

# DIFtree: A software package for the analysis of dynamic fault tree models

Joanne Bechta Dugan □ University of Virginia □ Charlottesville  
Bharath Venkataraman □ University of Virginia □ Charlottesville  
Rohit Gulati □ Cadence Design Systems □ Sunnyvale

Key Words: Fault tree, Binary Decision Diagram (BDD), Markov model, Software Tool

## SUMMARY AND CONCLUSIONS

Several recent advances in fault tree models have been developed as an aid in the analysis of computer systems, and these have appeared in the RAMS symposium in the past several years. Such advances include the ability to model sequence dependencies [8, 6], the application of fault trees to the analysis of hardware- and software- fault tolerant systems [11], the ability to include details of the recovery process in the fault tree solution [2], and the use of modularization [16] and Binary Decision Diagrams as an aid in solution [4].

In this paper we present a software package (DIFtree) that implements these advances into a single methodology that can solve both dynamic and static fault trees, and which is applicable to the analysis of hardware, software and humanware in complex computer-based systems.

## 1. INTRODUCTION

DIFtree (Dynamic Innovative Fault Tree) is both a methodology and prototype software tool providing a unique approach for reliability analysis of complex computer-based systems. The technical aspects of the DIFtree methodology have been described in [3, 10, 18, 15]; in this paper we describe the software package which implements the methodology.

DIFtree implements a modular approach to fault tree solution, efficiently combining the best of static and dynamic analysis approaches. What's even better is that DIFtree is easy to use, as it is accompanied by a powerful graphical user interface. This interface allows the user to edit the fault tree graphically, using a window-oriented approach. In addition, the user may describe the fault tree in a textual file, from which the graphical interface will draw the tree automatically. This approach allows the user to interact with DIFtree either textually or graphically.

The graphical interface supports the development of large trees, automatically breaking a large tree into linked pages. Navigation between the pages is simple and intuitive. Shared basic and internal events are handled easily and correctly (which is not the case with some commercially available packages). The solution algorithm finds

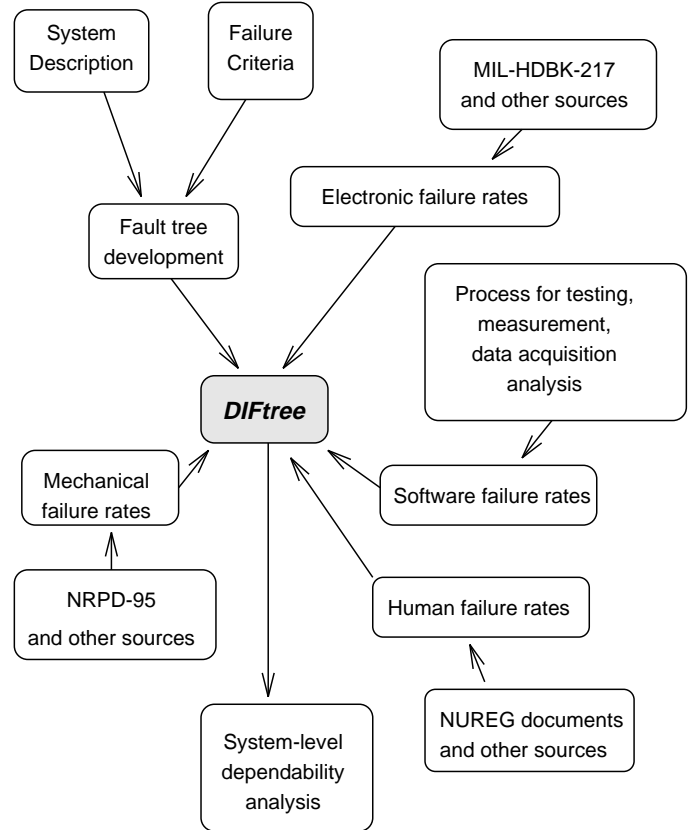


Figure 1: The role of DIFtree in dependability analysis of computer-based systems

independent subtrees automatically, and separately applies the best solution technique to each subtree. The results of the analysis are displayed on the tree, or in separate plot files. All drawings and outputs are available as postscript files for ease of inclusion into reports.

Figure 1 shows the role that DIFtree plays in reliability analysis of computer-based systems. A program such as DIFtree requires the analyst to have a thorough understanding of the system being analyzed, and its success/failure criteria, in order to develop an accurate fault tree representation of the system. In addition, the analyst must provide failure parameters in terms of failure rates

or probabilities; these failure parameters can come from several sources. A fault tree analysis package takes the description of the fault tree and the input parameters and produces an estimate of the probability of system failure as an output.

## 2. HECS: A HYPOTHETICAL EXAMPLE COMPUTER SYSTEM

In order to demonstrate the modeling capability of DIFtree consider a simple hypothetical example computer system (HECS), shown in figure 2. HECS consists of dual-redundant processors A1 and A2 and a cold spare which can replace either upon failure. A cold spare is one which is assumed not to fail before being used.

HECS has 5 memory units; three are required. These memory units are connected to the bus via two memory interface units. If the memory interface unit fails, the memory units connected to it are unusable. Memory unit 3 (M3) is connected to both interfaces for redundancy; thus M3 is accessible as long as either interface unit is operational.

There is also a human operator who interfaces with the system via a console, and runs some software application.

HECS requires at least one of the three A processors, at least 3 of the memory units, at least one of the redundant busses, and the operator, console and software to be operating correctly. We will look at each of these constituent groups of components separately, to see how DIFtree can model their behavior.

### 2.1 Processor modules

Consider first the dual-redundant processors A1 and A2 and their cold spare. The cold spare has a zero failure rate until it is needed, that is, until one of the other processors (A1 and A2) fail. When the cold spare is switched into active operation, its failure rate jumps to some non-zero value (presumably the failure rate of an active processor). Further, the cold spare is available to replace whichever of the two active processors (A1 or A2) fails.

Figure 3 shows the portion of the fault tree which models the failure behavior of the processors. The discontinuous failure rate is handled by the *Cold spare gate*[7] in the dynamic fault tree model. The leftmost input to the cold spare gate is the primary (active) component. The other inputs represent cold spare units which are switched into active operation as needed. As shown in figure 3, the spare inputs can be shared with other cold spare gates.

The dynamic fault tree model provides a warm spare gate and hot spare gate as well. A warm spare has a reduced failure rate when dormant while a hot spare has the same failure rate as when active.

### 2.2 Memory Units

The function of the memory units provide an interesting illustration of functional dependencies. There are 5 memory units, of which 3 are required for reliable operation. Thus nominally, one would expect to connect the basic events for the 5 memory units to a simple 3/5

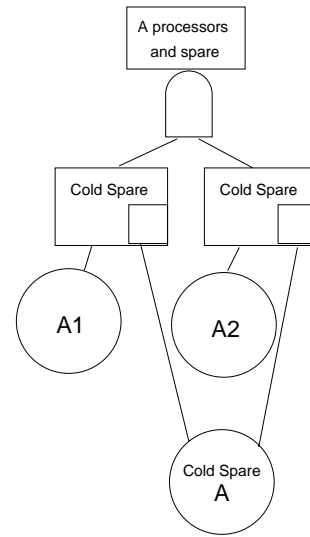


Figure 3: Fault tree model for processors

gate, since when 3 of the 5 units fail, the system becomes unreliable. However, there is an added complication to consider. The memory units are connected to the busses via a pair of memory interface units. The memory interface unit must be operational in order for the memory unit to be accessible; thus we say that the memory units are *functionally dependent* on the interface.

Figure 4 shows the portion of the fault tree that models the memory units. The 5 memory units (M1 through M5) are indeed connected to a 3/5 gate. The dependency of the memory units on the interface is captured in the three functional dependency gates. The memory interface units are the *trigger* input to the functional dependency gates, and the dependent basic events are the memory units. The functional dependency gate operates by labeling the dependent basic events as failed when the trigger event occurs [7]. The trigger event for a functional dependency gate need not be a basic event; the second functional dependency gate triggers the failure of memory unit M3 when both interface units fail.

Since the functional dependency gate does not produce a logical output that affects the fault tree output, it is connected to the fault tree via a dummy output signified by a dashed line in the figure. Thus the 3/5 gate has five logical inputs, and three dummy inputs.

### 2.3 HECS system level fault tree

The full system-level fault tree for the HECS system is shown in figure 5. The **operator console** basic event represents the hardware, while the **console software** basic event represents the application software. The undeveloped basic event [1] labeled **operator** represents potential failures that the operator might induce.

For the redundant busses, the two statistically identical busses have been combined into one replicated event labeled **2\*Bus**. This basic event is treated logically the same as two separate events; combining them together

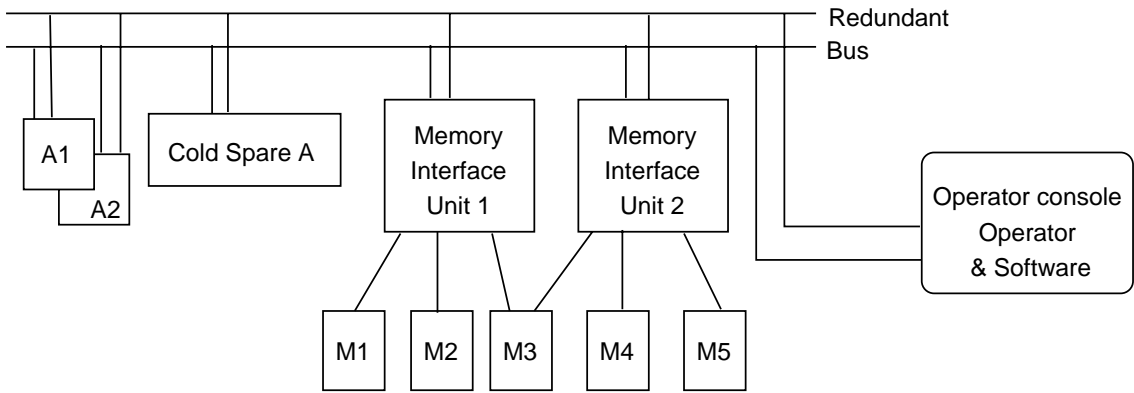


Figure 2: HECS: A hypothetical example computer system

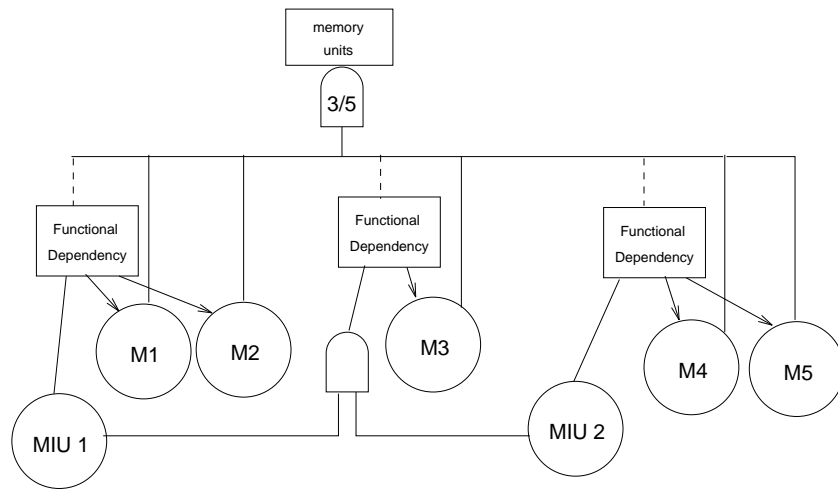


Figure 4: Fault tree model for memory units

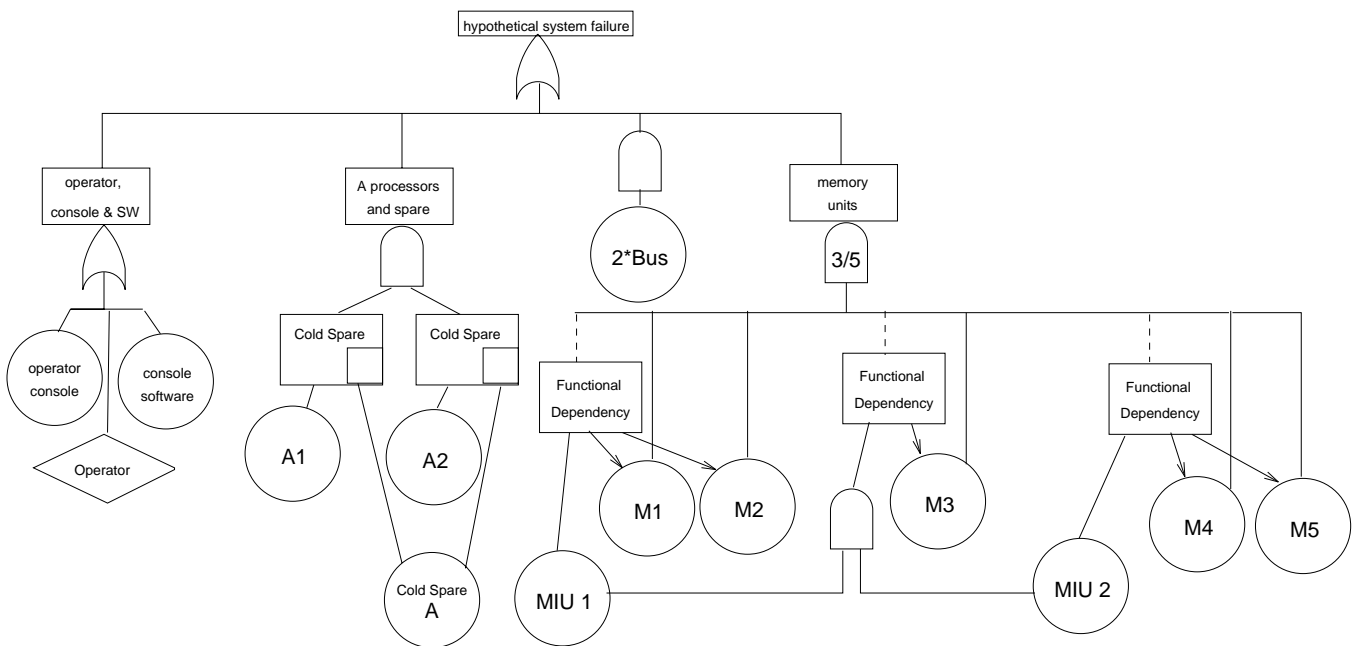


Figure 5: HECS system-level fault tree model

aids in both model development and solution.

### 3. DIFtree GRAPHICAL AND TEXTUAL INTERFACE

DIFtree provides a graphical user interface that allows the user to draw the fault tree on the screen. Editing options such as **Cut**, **Paste**, **Delete**, and **Insert** are supported by pull-down menus. The fault tree is drawn automatically by the interface as nodes (gates and basic events) are added. When the tree being developed is too large to fit on a single screen, additional pages are added and connected automatically. Hierarchical navigation between pages allows the user to trace any path up or down through multiple pages. As additional nodes are added or deleted, the page separations are dynamically recomputed.

Hard copies of the fault tree are produced in Postscript files, which may be printed or incorporated into other documents. Multi-page fault trees are automatically paginated, and page pointers are provided so that the analyst can easily see how the pages are connected.

Once the fault tree is solved, the probabilities of occurrence for the top node and several internal nodes (those that are the roots of independent subtrees) are displayed on the fault tree. If multiple solution times were requested, then a plot of the probability of the top event is automatically displayed.

In addition to providing a powerful graphical interface, a textual description of the fault tree is maintained as well. The analyst can view/edit the textual description as well as the graphical description. The graphical representation of the fault tree is drawn automatically from the textual description, and any changes made graphically can be saved in the textual format.

The textual fault tree file describes in a simple manner the logical nodes in the fault tree in terms of their inputs. Keywords are used to facilitate understanding. For example, the top event in the tree is indicated by the keyword *Toplevel* followed by the node name for the top event. Other nodes are listed by name, followed by node type, followed by a list of inputs. For the HECS system, textual description of the tree begins as shown in figure 6. As you can see, each line of the text file describes a node in the fault tree. The format of the line is *node-name, node-type, list of input nodes* followed by a semicolon. The nodes can be described in any order and node descriptions can span several lines.

The basic events in the textual file are described using keywords. For a basic event node, the format of the line is *node-name* followed by either **prob=** or **rate=** and the numerical value. Other parameters (such as the coverage parameters and dormancy factor for warm spares) are described similarly.

### 4. COVERAGE PARAMETERS

Each basic event has associated with it more than simply its probability or rate of occurrence. Since DIFtree

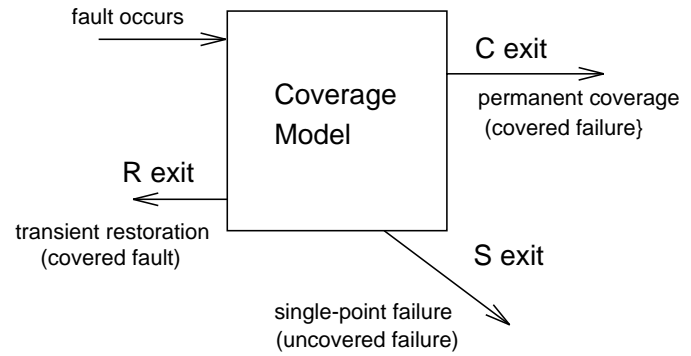


Figure 7: General structure of a coverage model

is being developed with the goal of being able to model computer-based systems, the ability to model imperfect fault coverage is a vital requirement.

*Coverage* is the probability that a system can automatically recover from a fault, given that a fault occurs. A coverage model is used to analyze the fault and error recovery behavior of an adaptive system, and is used to assess coverage. A coverage model includes information about fault duration (permanent, transient) and the effectiveness of the system recovery procedures [12]. Including the concept of coverage (and the possibility that recovery may be imperfect) in the system level model is critical to an accurate dependability assessment.

DIFtree coverage parameters are derived from a coverage model which takes the form shown in figure 7. The entry point to the model signifies the occurrence of the fault, and the three exits signify three possible outcomes. The transient restoration exit (labeled *R*) represents the correct recognition of and recovery from a transient fault. A transient is usually caused by external or environmental factors, such as excessive heat or a “glitch” in the power line. It is generally believed that the majority of computer system faults are transient. Successful recovery from a transient fault restores the system to a consistent state without discarding any components, for example by retrying an instruction or rolling back to a previous checkpoint. Reaching this exit successfully requires timely detection of an error produced by the fault, performance of an effective recovery procedure, and the swift disappearance of the fault (the cause of the error).

The permanent coverage exit (labeled *C*) denotes the determination of the permanent nature of the fault, and the successful isolation and removal of the faulty component. The single point failure exit (labeled *S*) is reached when a single fault causes the system to crash. This generally occurs if an undetected error propagates through the system, or if the faulty unit cannot be isolated and thus the system cannot be reconfigured. For a more complete discussion of coverage modeling, see [12, 5, 9, 17].

DIFtree incorporates coverage models into the fault tree model via three parameters which are (optionally) associated with each basic event. These three coverage parameters are the probability of reaching each of the three

```

Toplevel Hypothetical-system-failure;
Hypothetical-system-failure OR console-operator-software a-processors memory-units;
  console-operator-software OR OPERATOR SOFTWARE OPERATOR-CONSOLE;
  a-processors AND csp1 csp2;
    csp1 CSP A1 cold-spare-A;
    csp2 CSP A2 cold-spare-A;
memory-units 3OF5 M1 M2 fdep1 M3 redundant-memory-interface M4 M5 fdep2;
  fdep1 FDEP memory-interface-unit1 M1 M2;
  redundant-memory-interface FDEP both-MIU M3;
    both-MIU AND memory-interface-unit1 memory-interface-unit2;
  fdep2 FDEP memory-interface-unit2 M4 M5;

```

Figure 6: Portion of textual description of HECS

exits of a coverage model of the type shown in figure 7, where the three parameters sum to one.

### 5. Modular solution

DIFtree models are solved via a modular combination of dynamic and static solution methods. Independent subtrees, which share no inputs, are found using an algorithm developed by Dutuit and Rauzy [13, 15]; in the example system there are 4 independent subtrees, as shown in figure 8. Subtrees can be detected at any level of the fault tree; in our example they all happen to appear as inputs to the top event.

The first and fourth independent subtrees are static, since they have no dynamic gates. The basic events in these subtrees can have either a fixed probability of occurrence or a constant failure rate. A static subtree is solved using a Binary Decision Diagram solution which incorporates coverage [3].

The second and third independent subtrees are dynamic, since each contains at least one dynamic gate. For a dynamic fault tree, a constant failure rate must be defined for each basic event, since the model is solved using Markov methods [7, 14].

When the independent subtrees are solved, their solutions are combined using the fault tree gates which remain. That is, each independent subtree is replaced in the overall fault tree with a basic event (with a corresponding probability of failure and coverage parameters). That is, the HECS fault tree is reduced to a single OR gate with four inputs, one from each independent subtree.

### 5. FEATURES

DIFtree (Dynamic Innovative Fault Tree) is both a methodology and prototype software tool providing a unique approach for reliability analysis of complex computer-based systems. DIFtree combines the best of static and dynamic fault tree analysis techniques using a modular analytical approach, and is unique in several aspects:

- The detection of modules in a fault tree uses a fast and efficient algorithm to identify independent

pieces. These independent submodels can be solved separately, thereby allowing an exact solution in minimum time.

- The submodels are classified as either static or dynamic, depending on the temporal relationships between the input events. Static subtree gates express the failure criteria in terms of combinations of events. Dynamic subtree gates express the failure criteria in terms of combinations and order of occurrence for input events. More important than the classification, however, is that the different subtree types are solved using different solution techniques. That is, the static subtrees are solved using combinatorial methods (specifically a method based on Binary Decision Diagrams) while the dynamic subtrees are solved using Markov methods.
- Since it is known that modeling the possibility of imperfect fault coverage is critical to correct evaluation of computer-based systems, we include this capability in both the static and dynamic solution techniques.
- The solution of static fault trees using methods based on Binary Decision Diagrams, and including imperfect coverage in this solution, is unique to our approach. The BDD approach to fault tree analysis is a new innovation which holds great potential for the analysis of large models.
- The dynamic fault tree model allows the analysis of complex redundancy management techniques in a relatively simple manner. Factors which normally lead to difficulties with analysis include cold, warm and hot spares, spares which are shared among several different components, functional dependencies and common-cause failures; these and more are handled easily using our special dynamic gates.
- Our modular approach to dynamic models can allow the analysis of much larger systems than with traditional (non-modular) approaches. Further, the iden-

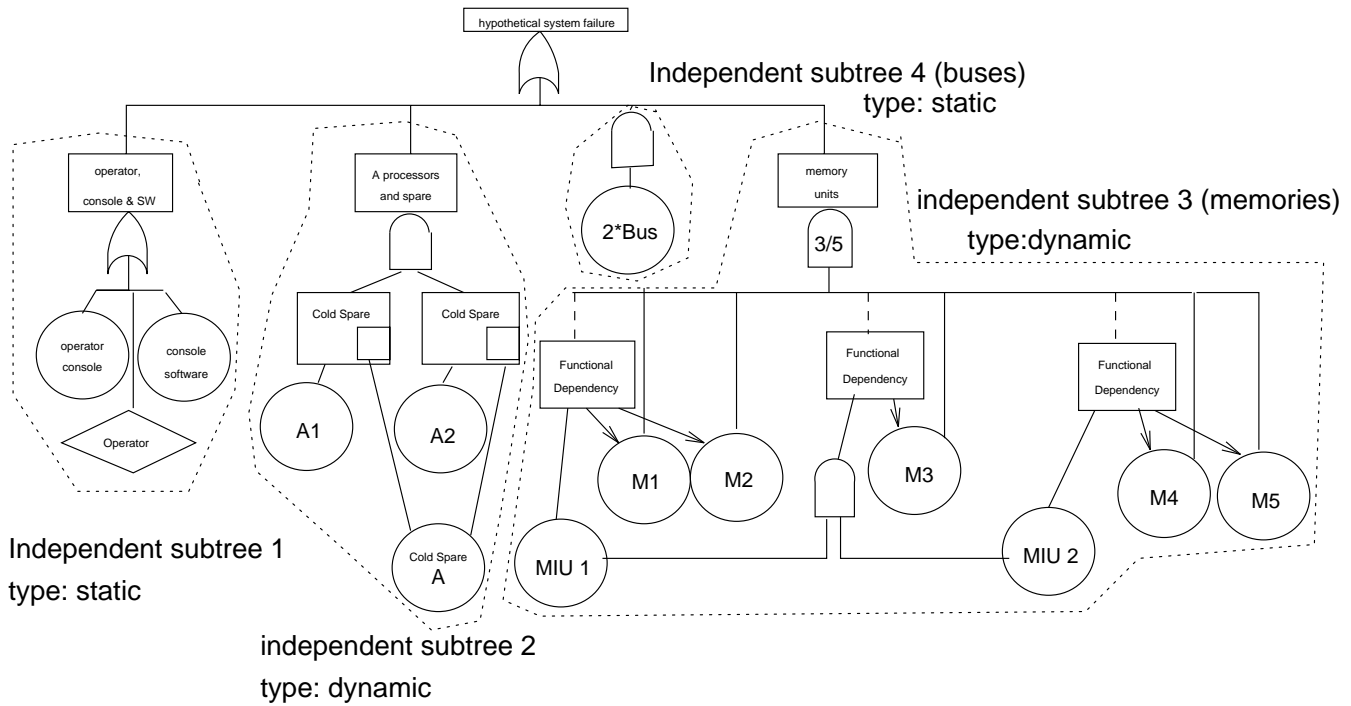


Figure 8: Independent subtrees in fault tree model for HECS

tification of independent submodels is significantly easier in our approach, since it is done at the fault tree level. It is not feasible, in general, to detect modules directly in a Markov model.

- Since our methodology allows the combination of constant probabilities and rates of occurrence, it is therefore applicable to more than just hardware. Our methodology is equally applicable to software and human operator failures as to hardware failures, and thus is well suited to the analysis of complex computer based systems.

## 6. ACKNOWLEDGEMENTS

The authors are grateful to Henk Roelant for many helpful discussions, and to Rajiv Reddy and Ramesh Rao for technical assistance. DIFtree was developed at the University of Virginia under the sponsorship of NASA Langley Research Center grant number NCC-1-210. DIFtree is a prototype implementation of part of the SHADE Tree methodology [18].

## References

- [1] Fault tree handbook. US Nuclear Regulatory Commission NUREG-0492, 1981.
- [2] Stacy A. Doyle and Joanne Bechta Dugan. Fault trees and imperfect coverage: A combinatorial approach. In *Proceedings of the Reliability and Maintainability Symposium*, pages 214–219, January, 1993.
- [3] Stacy A. Doyle and Joanne Bechta Dugan. Dependability assessment using binary decision diagrams. In *Proc. IEEE Int. Symp. on Fault-Tolerant Computing, FTCS-25*, June 1995.
- [4] Stacy A. Doyle, Joanne Bechta Dugan, and Mark A. Boyd. Combinatorial-models and coverage: A binary decision diagram (BDD) approach. In *Proceedings of the Reliability and Maintainability Symposium*, January, 1995.
- [5] Stacy A. Doyle, Joanne Bechta Dugan, and Ann Patterson-Hine. A combinatorial approach to modeling imperfect coverage. *IEEE Transactions on Reliability*, pages 87–94, March 1995.
- [6] Joanne Bechta Dugan, Salvatore Bavuso, and Mark Boyd. *Tutorial: Modeling Advanced Fault Tolerant Architectures*. 1991 Reliability and Maintainability Symposium, 1991.
- [7] Joanne Bechta Dugan, Salvatore Bavuso, and Mark Boyd. Dynamic fault tree models for fault tolerant computer systems. *IEEE Transactions on Reliability*, September 1992.
- [8] Joanne Bechta Dugan, Salvatore Bavuso, and Mark Boyd. Fault trees and sequence dependencies. In *Proceedings of the Reliability and Maintainability Symposium*, pages 286–293, January, 1990.
- [9] Joanne Bechta Dugan, Salvatore J. Bavuso, and Mark A. Boyd. Fault trees and markov models for reliability analysis of fault tolerant systems. *Reliability Engineering and System Safety*, 39:291–307, 1993.
- [10] Joanne Bechta Dugan, Stacy A. Doyle, and Laura L. Pulum. New approaches to fault tree analysis. *Communication in Reliability, Maintainability, Supportability and Logistics*, July 1996.

- [11] Joanne Bechta Dugan and Ann Patterson-Hine. Simple models of fault tolerant software. In *Proceedings of the Reliability and Maintainability Symposium*, January, 1993.
- [12] Joanne Bechta Dugan and K. S. Trivedi. Coverage modeling for dependability analysis of fault-tolerant systems. *IEEE Transactions on Computers*, 38(6):775–787, 1989.
- [13] Yves Dutuit and Antoine Rauzy. A linear-time algorithm to find modules in fault trees. *IEEE Transactions on Reliability*, September 1996.
- [14] Rohit Gulati. A modular approach to static and dynamic fault tree analysis. Master’s thesis, University of Virginia Department of Electrical Engineering, 1996.
- [15] Rohit Gulati and Joanne Bechta Dugan. A modular approach for analyzing static and dynamic fault trees. In *Proceedings of the Reliability and Maintainability Symposium*, January, 1997.
- [16] Ann Patterson-Hine and Joanne Bechta Dugan. Modular techniques for dynamic fault tree analysis. In *Proceedings of the Reliability and Maintainability Symposium*, pages 363–368, January, 1992.
- [17] Michael Pecht, editor. *Product Reliability, Maintainability and Supportability Handbook*. CRC Press, 1995.
- [18] Laura L. Pullum and Joanne Bechta Dugan. Fault trees models for the analysis of complex computer-based systems. In *Proceedings of the Reliability and Maintainability Symposium*, pages 200–207, January, 1996.

## Biographies

Prof. Joanne Bechta Dugan  
 Department of Electrical Engineering  
 Thornton Hall  
 University of Virginia  
 Charlottesville, VA 22903-2442  
 (804) 982-2078  
 FAX: (804) 924-8818  
 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 PhD 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 ACM, Eta Kappa Nu and Phi Beta Kappa.

Bharath Venkataraman  
 Department of Electrical Engineering  
 Thornton Hall  
 University of Virginia  
 Charlottesville, VA 22903-2442  
 FAX: (804) 924-8818  
 e-mail: bharath@Virginia.edu

Bharath Venkataraman is presently a graduate student in Electrical Engineering at the University of Virginia. He was awarded the B.E. degree in Electronics and Communication engineering from Osmania University, India, in 1994. His research interests include reliability and communications. He is a member of the IEEE communications society and Eta Kappa Nu.

Rohit Gulati  
 Alta Group of Cadence Design Systems  
 555 N. Mathilda Avenue  
 Sunnyvale, CA  
 e-mail: rgulati@cadence.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. He worked for a year as a software engineer in CMC Ltd., India. Rohit is currently working in the R&D wing of 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.