions on Reliability for 10 years; and is currently Associate Editor of the IEEE Transactions on Software Engineering. She is a past winner of both the P.K. McElroy and the Alan O. Plait Awards.