

# Combining Various Solution Techniques for Dynamic Fault Tree Analysis of Computer Systems

Ragavan Manian and Joanne Bechta Dugan

University of Virginia Department  
of Electrical Engineering

Thornton Hall, Charlottesville, VA 22903

jbd@ranger.ee.virginia.edu, rmanian@fore.com

David Coppit and Kevin J. Sullivan

University of Virginia Department  
of Computer Science

Thornton Hall, Charlottesville, VA 22903

{coppit|sullivan}@cs.virginia.edu

## Abstract

*Fault trees provide a graphical and logical framework for analyzing the reliability of systems. A fault tree provides a conceptually simple modeling framework to represent the system-level interactions between component reliabilities. Dynamic fault trees have been shown particularly useful for reliability analysis of embedded computer systems. Dynamic fault trees are a superset of traditional (static) fault trees in that additional gates are used to model sequential behavior. DIFtree [1] is our fault tree methodology for the analysis of dynamic fault trees, effectively combining the best static fault tree solution technique (Binary Decision Diagrams) with Markov solution techniques for dynamic fault trees. DIFtree includes advanced techniques for modeling coverage; coverage modeling has been shown to be critical to the analysis of fault tolerant computer systems. DIFtree is based on a divide-and-conquer technique for modularizing the system level fault tree into independent sub-trees; different solution techniques can be used for sub-trees. In this paper we extend the DIFtree analysis capability to model several different distributions of time to failure, including fixed probabilities (no time component), exponential (constant hazard rate), Weibull (time varying hazard rate), and log normal. Our approach extends both the BDD and Markov analytical approaches and incorporates simulation as well. An example analysis illustrates our approach.*

## 1. Introduction

The reliability of embedded computer systems can be difficult to analyze for several reasons. First, hardware and software are integrated and thus must be considered in a single model. Second, fault tolerance techniques for automatic recovery from detected errors raises the possibility

that these automatic mechanisms may themselves fail, a consideration which is embodied in a coverage model [2]. Third, comprehensive models of embedded computer systems are frequently quite large, and thus require long solution times, since the solution time for reliability models is exponential in the size of the model [3]. Finally, analytic solutions require simplifying assumptions about the time to failure behavior of the components. DIFtree (Dynamic Innovative Fault trees) is a fault tree methodology that provides an interesting solution to the first three problems. DIFtree is based on a modularization technique [4] that identifies, in linear time, independent sub-trees. We note that [5] is pursuing a similar modularization based on Rauzy's algorithm. These sub-trees are then classified as static or dynamic, depending on whether the failure relationships are Boolean or temporal [6]. Static fault trees use traditional Boolean logic gates (AND, OR, K-of-M) to represent the combinations of component failures that cause system failure. Static fault trees are most efficiently modeled using a technique based on Binary Decision Diagrams [7]. Dynamic fault trees model sequential dependencies between events, and are analyzed using Markov methods. Techniques for incorporating coverage models are available for both approaches [8]. Galileo is an experimental software tool including the DIFtree methodology in a rich analysis environment [10].

The primary contribution of this research is to extend the DIFtree modeling framework to support new distributions and fault tree configurations, and to support an alternative solution technique in the form of the Monte Carlo simulation. The benefit of the improved analysis engine is to model more parametric distributions such as the Weibull and Log Normal distributions. Monte Carlo simulation for fault tree analysis allows the analysis of systems that cannot be modeled even by the extended analytical approach. The modularization algorithm ensures that the use of BDDs, Markov Chains or Monte Carlo

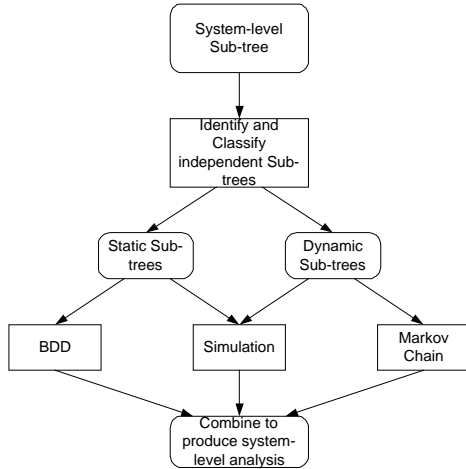


Figure 1: The DIFtree methodology

simulation for solving the various sub-trees is transparent to the user.

The rest of the paper is organized as follows: Section 2 provides an overview of the major modules of the DIFtree framework. Section 3 discusses the properties of the distribution types supported. Section 4 deals with the issues involved in extending support to the various distributions. Section 5 discusses the integration of the Weibull distribution as an instance of the integration process. The modeling capability of our methodology is illustrated using a simple hypothetical example in Section 6. Results and Conclusions are found in Section 7.

## 2. Background – The DIFtree methodology

The fault tree of a complex system is often large and complex in itself. In order to solve such a fault tree efficiently, DIFtree uses a divide-and-conquer method by which independent sub-trees are identified and solved by suitable solvers. Figure 1 illustrates the modularization approach which is characteristic of the DIFtree methodology. Sub-trees comprised of only AND, OR and K-of-M gates are solved by conversion to the equivalent BDD or possibly by simulation, while dynamic sub-trees are solved by Markov models or simulation.

### 2.1 Static fault tree analysis using binary decision diagrams

Static fault trees are graphical representations of combinatorial (And-Or) relationships between basic events; these relationships can be expressed by Boolean functions.

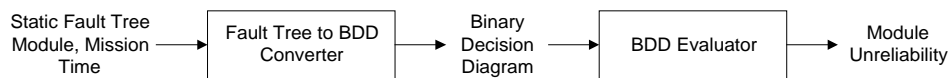


Figure 2: Static solver program flow

A BDD is a compact canonical representation of a Boolean function. The BDD has no duplicate sub-trees or redundant vertices [12]. The algorithms for manipulating the BDD have time complexity proportional to the sizes of the graphs operated on, and hence are quite efficient as long as the graphs do not grow too large. The static fault tree analysis program [8] uses Binary Decision Diagrams (BDD) as an alternative to the traditional cutset-based solution. Since there seems to be no simple relation between the number of components of a system and the size of the BDD that represents it, BDDs provide efficient solutions for large combinatorial systems without resorting to cutsets [7]. The program uses a modified representation of BDDs to account for coverage modeling [2], a means of accounting for the possible failure of automatic recovery mechanisms built into fault tolerant systems. The general algorithm is shown in Figure 2.

### 2.2 Dynamic fault tree analysis using Markov chains

Dynamic trees model failures that cannot be represented using simple combinatorial relationships, such as failure sequences, functional dependencies, and stand-by spares. In order to solve dynamic fault trees, it is necessary to make use of a representation that will keep track of the history of the system, in the form of a “state”. For this reason, dynamic fault tree analysis uses Markov chains [13]. A state in the Markov Chain contains all the system information regarding component failures, sequence of component failures, and information on spare allocations. Component failures lead from existing states to new states in a recursive fashion. Hazard rates of the failed components characterize the transitions from one state to another. The Markov chains map into a set of equivalent ordinary differential equations with variables corresponding to state probabilities. These equations are solved using an ordinary differential equation solver. Currently we use the CVODE solver [14]. The probability of being in any failed state during the mission time gives an estimate of the system unreliability. The general algorithm is shown in Figure 3

### 2.3 Monte Carlo simulation

A third technique uses Monte Carlo simulation to evaluate fault trees. Simulation is an attractive alternative because it allows the modeling of virtually any time-to-failure distribution, as well as allowing behaviors that preclude analytic solution. Further, simulation may be a more efficient approach for high-redundancy situations, for ex-

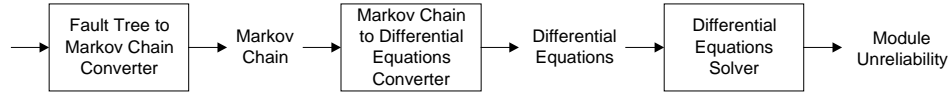


Figure 3: Dynamic solver program flow

ample in k-of-m arrangements with large values of k and m, and for non-identical replicates. The combinatorial explosion in the case of analytical models in such situations gives simulation an advantage over analytical solutions. The major disadvantage of simulation is the long simulation times needed to achieve high accuracy. However, using variance reduction techniques help achieve dramatic reductions in solution time. The feasibility of simulation using importance sampling to achieve variance reduction when simulating some dynamic fault trees was demonstrated in MCI-HARP [15].

### 3. Component reliability functions supported by DIFtree

Two models for time to failure were supported in the DIFtree methodology: fixed probabilities (independent of time) and exponential time to failure (constant hazard rate). Fixed probabilities are frequently applied to software design faults while the latter is frequently applied to physical random faults [16]. Software for embedded computer applications is most easily (though not always adequately) modeled with fixed probabilities of failure, as data for more complex growth models is usually unavailable. Physical failures of hardware and random failures of system software are more accurately modeled using probability distributions, usually exponential. Fixed failure probabilities are only applicable to static fault tree models; however static fault tree solution methods can support distributions as well. Dynamic fault tree models require consideration of time to failure in order to model sequential behavior and thus cannot support fixed (time independent) failure probabilities. In this section, we discuss three reliability distributions currently supported by our tool. Although several other failure probability functions exist, our fault tree analysis tool presently supports the ones discussed here because they are representative of a diverse class of systems.

#### 3.1 Exponential distribution

The exponential distribution is the only continuous distribution with a constant hazard function. It is appropriate in cases when a used component that has not failed is as “good” as a new component. Some of the lifetime distribution representations for the exponential distribution are:

$$S(t) = e^{-\lambda t}, \text{ and}$$

$$h(t) = \lambda$$

where

$S(t)$  = Survivor function, and  $h(t)$  = hazard rate

The statistical techniques for the exponential distribution are particularly tractable; many components use the exponential distribution for the sake of simplicity. The memory-less property imposes a restriction on its applicability to a wide class of component failures. Therefore there is a clear need for other distributions that are more flexible for modeling, although more complex mathematically.

#### 3.2 Weibull distribution

Many electro-mechanical systems such as computer peripheral devices typically degrade over time, and are more likely to follow a distribution with a strictly increasing hazard rate with time. In some other cases, systems become less likely to fail when used longer, implying a decreasing hazard rate. The Weibull distribution can model both increasing and decreasing hazard rates. Some of the lifetime distribution representations for the Weibull distribution are:

$$S(t) = e^{-(\lambda t)^\kappa}, \text{ and}$$

$$h(t) = \kappa \lambda^\kappa t^{\kappa-1}$$

where  $\lambda > 0$  and  $\kappa > 0$  are the scale and shape parameters of the distribution. The hazard function approaches zero from infinity for  $\kappa < 1$ , and increases from zero when  $\kappa > 1$ . The Weibull distribution has some special properties for certain values of  $\kappa$ : for  $\kappa = 1$ , it is the exponential distribution; for  $\kappa = 2$  it is the Raleigh distribution; and for  $3 < \kappa < 4$ , the probability density function (*pdf*) resembles that of the Normal distribution's *pdf*, and the mode and median of the distribution are equal when  $\kappa = 3.26$ .

#### 3.3 Log normal distribution

The log normal distribution has an upside-down bathtub (UBT) shaped hazard function, where  $h(t)$  increases initially and then decreases. The parameters are  $\mu$  and  $\sigma$ , since the logarithm of a log normal random variable is a normal random variable with mean  $\mu$  and standard deviation  $\sigma$ . Since the survivor function of the log normal distribution is not in a closed form [17], it is not commonly used. The log normal distribution is not applicable for use in either homogenous or non-homogenous Markov models because it does not satisfy the Markov property [18].

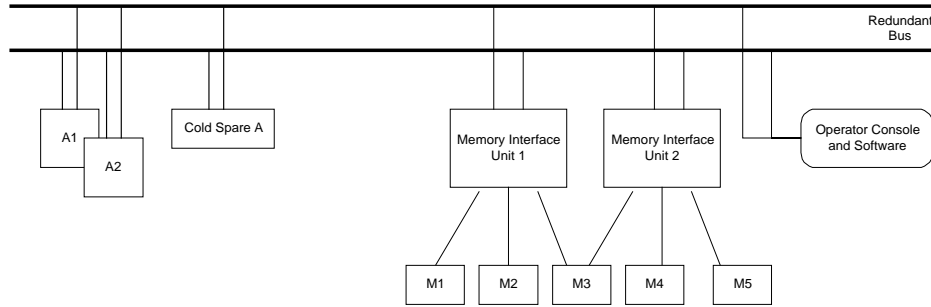


Figure 4: HECS block diagram

#### 4. Issues involving tool support for various distributions

The framework of the DIFtree methodology allows it to modularize fault trees into static and dynamic parts and solve them separately, and then to integrate the results. Since the analysis techniques of static and dynamic fault trees are quite different, the distributions that they can support are also different. The BDD evaluation algorithm only requires the probabilities of the component failures obtained by evaluating the survivor functions of these components at the mission time. In the case of the Markov model, the hazard rates of the components becomes the measure of interest. This is because the transition rates from one Markov state to another are functions of these hazard rates. These transition rates in turn form the coefficients of a family of ordinary differential equations, whose variables correspond to the probabilities of the system being in each of the Markov states. Because the solution to a dynamic tree is equivalent to the solution of this family of ordinary differential equations, the properties of the component time-to-failure distributions and hazard rates become relevant.

#### 5. Integrating Weibull distribution to the fault tree tool

##### 5.1 Integrating Weibull into Static Solver

The fault tree gates determine the Boolean relation between the basic events and system failure, while the ordering of the basic events determines the exact BDD that is created. The BDD solver employs a heuristic algorithm to arrive at an optimal ordering of the basic events. The conversion of fault tree to Binary Decision Diagram involves obtaining the probabilities of the BDD nodes (each BDD node represents a component failure). Each component's failure distribution parameters are stored in the fault tree node that corresponds to that basic event.

Given the mission time, the BDD nodes are decorated with their probabilities of failure.

##### 5.2 Integrating Weibull into dynamic solver

Components that obey Weibull distribution of failure times result in time-varying transitions, while those that obey exponential distribution of failure times result in time-invariant transitions. This is handled by creating different classes of transitions within the Markov chain for different distributions. A time-varying transition is evaluated to get the numerical value of the corresponding hazard rate from the algebraic expression stored within the corresponding transition instance in the Markov Chain. Evaluating the Markov chain at start time first creates the transition rate matrix. The entries in the matrix that are composed of Weibull transitions are also identified at this stage. The solver [14] solves the differential equations by integrating over the mission time, and invokes the functional interface of the Markov Chain to re-calculate the Weibull hazard rates at the internal time steps. The system of equations may be stiff since the coefficients of the transition rate matrix can vary over a large range of values in case of Weibull distributions with shape factors  $\ll 1$ : appropriate solver options are exercised.

#### 6. Illustration: A Hypothetical Example Computer System (HECS)

We demonstrate the capabilities of the DIFtree methodology that we discussed above (viz. modularization into static and dynamic fault trees, alternative evaluation using simulation, and handling of different time to failure distributions) using the hypothetical example computer system called HECS. The HECS example is not a real computer system; it was first used in [1] and has been adapted here for the sake of illustration. HECS consists of dual-redundant processors A1 and A2 and a cold spare that can replace either upon failure. The cold spare does not fail before its use.

HECS has five 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

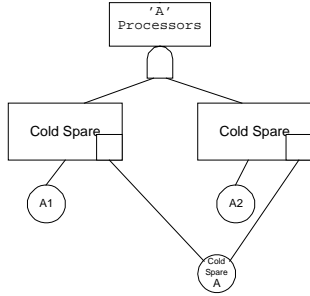


Figure 5: Processor modules fault tree

HECS requires that at least one of the three A processors, at least three 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 we can model their behavior.

### 6.1 Processor modules

We consider the dual-redundant processors first. Figure 5 above shows the portion of the fault tree that models the failure behavior of the processors. The cold spare has zero failure rate until it is switched into active use. Further, the cold spare is available to replace whichever of the two active processors (A1 or A2) fails. This discontinuous failure rate is handled by the CSP (cold spare gate) in the corresponding dynamic fault tree. The left most input to the cold spare gate is the primary (active) component. The other inputs represent cold spare units that are switched into active operation as needed. In this case, the failure distribution for this spare may be assumed to be either exponential (same failure rate at all times) or Weibull (decreasing failure rate due to burn-in), depending on the failure data and behavior. One cannot solve a dynamic model that has a Weibull rate of cold spare failure if it is modeled using an analytical method such as Markov chains. This is because it will be unclear what the failure rate is at the time of switching in. In other words, if it is switched on at time “t”, the failure rate could either be  $\lambda(0)$  or  $\lambda(t)$  depending on whether we assume the switched-on spare to be “good as new” or “good as old”. The latter case can be solved using the Markov Chain-based analytical model but contradicts the cold spare requirement. However, this situation can be modeled by simulation because different clocks (i.e. “t” values) can be maintained for different components. This sub-tree reiterates the need for simulation as an alternative to analytical models.

### 6.2 Memory units

The memory units require the memory interface units to be operational in order to be connected to the rest of 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.

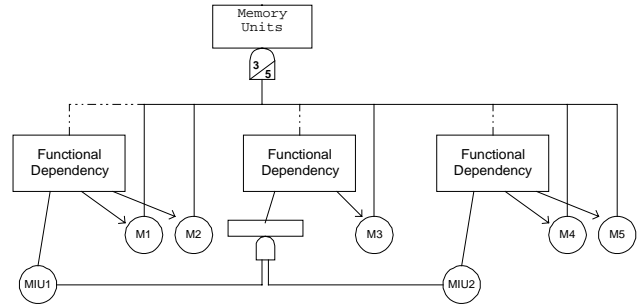


Figure 6: Memory units fault tree

system, and this fact is modeled using FDEP gates with the Memory Interface Units (MIU1 and MIU2) as triggers, as shown in Figure 6. There are five memory units (M1-M5) of which three are required for reliable operation. The failure of three out of the five of them would cause the system to fail. The memory units are connected to the buses via a pair of memory interface units. It is assumed in this example that two of the memory modules (M3 and M4) obey a Weibull distribution of *tff*, while all the other components obey the exponential distribution.

### 6.3 Operator, console and software

The system fails when any one of operator, console or software fails. In order to demonstrate the newly integrated Log Normal distribution (bathtub based hazard rate) the operator failure is modeled using Log Normal distribution. The operational console being an electromechanical device is most aptly modeled by a Weibull time to failure distribution with increasing hazard rate. The fault tree is shown in Figure 7

### 6.4 HECS system-level fault tree

The full system-level fault tree is shown in Figure 8. The modularization algorithm identifies each of the four main sub-trees to be an independent module of either type static or type dynamic, and sends it to the appropriate solver. Thus the “Operator, console and SW”, and the Buses blocks are sent to the static solver. The “Buses” sub-tree has a basic event labeled “2\*Bus” to represent the fact that there are two statistically identical redundant buses. The combination of these two components into one event can reduce the computation needed for the solution. The “A Processors” block goes the Monte Carlo Simulator. The Memory Units go to the dynamic solver. For providing a better understanding of the underlying analytical techniques, the intermediate representation of the static tree modules, the Binary

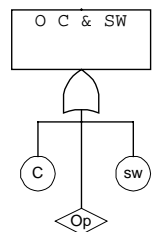


Figure 7: Operator, console, and software

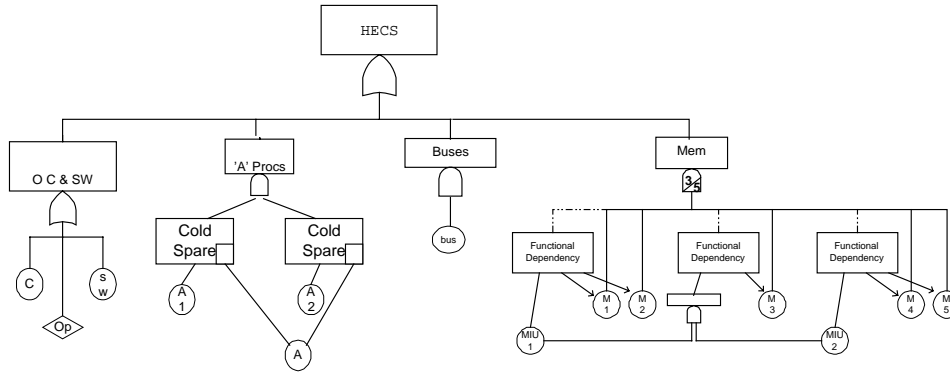


Figure 8: HECS system-level fault tree

Decision Diagram is shown for the static modules. A portion of a Markov Chain, which is the intermediate representation of the dynamic tree modules, is also shown for one of the dynamic modules. These representations provide insight into how the numerical solution is obtained.

### 6.5 BDD representations for the operator-control software, and the two bus configurations

The BDD representations of the Operator, Console and Software (OCS) block and the Buses block are shown in Figure 9. Each non-leaf node in the BDD represents a component that can fail, and each path from the root to a leaf node represents a disjoint combination of component failures (Boolean value 1) and non-failures (Boolean value 0). If the leaf node for a path is labeled with a "1" then the path leads to system failure; if the leaf is labeled with a "0" then the path represents an operational system configuration. The unreliability of the system is given by the sum of the probabilities for all the paths from the root to a leaf node labeled "1".

Thus, the Boolean and probability equations represented by the BDDs in Figure 9 are:

Operator, Console and Software:

$$F = OC + SC + Op, \text{ and}$$

$$P(F) = P(OC) + (1 - P(OC)) \cdot P(SC) + (1 - P(OC) - P(SC)) \cdot P(OP)$$

Buses:

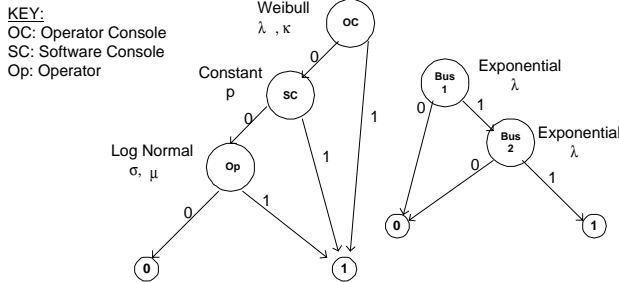


Figure 9: OCS and bus BDDs

$$F = \text{Bus1} \cdot \text{Bus2}, \text{ and}$$

$$P(F) = P(\text{Bus1}) \cdot P(\text{Bus2});$$

The values for the P's are obtained by substituting the parameters in the Survivor function of the respective distributions at the required time t. Imperfect coverage is added automatically via a methodology presented in [8].

### 6.6 Markov Chain representation of the memory module fault tree

In Figure 10 below, a portion of the automatically generated Markov chain model of the fault tree of the Memory Units module is shown (there was a total of 87 states and 190 transitions in this Markov chain). Because of the redundant configuration of the system represented by the 3-of-5 gate (ref. Figure 7) these component failures do not necessarily result in system failure. Only certain sequences of failure will cause the system to reach "F", the system Failure states. The F's are absorbing states, since no further failures are possible from F. This continuous-time Markov chain is evaluated by deriving a family of differential equations for the unconditional probabilities of being in a given state, and solving those equations for the given time period [19]. The unreliability of the system is the sum of the probabilities for the failure states.

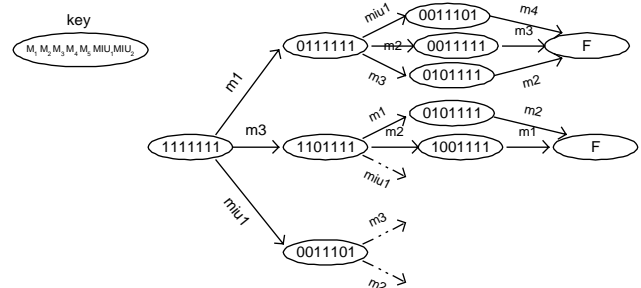


Figure 10: Portion of the Markov chain

| Independent Sub-tree Name | Independent Sub-tree type | Gates Used  | Solution Technique     | Component Failure Types   |                               |                              |              |
|---------------------------|---------------------------|-------------|------------------------|---------------------------|-------------------------------|------------------------------|--------------|
|                           |                           |             |                        | Exponential ( $\lambda$ ) | Weibull ( $\lambda, \kappa$ ) | Log Normal ( $\sigma, \mu$ ) | Constant (p) |
| Processors                | Dynamic                   | Cold Spares | Monte Carlo Simulation | ✓                         | ✓                             | –                            | –            |
| Operator, SW & console    | Static                    | Or          | BDD                    | ✓                         | –                             | ✓                            | ✓            |
| Memory units              | Dynamic                   | FDEPs, KofM | Markov chain           | ✓                         | ✓                             | –                            | –            |
| Buses                     | Static                    | And         | BDD                    | ✓                         | –                             | –                            | –            |

Table 1: HECS fault tree features

## 7. Results and conclusions

The simple example clearly demonstrated the benefits of this work. By extending the dynamic fault tree analysis methodology to include new time-to-failure distributions (Weibull and log normal) and solution models (Monte Carlo simulation) we were able to estimate the reliability of a system which lay outside the scope of traditional fault tree analysis techniques. The DIFtree approach to dynamic fault tree analysis via modularization provides an excellent capability for the reliability analysis of computer based systems. The extension of this modeling capability to incorporate alternative distributions for time to failure has highlighted the utility of this approach in several ways. First, the modularization approach, splitting the system level tree into independent sub-trees and then labeling each sub-tree as either static or dynamic, has allowed us to permit a wider range of distributions for the static sub-tree. In fact, virtually any distribution can be supported in the simulation approach. Second, the modularization helps identify cases where independent Markov chains can be developed, resulting in smaller models that would be needed otherwise. Smaller Markov models make it more feasible to incorporate time-dependent failure rate, as the presence of such time dependence can dramatically increase solution time. Third, the incorporation of simulation as an alternative for sub-tree solution further extends the set of distributions that can be supported. The Monte Carlo simulator is best utilized in solving fault trees that cannot be solved by either analysis technique. Each of the separate approaches (combinatorial, Markov and simulation) has both advantages and disadvantages. The combination of these via modularization helps us to exploit the advantages and avoid some of the drawbacks.

## Acknowledgements

This work was funded in part by the National Science Foundation, NASA Langley Research Center and Lockheed-Martin Federal Systems. The authors are grateful to Sal Bavuso for his continued support of our work, for in-

sightful technical discussions and for the use of the MCI-HARP simulation engine for our proof of concept. We wish him well in his retirement.

## References

- [1] Joanne Bechta Dugan, Bharath Venkataraman and Rohit Gulati. "DIFtree: A software package for the analysis of dynamic fault tree models." Annual Reliability and Maintainability Symposium, 97RM-047 pages 1-7, 1997.
- [2] 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
- [3] Ros75 Placeholder until we get the reference
- [4] Yves Dutuit and Antoine Rauzy. "A linear-time algorithm to find modules in fault trees." IEEE Transactions on Reliability, September 1996.
- [5] Anju Anand and Arun K. Somani. "Hierarchical analysis of fault trees with dependencies, using decomposition." Proceedings of the Annual Reliability and Maintainability Symposium, pages 69-75, 1998
- [6] Rohit Gulati, Joanne Bechta Dugan. "A modular approach for analyzing static and dynamic fault trees." Proceedings of the Annual Reliability and Maintainability Symposium, 97RM-045: pages 1-7, 1997
- [7] Olivier Coudert and Jean Christophe Madre. "METAPRIME, an Interactive Fault-Tree Analyser". IEEE Transactions on Reliability, March 1994.
- [8] Stacy A. Doyle and Joanne Bechta Dugan. "Dependability assessment using Binary Decision Diagrams." Proceedings of the International Symposium on Fault Tolerant Computing, pages 249-258, June 1995
- [9] Rohit Gulati. "A modular approach to static and dynamic fault tree analysis." Master's thesis, University of Virginia Department of Electrical Engineering, 1996
- [10] Kevin J. Sullivan. "Galileo: An advanced fault tree analysis tool." URL: <http://www.cs.virginia.edu/~ftree>
- [11] Kevin J. Sullivan. "Better, cheaper, faster tools." Proceedings of the Annual Reliability and Maintainability Symposium, 1997.

- [12] Randal E. Bryant. "Graph-Based Algorithms for Boolean function manipulation." IEEE Transactions on Computers, August 1986
- [13] Joanne Bechta Dugan, Salvatore Bavuso and Mark Boyd. "Dynamic fault tree models for fault tolerant computer systems." IEEE Transactions on Reliability, September 1992
- [14] Cohen, S.D., and A.C. Hindmarsh. "CVODE, a Stiff/Nonstiff ODE Solver in C." Computers in Physics, March/April 1996.
- [15] Mark A. Boyd and Salvatore J. Bavuso. "Simulation modeling for long duration spacecraft control systems." Proceedings of the Annual Reliability and Maintainability Symposium, pages 106-113, 1993
- [16] Michael R. Lyu, editor." Handbook of Software Reliability Engineering." IEEE Computer Society Press, 1996.
- [17] Lawrence M. Leemis. "Reliability: Probabilistic models and statistical methods." Prentice-Hall, 1995.
- [18] J. R. Norris, "Markov Chains." Cambridge University Press 1997.
- [19] Kishor S. Trivedi. "Probability and statistics with reliability, queing, and computer science applications." Prentice-Hall, 1982