A Modular Approach for Analyzing Static and Dynamic Fault Trees

Rohit Gulati; Alta Group of Cadence Design Systems, Sunnyvale
Joanne Bechta Dugan; University of Virginia, Charlottesville

Key Words: Static and Dynamic Fault trees, Markov chain, Binary Decision Diagrams, Modularization, Reliability Analysis

SUMMARY AND CONCLUSIONS

Three commonly used analytical techniques for reliability evaluation are fault trees, Binary decision diagrams and Markov chains. Each of these techniques have advantages and disadvantages and the choice depends on the system being modeled. Fault trees have been found to be the most popular choice in terms of building an analytical model of a system. It provides a compact representation of the system and is easily understood by humans. However, fault trees lack the modeling power and its solution time increases exponentially with the size of the system being modeled.

In this paper, we present a new exciting hybrid approach, called the modular approach, for the efficient analysis of both static and dynamic fault trees. It provides a combination of BDD solution for static fault trees and Markov chain solution for dynamic fault trees coupled with the detection of independent subtrees. The algorithms used for modularization, integrating the results obtained from the separate solution of the independent modules (subtrees) and incorporating coverage modeling are discussed in detail in this paper. The modular approach is applied to an example systems to demonstrate the potential of this research

1 INTRODUCTION

Fault Trees have been widely used for reliability analysis for about forty years. They were first developed in the 1960s to facilitate analysis of the Minuteman missile system [21], and have been supported by a rich body of research since their inception. It provides a compact representation of the system and can be easily specified by humans. However, the analysis of large fault trees is a tedious process even though considerable progress has been made in this field. Bryant's Binary Decision Diagram (BDD) [5] provides a faster and more efficient means for analyzing such large fault trees. But the use of BDD is limited to the analysis of static fault trees only. Dynamic fault trees, which are used for modeling sequence dependencies, can only be solved using Markov solutions methods. The disadvantage of Markov models and BDDs, is that its direct specification is an awkward, time consuming and an error prone task.

Most of the tools developed thus far, for reliability analysis are based on fault trees which are solved by converting to either its equivalent BDD [1][7][8] or to Markov model [9]. Tools that convert fault trees to BDD for solution can not be used to model sequence dependencies and hence, can not provide dynamic fault tree solutions. But, they can solve large fault trees in little time because there is no relation between the number of components of a system and the size of the BDD that represents it. It is not uncommon to find large fault trees. For example the fault tree for the newly designed space station contains approximately 1600 basic events [17].

Tools that allow dynamic fault tree analysis by converting the fault tree to a Markov chain can not solve large fault trees because the size of the resulting Markov model increases exponentially and so does the solution time. Moreover, such tools convert the entire fault tree to a Markov model irrespective of the fact whether the fault tree has any dynamic gates. A dynamic fault tree can have static and dynamic gates, hence, conversion of a fault tree containing only static gates to a Markov model can be very inefficient.

A more efficient and powerful means for fault tree analysis can be provided by the use of both BDD and Markov solution methods in tandem. *DIFtree*, a fault tree analysis tool developed at the University of Virginia [10], uses a modular approach which provides a BDD solution for static fault trees and Markov chain solution for dynamic fault trees. A fault tree is converted to a Markov chain only if it has one or more dynamic gates. If the fault tree has only static gates it is solved by converting to a BDD.

Often a very small part of the entire fault tree is dynamic in nature. Hence, independent subtrees are identified in the modular approach and the decision to use a Markov solution or BDD solution is made for the subtree instead of for the fault tree as a whole. These independent subtrees are treated separately and their solutions are integrated to get the solution for the entire fault tree. In other words, several Markov models and BDDs are used for the solution of a fault tree instead of one

0-7803-3783-2/97/$5.00 © 1997 IEEE

Markov model or one BDD. The advantage of using several Markov models and BDDs is that each will be substantially smaller in size than the single Markov model or BDD. It is easy to handle three independent Markov models with 1000 states each compared to one Markov model with 10^9 states.

The paper is organized as follows. The next section provides a brief overview of the *DIFtree* methodology. Then the modular approach is introduced and explained using an example fault tree in section 3. In sections 4, 5 and 6 the algorithms used for modularization and synthesis of results are discussed. The way coverage modeling is incorporated in the modular approach is discussed in section 7. Finally the modular approach is applied to the fault tree of a real life system (ASID-MAS), to demonstrate the potential of this research.

2 *DIFtree METHODOLOGY*

Figure 1 provides an overview of the *DIFtree* methodology. The system level fault tree can be described either graphically or textually. The Graphical user interface (GUI) can automatically generate the textual format and can automatically display a fault tree from the textual format. In other words, the GUI provides the ability to switch between the textual format and the graphical display. The textual input language relies on the use of keywords to linearly describe the fault tree connections. Once the fault tree is described it can be solved using the modular approach, explained in the next section.

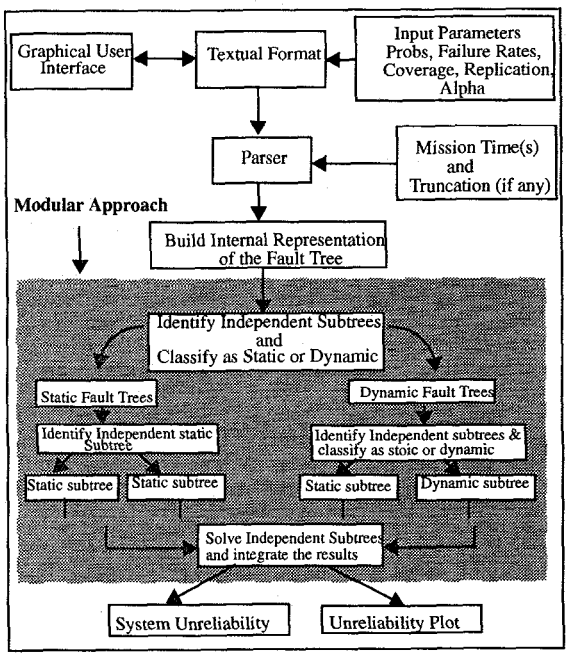

Figure 1 *DIFtree* Methodology

3 *MODULAR APPROACH*

In the modular approach (shaded part in Figure 1) the fault tree is divided into independent subtrees, and different solution techniques are applied to each subtree depending on the latter's characteristics. Independent subtrees (subtrees that share no inputs) are found using a very efficient linear time algorithm [13]. The subtrees are further identified as static or dynamic. If the subtree is static in nature, then the BDD solution is used. If the subtree is dynamic in nature, then the Markov chain solution is used. Modularization is a recursive process as subtrees might themselves contain independent subtrees. The solutions of various independent subtrees are integrated using a relatively straightforward and recursive algorithm.

As an example, modular approach is applied to the fault tree in Figure 2. There are two independent static subtrees nested in a larger static subtree and one dynamic subtree. Markov model for the dynamic subtree and the BDD model for one of the static subtree is shown in the figure. Modularization techniques are not applied to the solution of a dynamic subtree whose toplevel node is a dynamic gate because it does not provide an exact solution. Thus, independent subtrees are not identified for the dynamic subtree in Figure 2 because the toplevel node is a Priority-AND gate [14].

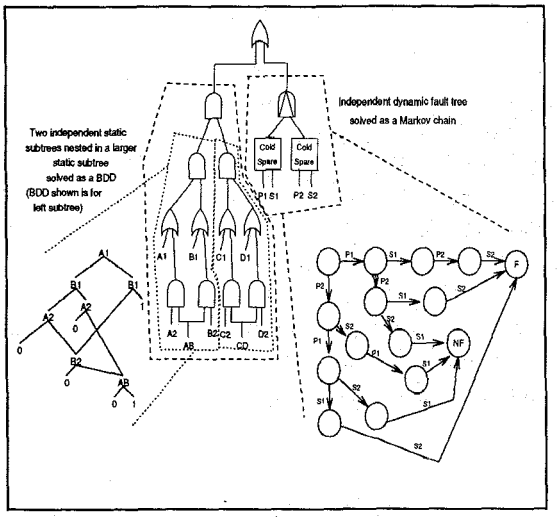

Figure 2 An Example of Modular Approach [13]

The basic idea in integrating the results obtained from the separate solution of the independent subtrees is to traverse the fault tree, *depth-first-left-most* starting at the toplevel node. Then determine for each gate node, if there are any independent subtrees below it. If there are, then traversal continues to the inputs of the node. If there are no independent subtrees below the gate node and if the gate node is independent (is the topnode of an independent subtree), it is solved using the appropriate solution technique depending on the type of the subtree it represents. It is then replaced in the parent node as a basic event with the probability of failure equal to that obtained from the

solution of the subtree. This is equivalent to replacing the entire subtree in the fault tree by a basic event. The traversal then returns to the parent node and continues from there.

Once all the independent subtrees below a gate node have been solved and replaced by a basic event, the gate node is checked for independence. If it is independent, it is solved using the technique depending on the type of subtree it represents. It is then replaced as a basic event in its parent node. Hence, recursively the solutions of various independent subtrees are integrated to get the failure probability of the toplevel node.

For a static subtree, the probability of occurrence of the top event of the subtree is the probability of the basic event replacing the subtree. On the other hand, for the dynamic subtree, the probability of reaching the failure state of the resulting Markov chain is the probability of the basic event which replaces the subtree.

4 IDENTIFICATION OF MODULES

The most important thing in modularization is to find an efficient way of identifying independent subtrees (modules). A number of methods [3][6][15][16][18][19][20][22] have been proposed to detect modules or independent subtrees in a fault tree. Chatterjee [6] and Birnbaum et al [3] first developed the properties of modules and demonstrated their use in fault tree analysis. Locks [16] expanded the concept to non-coherent fault trees and showed its effectiveness in obtaining cutsets. Rosenthal [18] obtained modules in a fault tree by using an algorithm for finding cut vertices from a fault tree. Wilson [20] obtained modules from the boolean indicator expression of a fault tree.

However, the most efficient and simple algorithm, to date, has been proposed by Rauzy and Dutuit [13]. It detects modules in a fault tree (coherent or not) with several hundred gates and basic events within a few milliseconds. Moreover, its simplicity makes it easier to implement. Hence, the modular approach uses Rauzy's linear time algorithm to detect independent subtrees.

The basic principle of the algorithm can be stated as follows: *Let v be an internal event and t_1 and t_2 respectively the first and second dates of visits of v in a depth first left most traversal of the fault tree. Then v is a module iff none of its descendents is visited before t_1 and after t_2 during the traversal* [13].

5 ALGORITHM FOR MODULARIZATION

Figure 3 shows the algorithm used for modularization in *DIFtree*. It uses Rauzy's algorithm for finding independent subtrees. The fault tree is traversed twice. In the first *depth-first left-most* traversal, the counters are set according to the following rule: in the first visit to a node the First counter for that node is set, and in the second visit the Second counter is set. Further visits to the node increments the Last counter. If the First counter to a node is set then its inputs are not visited.

In the second traversal, for each gate node all its inputs are traversed. The minimum of the *first dates* (value in the first counter) and maximum of the *last dates* of visits of all inputs are collected. If any input is independent then the *indep_below flag* of the gate node is set to TRUE. This flag indicates if there are any independent subtrees below a gate node. If any input to a gate is dynamic and is not independent, then the gate node is also made dynamic.

```
Initialize the counters and variables;
Perform a depth first left most traversal of the tree
starting at toplevel to set the counters;
Perform a second depth first left most traversal of the
tree starting at toplevel node.
For each node x {
    if (x.node_type = BASIC_EVENT)
        return;
    else if (x.node_type = GATE) {
        collect from all its inputs {
                min = minimum of the first dates;
                max = maximum of the last dates;
                if any input is independent
                    x.indep_below = YES;
                if any input is Dynamic and not independent
                    x.tree_type = DYNAMIC;
        }
        Check for independence
            (x is a module iff max < x.second && min > x.first);
        if (x.node_type == DYNAMIC) {
            x.indep_below = NO;
            x.tree_type = DYNAMIC;
        }
        else
            x.tree_type = STATIC;
    }
}
```

Figure 3 Algorithm for modularization in *DIFtree*

After traversing all the inputs of a gate node, thecate node (say *x*) is checked to see if it is a module (or is the toplevel of an independent subtree). It is a module *iff* the collected minimum and maximum are respectively greater than the *first date* of *x* and less than the *second date*. The *tree_type flag* of the gate node is set to the type of subtree it represents. If the *tree_type flag* is dynamic then *indep_below flag* is reset to FALSE, indicating there are no independent subtrees below it. This is done because modularization technique is not applied to the solution of a subtree whose toplevel gate is dynamic.

Results of various independent subtrees are integrated using the two flags, *indep_below* and *tree_type*. Next section describes the integration algorithm.

6 ALGORITHM FOR SYNTHESIS OF RESULT

Figure 4 shows the algorithm used for synthesis of the solution of various independent subtrees. The fault tree is again traversed *depth-first left-most* starting at the toplevel. If the

indep_below flag of a gate node is TRUE, indicating that there are nested independent subtrees below it, then the traversal continues to the inputs of the node, so that the nested subtrees are solved first. If the *indep_below flag* is FALSE and the gate node is independent, it is solved using the appropriate solver depending on the type of subtree the gate node represents. Then the type of the node is changed from *gate_node* to *leaf_node*, the inputs of the node are cleared and the failure probability of the node is set to the result obtained from the solver.

```
Perform a depth first left most traversal of the tree
starting at the toplevel node.
For each node x {
    if (x. node_type == BASIC_EVENT)
        return;
    else if (x.node_type == GATE) {
        if (x.indep_below == YES)
            traverse the inputs;
        if (x.independence == NO)
            return;
        if (x.independence == YES) {
            if (x.tree_type == DYNAMIC)
                failure_prob = dynamic_solver();
            else
                failure_prob = static_solver();
            x.node_type = BASIC_EVENT;
            x.inputs.clear();
            x.prob = failure_prob;
        }
    }
}
```

Figure 4 Algorithm for Synthesis of Results

As an example consider the dynamic fault tree in Figure 2. There are two independent static subtrees nested in a larger static subtree and one dynamic subtree. To integrate the results, the fault tree is traversed *depth-first-left most* starting at toplevel node A. The *indep_below flag* of node A is TRUE because there are independent subtrees below it. The traversal continues to the inputs of node A and its leftmost input B is checked next. The *indep_below flag* of node B is also TRUE because there are nested independent subtrees below it. The traversal continues to the inputs of node B and its leftmost input (node D) is checked first. It represents a static subtree and its *indep_below flag* is FALSE (because there are no independent subtree below it). Hence it is converted to a BDD and solved using a BDD solver. It is then replaced in the parent node (gate node B) as a basic event. The failure probability of this basic events is set to that obtained from the solution of the BDD. The traversal then returns to gate node B and its next input (node E) is checked. It also represents a static subtree and its *indep_below flag* is also FALSE. Hence, it is converted to a BDD and solved. It is then similarly replaced in the parent node (gate node B) as a basic event. The traversal again returns to gate node B. Both the inputs of gate B are now replaced by two basic events. The probabilities of these two basic events are obtained from the separate solutions of the two BDDs.

Node B (with both inputs replaced by basic events) is checked next because all its input nodes have been processed. It also represents an independent static subtree. Hence, It is converted to a BDD, solved and replaced in its parent node (node A) as a basic event. The failure probability of this basic events is set to that obtained from the solution of the BDD. The traversal returns to node A and its next input node C is checked. It represents an independent dynamic subtree. Note that it has two independent subtrees below it but modularization techniques are not used for dynamic gates because it does not provide an exact solution. Hence, the entire subtree is replaced by a basic event. The probability of this basic event is obtained by converting the subtree to a Markov model and solving the resulting Markov chain. This reduces gate A to only two basic events (node B and node C), a BDD solution of which gives the probability of failure of the toplevel node.

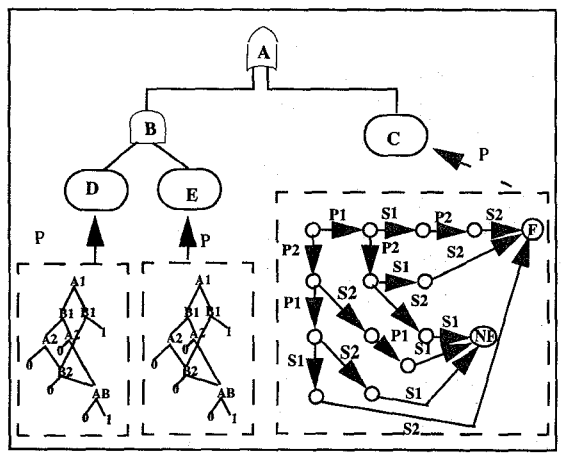

Figure 5 Example of integration of results of various subtrees

If modular approach were not used in the analysis of the dynamic fault tree in Figure 2, then the entire tree would have to be converted to a Markov model [4][9]. Moreover, BDD solution technique would not have been possible as the tree is dynamic. Hence, modular approach is a very powerful means of analyzing large and complex fault trees.

7 MODULARIZATION AND COVERAGE

Fault coverage is the conditional probability *c* that the system recovers from a fault given that a fault has occurred. Including the concept of coverage (and the possibility that recovery may be imperfect) in the system level model is critical to an accurate reliability (and other dependability measures) assessment [12]. Coverage modeling can be automatically

incorporated into both dynamic and static fault tree models. Thus, coverage modeling is an integral part of the modular approach. This section discusses the way coverage modeling is incorporated in the modular approach.

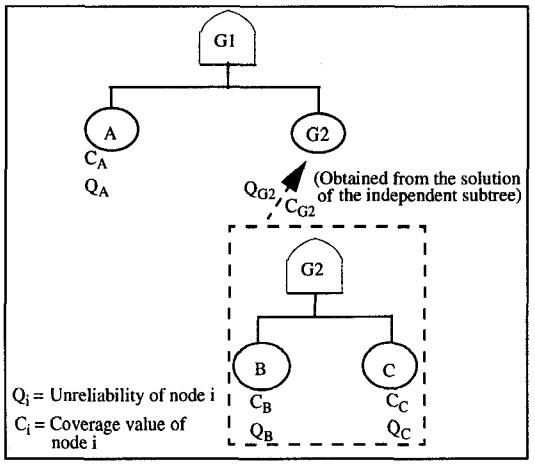

Figure 6 Example to illustrate coverage modeling

In the modular approach, independent subtrees are solved separately and replaced in the fault tree by a basic event. The probability of failure of the basic event replacing an independent subtree, is obtained from the solution of the subtree (BDD solution or Markov solution depending on the type of subtree). In coverage modeling there are two kinds of failures associated with each component: *covered failure* and *uncovered failure.* Solution of the independent subtree provides the total probability of failure associated with the subtree (that is the sum of covered and uncovered failure). For coverage modeling we need to obtain the fraction of the total failure of the independent subtree that is covered (c) and that is uncovered (s) for the basic event replacing the subtree.This is because the covered failure may or may not lead to system failure depending on the remaining redundancy of the system, whereas, an uncovered failure is considered as the single point of failure for the basic event. Hence, the c and s value can have significant impact on reliability calculations.

	Unreliability
Without Modularization	0.117528e-04
With Modularization and without calculating the c and s value for the subtree.	0.686539e-05

Table 1 Results to show the impact of coverage on modularization

For example consider the fault tree in Figure 3.7. The independent subtree, with toplevel gate G2, is replaced in the fault tree by a basic event. The probability of failure and the coverage value for the basic event is obtained from the solution of the subtree. If we do not calculate the c and s value for the independent subtree, then the solution of the fault tree with the independent subtree replaced by a basic event will treat the failure of the basic event (G2) to be fully covered (that is no single point of failure). This will have a very significant impact on the system reliability calculations of the fault tree. Table 1 shows the solution of the fault tree in Figure 6 with modularization (and without the calculation of c and s value for the independent subtree) and without modularization. The unreliability for the fault tree (or the probability of failure of node G1) with modularization (and without the calculation of c and s value for the subtree G2) is significantly underestimated, because the solution assumed that the basic event G2 had perfect coverage (c = 1). Thus, it is important to calculate the c and s value for the independent subtree, and provide it to the basic event that replaces it in the fault tree.

$$C = \frac{\text{P (covered failure of subtree)}}{\text{P (uncovered failure of subtree)} + \text{P (covered failure of subtree)}} \quad (1)$$

Equation 1 is used to calculate the c value for the independent subtree and equation 2 is used to calculate the s value for the subtree.

$$S = \frac{\text{P (uncovered failure of subtree)}}{\text{P (uncovered failure of subtree)} + \text{P (covered failure of subtree)}} \quad (2)$$

8 ASID-MAS EXAMPLE SYSTEM

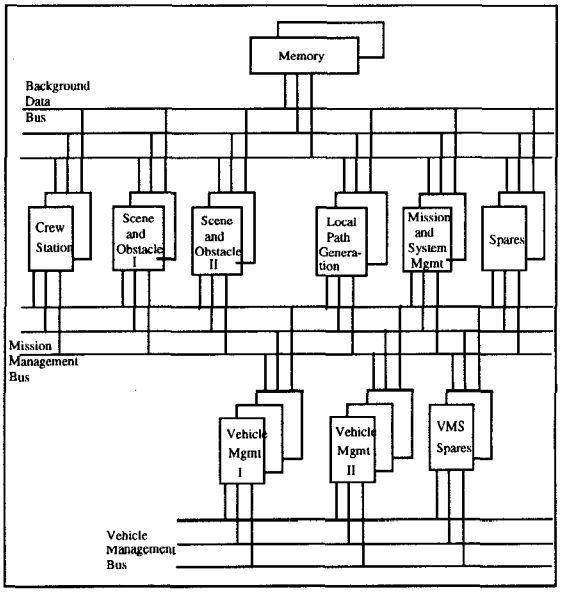

Figure 7 ASID-MAS system architecture[11]

The ASID (Advanced System Integration Demonstration) [2] project was the first large scale effort in the development of the PAVE PILLAR architecture for advanced tactical fighters. In this section we analyze the reliability of the critical functions of the mission avionics subsystem (MAS) of the ASID system. There are several critical functions within the mission avionics system. The loss of any one of these functions causes the system to fail. The critical functions include the vehicle management system (VMS), the crew station control and display functions, mission and systems management, local path generation, and scene and obstacle following functions.

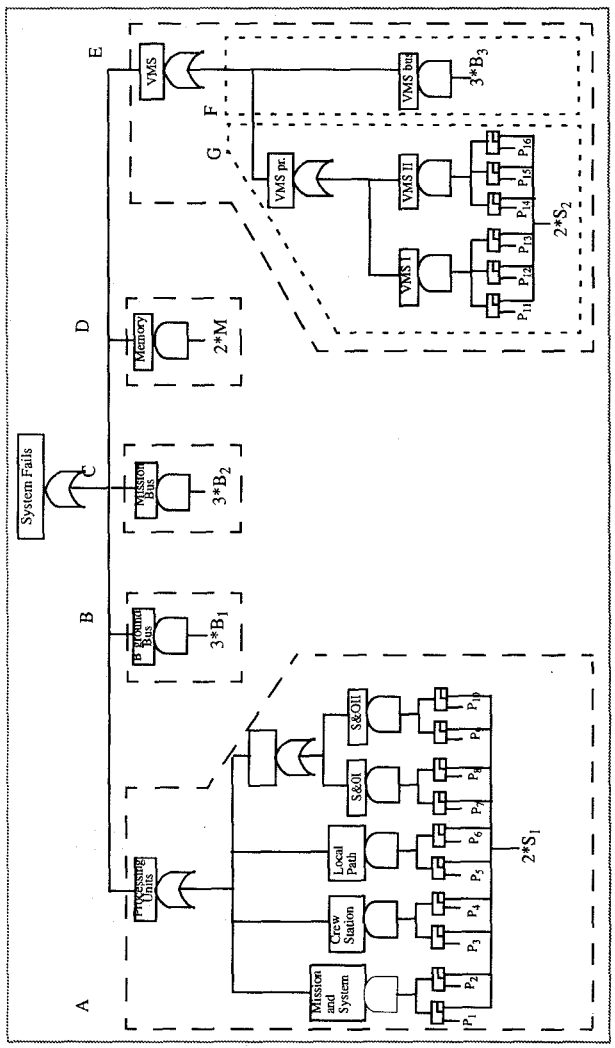

Figure 8 Fault Tree for the ASID-MAS system

Figure 8 shows the fault tree for the ASID-MAS system architecture shown in Figure 7. One processing unit is required for the crew station functions, local path generation, and mission and system management. Each of these processing units is supplied with a hot spare backup to take over control should the primary processor detect an error. For example, mission and system management has P_1 as the primary unit and P_2 as the hot spare unit. The scene and obstacle and VMS subsystems both require more functionality than one processing unit can provide, and thus each use two processing units. The scene and obstacle processing units are also replicated, providing a hot spare backup (P_8 and P_{10}). The VMS system is triplicated, providing two hot spare backups.

In addition to the hot spare backups, 2 additional pools of spares are provided, each containing two spare processing units (S_1 and S_2). S_1 can be used to cover the first two processor failures in the subsystems other than the VMS. S_2 covers the failure in the VMS subsystem. The subsystems are connected via 2 triplicated bus systems, the first being the data bus (B_1) and the second being the mission management bus (B_2). The VMS has an additional triplicated bus, the vehicle management bus (B_3).

The system fails if any one of the functions cannot be performed, or if both of the two memories fail, or if all three of any one type of bus fails. The following failure rates were used in the solution of the fault tree: processors (2.5×10^{-5}(); buses (2.5×10^{-6}) and memories (1.0×10^{-6}). Fault detection was assumed to be perfect.

The fault tree for the ASID-MAS system was solved using the modular approach (results are shown in table 2). Use of modularization resulted in the identification of 7 independent subtrees (labeled A through G, where F and G are nested within E). The independent subtrees were solved separately and the results were integrated to obtain the result for the whole ASID-MAS fault tree. If modular approach was not used for the solution of the fault tree, then the entire tree had to be converted to a Markov model for solution (because of dynamic gates). Markov model for the entire tree would have been too big for any existing commercial tool to be able to solve it (without generating a truncated markov model). Thus modular approach is a very efficient means for solving large and complex fault trees.

Subtree	Unreliability
A	2.12071e-09
B	1.56191e-11
C	1.56191e-11
D	9.99900e-09
E	1.61612e-11
F	1.56191e-11
G	5.42133e-13
ASID-MAS	1.21671e-08

Table 2 Results of the ASID-MAS system

The modular approach, presented in this paper, is useful for solving real world problems many of which manifest themselves in large and complex fault trees. The solution of such large and complex fault trees is a tedious process. There is no known commercially available fault tree package that can provide an exact solution for such systems. Other tools either provide an approximate solution or can solve only small fault trees.

This research was supported by NASA Langley Research Center under grant number NCC-1-210 and Quality Research Associates under grant number QRA-DMI-9460027. Their support is gratefully acknowledged. We plan to incorporate DIFtree as part of the shade tree methodology[17].

REFERENCES

[1] ARALIA Group, "Computation of Prime Implicants of a Fault Tree Within ARALIA," *Proceedings of the ESREL 1995 Conference*, Bournemouth, England, 1995

[2] S.W. Benhen, W.A. Whitehouse, R. J. Farrell, F. M. Leahy, and L. E. Moen, "Advanced system integration demonstration (ASID) system definition," Technical report, AF Wright Aeronautical Laboratories, 1984

[3] Z. W. Birnbaum and J. P. Esary, "Modules of coherent Binary Systems," *SIAM Journal of Applied Mathematics*, 1965, pp 442-462

[4] Mark A. Boyd, "Dynamic Fault Tree Models: Techniques for Analysis of Advanced Fault Tolerant Computer Systems," *Ph.D. Dissertation*, Department of Computer Science, Duke University, 1991

[5] Randal E. Bryant, "Graph-based Algorithms for Boolean Function Manipulation," *IEEE Transactions on Computers*, c-35(8), August 1986, pp 677-691

[6] P. Chatterjee, "Modularization of Fault Trees: A Method to Reduce Cost of Analysis," *Reliability and Fault Tree Analysis, SIAM*, 1975, pp101-137

[7] Oliver Coudert and Jean Christophe Madre, "Fault Tree Analysis: 10^{20} Prime Implicants and Beyond," *Proceedings of the Annual Reliability and Maintainability Symposium*, 1993, pp 240-245

[8] Stacy A. Doyle and J. B. Dugan, "Analyzing Fault Tolerance using DREDD," *In Proceedings of the 10th Computing in Aerospace Conference*, March 1995

[9] J. B. Dugan, K. S. Trivedi, M. K. Sometherman and R. M. Geist, "The Hybrid Automated Reliability Predictor," *AIAA Journal of Guidance, Control and Dynamics*, 9(3), June 1991, pp 554-563

[10] J. B. Dugan, B. Venkataraman, Rohit Gulati, "DIFtree: A software package for the analysis of dynamic fault tree models," *In Proceedings of the Reliability and Maintainability Symposium*, January, 1997

[11] J. B. Dugan, Salvatore J. Bavuso and Mark A. Boyd, "Dynamic Fault Tree models for Fault Tolerant Computer Systems," *IEEE Transactions on Reliability*, Volume 41, Number 3, September 1992, pp 363-377

[12] J. B. Dugan and Kishore S. Trivedi, "Coverage Modeling for Dependability Analysis of Fault Tolerant Systems," *IEEE transactions on Computers*, 38(6), 1989, pp 775-787.

[13] Yves Dutuit and Antoine Rauzy "A linear time Algorithm to find Modules of Fault Trees," *IEEE Transactions on Reliability*, 1996 (to appear)

[14] J. B. Fussel, E.F. Aber and R.G. Rahl, "On the Quantitative Analysis of Priority-AND Failure logic," *IEEE Transactions on Reliability*, Vol. R-25, No. 5, December 1976, pp 324-326

[15] T. Kohda, E. J. Henley and K. Inoue "Finding Modules in Fault Trees," *IEEE Transactions on Reliability*, volume 38, June 1989, pp165-176

[16] M. O. Locks "Modularizing, minimizing and interpreting the K & H fault tree," *IEEE Transactions on Reliability*, Val 40, 1981 Dec, pp411-415

[17] Laura L. Pullum and J. B. Dugan, "Fault Tree Models for the Analysis of Complex Computer Based Systems," *Proceedings of the Annual Reliability and Maintainability Symposium*, 1996, pp 200-207

[18] A. Rosenthal, "Decomposition methods for Fault Tree Analysis," *IEEE Transactions on Reliability*, vol 43, 1980 Jun, pp136-138

[19] K. D. Russell and M. Rasmuson Dale, "Fault Tree Reduction and Quantification - an Overview of IRRAS Algorithm," *Reliability Engineering and System Safety*, 40, 1993, pp 149-164

[20] J. M. Wilson "Modularizing and Minimizing Fault Trees," *IEEE Transactions on Reliability*, R-34, 1985 Oct, pp 320-322

[21] H. A. Watson and Bell Telephone Laboratories, "Launch Control Safety Study," Bell Telephone Laboratories, Murray Hill, NJ USA, 1961.

[22] J. Yllera "Modularization methods for evaluating fault trees of complex systems," *Engineering Risk and Hazard Assessment*, vol. 2, chapter 5, CRC

BIOGRAPHIES

Rohit Gulati
Alta Group of Cadence Design Systems
555, N. Mathilda Avenue
Sunnyvale, CA 94086
rgulati@altagroup.com

Rohit Gulati was awarded the B. E. degree in Electronics and Communication Engineering from Birla Institute of Technology, India in 1993 and the MS degree in Electrical Engineering from the University of Virginia in 1996. Rohit is currently working in the R&D wing of the Alta Group of Cadence Design Systems. His research interests include reliability and software engineering. He is a member of the IEEE computer and reliability societies and the Eta Kappa Nu.

Joanne Bechta Dugan
Dept. of Electrical Engineering
Thornton Hall
University of Virginia
Charlottesville, VA 22903
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 Associate Professor of Electrical Engineering at the University of Virginia, and was previously Associate Professor of Computer Science at Duke University and Visiting Scientist at the Research Triangle Institute. She has performed and directed research on the development and application of techniques for the analysis of computer systems which are designed to tolerate hardware and software faults. Her research interests thus include hardware and software reliability engineering, fault tolerant computing, and mathematical modeling using dynamic fault trees, Markov models, Petri nets and simulation. Dr. Dugan is an Associate Editor of the IEEE Transactions on Reliability, is a Senior member of the IEEE, and is a member of Eta Kappa Nu and Phi Beta Kappa. She is serving on the National Research Council Committee on Application of Digital Instrumentation and Control Systems to Nuclear Power Plant Operations and Safety.