Available online at www.sciencedirect.com
ScienceDirect
Reliability Engineering and System Safety 93 (2008) 1616–1627
RELIABILITY ENGINEERING & SYSTEM SAFETY
www.elsevier.com/locate/ress

# Construction of event-tree/fault-tree models from a Markov approach to dynamic system reliability

Paolo Bucci[a], Jason Kirschenbaum[a], L. Anthony Mangan[b], Tunc Aldemir[b,*], Curtis Smith[c], Ted Wood[c]

[a]*Department of Computer Science and Engineering, The Ohio State University, 395 Dreese Labs, 2015 Neil Avenue, Columbus, OH 43210, USA*
[b]*The Ohio State University, Nuclear Engineering Program, 427 Scott Laboratory, 201 West 19th Avenue, Columbus, OH 43210, USA*
[c]*Idaho National Laboratory, MS 3850, Idaho Falls, ID 83415, USA*

Received 28 August 2007; received in revised form 15 January 2008; accepted 20 January 2008
Available online 10 March 2008

**Abstract**

While the event-tree (ET)/fault-tree (FT) methodology is the most popular approach to probability risk assessment (PRA), concerns have been raised in the literature regarding its potential limitations in the reliability modeling of dynamic systems. Markov reliability models have the ability to capture the statistical dependencies between failure events that can arise in complex dynamic systems. A methodology is presented that combines Markov modeling with the cell-to-cell mapping technique (CCMT) to construct dynamic ETs/FTs and addresses the concerns with the traditional ET/FT methodology. The approach is demonstrated using a simple water level control system. It is also shown how the generated ETs/FTs can be incorporated into an existing PRA so that only the (sub)systems requiring dynamic methods need to be analyzed using this approach while still leveraging the static model of the rest of the system.


*Keywords:* Markov; Dynamic reliability; PRA; Dynamic event tree; Dynamic fault tree

## 1. Introduction

While the event-tree (ET)/fault-tree (FT) methodology is by far the most popular approach to probabilistic risk assessment (PRA), concerns have been raised in the literature over the past 25 years regarding its potential limitations in the reliability modeling of dynamic systems. These concerns include:

- lack of time element in the ET/FT methodology to represent fault propagation through logic loops or possible dependence of the system failure modes on the exact timing of the component failures with respect to the changing magnitudes of the plant process variables [1,2],
- treatment of the coupling between the plant physical processes and triggered or stochastic events (e.g., valve openings, pump startups) which could lead to statistical dependence between failure events [3],
- semi-quantitatively modeling of the propagation of system disturbances through a classification of changes in process variables (e.g., small, moderate, large) which may lead to omission of some failure mechanisms due to inconsistencies in the definition of the allowed ranges for the process variables [3,4] or due to possible significant changes in the system behavior arising from very small changes in system parameters [5],
- possible sensitivity of Top Event frequencies to stochastic changes in the system settings [2] or process dynamics.

A more detailed discussion of the possible limitations of the ET/FT approach is given in [6]. A more recent review regarding its applicability to the reliability modeling of digital instrumentation and control systems is given in [7].

Markov reliability models have been traditionally used to account for statistical dependencies between hardware failures [7]. Augmented with the cell-to-cell mapping

*Corresponding author.
*E-mail addresses:* bucci.2@osu.edu (P. Bucci), kirschen@cse.ohio-state.edu (J. Kirschenbaum), mangan.10@osu.edu (L.A. Mangan), aldemir.1@osu.edu (T. Aldemir).

0951-8320/$ - see front matter © 2008 Elsevier Ltd. All rights reserved.
doi:10.1016/j.ress.2008.01.008

technique (CCMT), Markov models can be also used to address the concerns indicated above for the ET/FT methodology [8,9]. A challenge in the use of Markov reliability models for nuclear plant PRAs is that, while they have been utilized to model plant subsystems with statistically dependent failures [10–12], they cannot be used (nor needed) to model the whole plant due to state-space explosion. The state-space explosion issue can be addressed by using Markov models only for plant subsystems where needed (e.g. steam generator feedwater control system of a pressurized water reactor [12] or for the risk modeling of digital instrumentation and control systems [13]), however, there are no mechanized procedures that allow the incorporation of the Markov reliability model for a portion of the plant into an existing PRA for the plant that is based on the ET/FT methodology.

While ET interpretation of Markov chains seems to be common in operations research and economics [14–18], very few attempts have been encountered in the reliability literature to generate ETs or FTs from Markov models [19]. In this paper, we explore a new approach to the generation of failure scenarios and their compilation into dynamic event trees (DETs) or dynamic fault trees (DFTs) from a Markov model of the system, which allows the Markov model to be incorporated into PRAs based on the ET/FT methodology in a mechanized manner. The DETs are similar to conventional event trees except that the branching times are determined from the system simulator through user specified branching rules and associated probabilities to generate and quantify the likelihood of possible scenarios following an initiating event [20–23]. The branching rules can be used to model the uncertainty in hardware/human/process behavior. For example, if the normal system operation requires valve opening upon pressure increasing above a preset limit, a branching rule could be that the valve opens or not (fails) when the system pressure as determined by the system simulator reaches this limit, each possibility leading to different scenarios. Refs. [22,24] show, respectively, how uncertainties in human and process behavior can be modeled using DETs starting from a given initiating event. In that respect, the DET generation is based on inductive logic and the DET has to be regenerated for changing initiating event conditions. The DFTs also account for timing of failure events, however, they use deductive logic to identify event sequences leading to a specified undesirable event (Top Event). They have been mostly used to model hardware/software failure dependencies [25] or system availability when there is repair [26].

While DETs can be independently generated using the probabilistic simulation of the dynamic behavior of the system as described above, the advantage of the use of Markov models to generate DETs over independent generation of DETs are the following:

- Markov models are generally applicable for all possible initial conditions in the discrete state-space range of interest.
- They can represent the dynamics of systems with control loops in a more compact manner than DETs for both normal and abnormal system operation.
- Modeling uncertainties or uncertainties in the initial conditions of systems with continuous controlled/monitored variables can be accounted for using the Markov/CCMT approach [7].

Section 2 describes the example dynamic system used for illustration of the proposed approach (Section 4). Section 3 gives an overview of the Markov/CCMT methodology used to construct the Markov reliability model for the example dynamic system. Section 5 implements the proposed approach on the example dynamic system and Section 6 illustrates how to incorporate the resulting DETs and DFTs into an existing PRA using the SAPHIRE code [27].

## 2. Example dynamic system

In this section we introduce a simple level control system [9] often used as benchmark in the literature. We will refer to this system as the example system throughout the rest of the paper. The example system is depicted in Fig. 1. The example system consists of a water tank, two water supply units (Units 1 and 2), and one drain unit (Unit 3). Each control unit has a separate level sensor and we assume the sensor is part of the control unit.

There is one process variable—the level $x$ of the water in the tank—that is maintained in the nominal control region, $\mathrm{lsp} \leqslant x \leqslant \mathrm{hsp}$, by the control units according to the following control laws:

- If $\mathrm{lsp} \leqslant x \leqslant \mathrm{hsp}$ then Unit 1 is on, Unit 2 is off, and Unit 3 is on.
- If $x < \mathrm{lsp}$ then Unit 1 is on, Unit 2 is on, and Unit 3 is off.
- If $x > \mathrm{hsp}$ then Unit 1 is off, Unit 2 is off, and Unit 3 is on.

The lsp and hsp are the low and high setpoints, respectively. There are two failure modes for the example system: *dryout* occurs when $x < L$ and *overflow* occurs when $x > H$, where $L$ and $H$ are the lowest and highest allowable liquid levels, respectively. Each control unit can be in one of four states: on, off, failed on, and failed off.

The evolution of the level process variable is described by

$$\frac{\mathrm{d}x}{\mathrm{d}t} = Q(\alpha_1 + \alpha_2 - \alpha_3) \quad \alpha_i = \begin{cases} 0 & \text{if unit } i \text{ is off or failed off} \\ 1 & \text{if unit } i \text{ is on or failed on} \end{cases} (i = 1, 2, 3) \tag{1}$$

where $Q$ is a positive real constant.

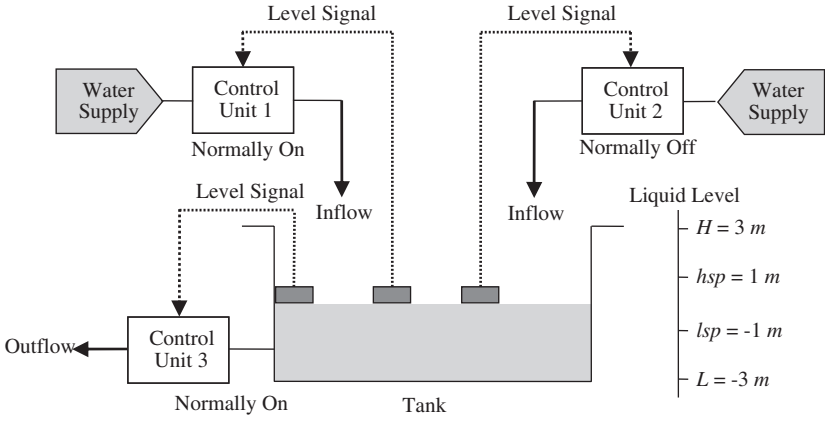


Fig. 1. A simple level control system (adapted from [9]).

## 3. Markov model and CCMT

The CCMT [28] regards system evolution in time as probability of transition of the process variables $x_l$ $(l = 1, \ldots, L)$ between their specified magnitude intervals. These intervals form $L$-dimensional cells $V_j = \{x_l;\ a_{l,j} \leqslant x_l < b_{l,j};\ j = 1, \ldots, J_l;\ l = 1, \ldots, L\}$ in the system state-space in a similar fashion to those used by finite difference or finite element methods. Once the $V_j$ are specified, the probability $p_{n,j}(k)$ at time $t = k\tau$ that $\boldsymbol{x}(k\tau) = \boldsymbol{x} \equiv [x_1\ x_2 \ldots x_l \ldots x_L]$ is within $V_j$ and $\boldsymbol{\alpha}(k\tau) = \boldsymbol{\alpha}_{n'}$ where $\boldsymbol{\alpha}_{n'}$ is the system parameter vector corresponding to component state combination $n(k\tau) \equiv n (n = 1, \ldots, N)$, can be recursively calculated from the Markov chain

$$p_{n,j}(k+1) = \sum_{n'=1}^{N} \sum_{j'=1}^{J=J_1 \ldots J_L} q(j, j|n', j', k) p_{n',j'}(k) \tag{2}$$

where the elements of transition matrix are defined as

$$q(n, j|n', j', k) = g(j|j', n', k) h(n|n', j' \to j, k) \tag{3}$$

with $g(j|j', n', k)$ denoting the conditional probability that $\boldsymbol{x}((k+1)\tau)$ is in $V_j$ given $\boldsymbol{x}(k\tau)$ is in $V_{j'}$ and $n((k+1)\tau) = n$, and $h(n|n', j' \to j, k)$ denoting the conditional probability $n((k+1)\tau) = n$, that given $n(t) = n'$ for $k\tau \leqslant t < (k+1)$, $\boldsymbol{x}(k\tau)$ is in $V_{j'}$ and $\boldsymbol{x}((k+1)\tau)$ is in $V_j$. The $h(n|n', j' \to j, k)$ in Eq. (3) are found from the given transition probabilities between component state combinations during the time interval $k\tau \leqslant t < (k+1)\tau$. For components with statistically independent failures, the $h(n|n', j' \to j, k)$ are simply products of the given individual component failure or non-failure probabilities during $k\tau \leqslant t < (k+1)\tau$. The $j$ and $j'$ in $h(n|n', j' \to j, k)$ are relevant if the component failure data are given as failure-per-demand or component failures and process variable transitions between $V_j$ are statistically dependent. In these situations, $h(n|n', j' \to j, k)$ are either direct inputs or can be determined from the given control laws for the system and the transition probabilities between component states during the time interval $k\tau \leqslant t < (k+1)\tau$ in a mechanized fashion as shown in [9]. The $g(j|j', n', k)$ in Eq. (3) are calculated from

$$g(j|j', n') = \frac{1}{v_{j'}} \int_{V_{j'}} \mathrm{d}\mathbf{x} e_j[\tilde{\mathbf{x}}(\mathbf{x}', n, \tau)] \tag{4}$$

where

$$e_j(y) = \begin{cases} 1 & \text{if } y \in V_j \\ 0 & \text{otherwise} \end{cases} \tag{5}$$

$\tilde{\mathbf{x}}(\mathbf{x}', n, \tau)$ is the arrival point of the system trajectory at time $t = (k+1)\tau$ in the state-space, given that the system departed from $\mathbf{x}'$ at $t = k\tau$ with configuration $n$, and $v_j'$ is the volume of the cell $V_{j'}$.

The choice of the modeling time step $\tau$ depends on the choice of the cells $V_j$ used to partition the system state-space, time constants of the system and component failure rates. The $\tau$ should be large enough that the system can move out of the cell it is in within $\tau$ for some of the starting points $x'$ in Eq. (4). It should be also small enough that the system does not cross more than one setpoint requiring a state change in the components within this time interval. Otherwise, the assumption $n(t) = n'$ for $k\tau \leqslant t < (k+1)$ leads to misrepresentation of system operation. There is no optimal choice of $\tau$. In practical applications, a few numerical experiments need to be carried out for a suitable choice of $t$.

## 4. Dynamic ET/FT generation

The basic idea of our approach is to use the transition matrix of the Markov model of the system as a graph representation of a finite state machine (a discrete process model of the stochastic dynamic behavior of the system). We can then use this representation and standard search algorithms [29] to explore all possible paths to failure (scenarios) with associated probabilities and to construct dynamic event trees of arbitrary depth. In addition, for each type of failure, we can construct a fault tree by traversing the DET backwards (i.e., from failure to initiating event) and collecting all possible failure paths through the event tree.

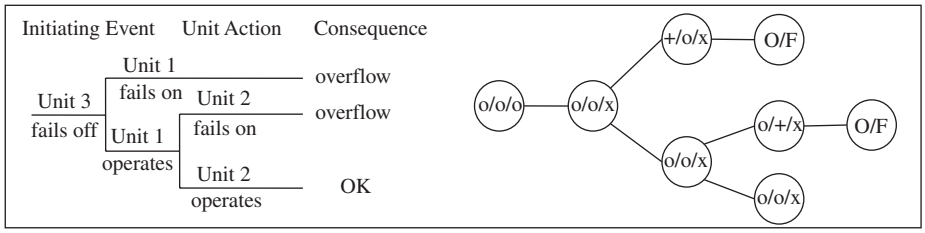


Fig. 2. Event tree vs. tree data structure.

```
initialize DET root node to initial state and probability 1
add DET root node to queue Q of nodes to process
while Q is not empty
    remove next node N = (S,P) from Q
    if S is not a sink state
        for each possible state S'
            if Prob[S,S'] > 0
                compute probability P' for this branch as Prob[S,S'] * P
                if P' > epsilon
                    create new node N' = (S',P')
                    add N' to the list of children of N in the DET
                    add N' to queue Q of nodes to process
                end if
            end if
        end for each
    end if
end while
```

Fig. 3. Dynamic event tree—Algorithm 1.

### 4.1. The DET generation algorithms

The DET is represented by a tree data structure. A tree data structure is composed of 'nodes' where information is stored and 'links' that connect the nodes. The nodes in the tree data structure correspond to the branching points in the DET and the links represent the branches. In Fig. 2 we show a simple event tree for the example level controller and a tree data structure that might be used to represent it. In the example, the tree nodes hold the information about the system component states (represented in the figure as a 3-tuple where the state of each unit is either 'o' for operational, '+' for failed on, or 'x' for failed off). The event corresponding to a specific branch in the tree can be deduced by comparing the configurations at the beginning and at the end of the branch. For instance, the branch between the nodes representing the configurations o/o/o and o/o/x indicates that Unit 3 has failed off.

#### 4.1.1. Algorithm 1

The nodes in the tree data structure used in the first DET generation algorithm (Fig. 3) store the state $S$ of the system and the probability $P$ that the system will have followed the specific path in the DET from the initial state to state $S$. In the current discussion, a *state* of the system is always meant to include both the state of all the process variables and the configuration of all system components.

The algorithm uses a queue (first-in, first-out behavior) to hold the nodes in the tree before they are processed by the algorithm. It starts by creating the top node in the tree (the root) containing the initial state of the system (usually the state in which the process variables are all in the nominal range and all the system components are operational) and a probability equal to 1. This node is added to the initially empty queue. Then the algorithm repeatedly goes through the steps of removing and processing the first node in the queue. The algorithm terminates when the queue becomes empty, or it could easily be modified to terminate when a certain number of levels in the tree have been generated. Each node $N = (S, P)$ extracted from the queue is processed as follows. If $S$ is a sink state, there is nothing more to be done. Otherwise, by consulting the transition matrix, *Prob*, generated by the Markov model, we find all the states $S'$ in which the system can evolve from state $S$ in time $\tau$. For each state $S'$ we compute the probability $P'$ of entering that state, at the given time, and having followed the given path through the tree, by multiplying the probability $P$ of being in node $N$ by the probability of going from $S$ to $S'$ (Prob[$S$, $S'$]). If $P'$ is below a chosen threshold, we do not need to expand that path any farther. Otherwise, we create a new node $N' = (S', P')$, add it to the list of children of $N$ in the tree, and append it to the queue to be processed at a later time.

The basic DET generated by this algorithm can be quite large due to the many branches emanating from most nodes. At the cost of a little extra complexity in the algorithm, we can group the DET paths by configuration changes (Section 4.1.2). The states at each level in the

```
initialize DET root node to initial state(s) and probability 1
add DET root node to queue Q of nodes to process
while Q is not empty
    remove node N = <(S1,P1),…,(Sk,Pk)> from Q
    initialize A: array [1..number of configurations] of nodes
    for each pair (S,P) in the list of pairs in N
        if S is not a sink state
            for each possible state S'
                if Prob[S,S'] > 0
                    compute probability P' for this branch as Prob[S,S'] * P
                    if P' > epsilon
                        if S' is not in the list of states in node A[Conf(S')]
                            add (S',P') to the list of states in node A[Conf(S')]
                        else
                            add P' to the current probability value associated with S'
                              in the list of states in node A[Conf(S')]
                        end if
                    end if
                end if
            end for each
        end if
    end for each
    add all the nodes in A that contain at least one pair
      to the list of children of N in the DET and to queue Q
end while
```

Fig. 4. Dynamic event tree—Algorithm 2.

tree are grouped by common configuration of system components. Each branch in the DET corresponds to failure of 0 or more of the system components.

#### 4.1.2. Algorithm 2

The algorithm in Fig. 4 shows how we can construct this new DET. In this case, each node in the event-tree representation corresponds to a specific configuration of the system components at a specific time. Each node contains a list of all the states in which the system could be at that time and in the particular scenario (path) in the tree and, for each state, the probability of being in that state at that time. All the states in a node share the same configuration of the system components. Two nodes in the tree are linked if it is possible for the system under consideration to go from one of the states in the first node to one of the states in the second node in the time $\tau$.

Just like Algorithm 1, Algorithm 2 employs a queue to hold the nodes in the tree before they are processed. In addition, Algorithm 2 uses an array of nodes to keep track, at each iteration, of which configurations have already been generated and which states in each configuration can be reached at that point in time. We start by creating the root node of the DET containing one or more initial states where all the process variables are in the nominal range and all system components are operational. If there is more than one initial state, each state is assigned a probability so that the sum of the probabilities is 1. The root node is inserted in the initially empty queue. Just like the first, Algorithm 2 repeatedly goes through the steps of removing and processing the first node in the queue, and terminates when the queue becomes empty. The processing of each node, however, requires some extra complexity. We start by initializing an array of nodes containing one node for each distinct configuration of system components. Then for each state, probability pair ($S$, $P$) in the current node for which $S$ is not a sink state, we find all the states $S'$ in which the system can evolve in the time increment $\tau$. For each state $S'$, we compute the probability $P'$ of entering that state, at the given time, and having followed the given path through the tree, by multiplying the probability $P$ of being in state $S$ by the probability of going from $S$ to $S'$ (Prob[$S$, $S'$]). If $P'$ is below a chosen threshold, we ignore that path. Otherwise, we add the pair ($S'$, $P'$) to the node in array $A$ corresponding to the configuration of system components in state $S'$ ($A$[Conf($S'$)]). If a pair with state $S'$ is already present, we simply add $P'$ to the current probability associated with $S'$. When we have processed all the pairs in the current node, we add all nodes in array $A$ that contain at least one (state, probability) pair to the list of children of $N$ in the tree, and add the same nodes also at the end of the queue to be processed at a later time. Note that both Algorithm 1 and Algorithm 2 rely only on the connectivity of the nodes and do not reflect the time-dependent characteristics of the Markov model.

The tree resulting from Algorithm 2 is usually still large enough that it cannot be viewed in its entirety. Instead of generating the entire tree with the algorithm in Fig. 4, we have developed a tool that allows the user to explore the DET interactively and to generate the DET on demand. The tool is very fast: only those parts of the tree that the user is currently viewing are actually generated. We discuss this tool and the results of the analysis of the level controller system in Section 6.

Initial State: 4 on/off/on -0.33 <= x <= 0.33 (NO FAILURE)

| State | Paths | Probability | Configuration | Level |
|---|---|---|---|---|
| 98 | 7 | 0.1168194473 | ON/ON/on | x > 3.00 (OVERFLOW) |
| 164 | 85 | 0.0976659975 | ON/OFF/OFF | x > 3.00 (OVERFLOW) |
| 186 | 124 | 0.0877072660 | OFF/ON/OFF | x > 3.00 (OVERFLOW) |
| 175 | 11 | 0.0745854679 | off/ON/OFF | x > 3.00 (OVERFLOW) |
| 296 | 85 | 0.0665902764 | ON/ON/ON | x > 3.00 (OVERFLOW) |
| 131 | 7 | 0.0546680051 | ON/off/OFF | x > 3.00 (OVERFLOW) |
| 197 | 189 | 0.0019635399 | ON/ON/OFF | x > 3.00 (OVERFLOW) |
| Total: | 508 | 0.5000000000 | | |

| State | Paths | Probability | Configuration | Level |
|---|---|---|---|---|
| 251 | 64 | 0.1250000000 | OFF/OFF/ON | x < -3.00 (DRYOUT) |
| Total: | 64 | 0.1250000000 | | |

| State | Paths | Probability | Configuration | Level |
|---|---|---|---|---|
| 257 | 13 | 0.1250000000 | ON/OFF/ON | -0.33 <= x <= 0.33 (NO FAILURE) |
| 281 | 16 | 0.0688316559 | OFF/ON/ON | 1.00 <= x <= 1.67 (NO FAILURE) |
| 145 | 18 | 0.0589711047 | OFF/OFF/OFF | -1.67 <= x <= -1.00 (NO FAILURE) |
| 149 | 16 | 0.0503159982 | OFF/OFF/OFF | 1.00 <= x <= 1.67 (NO FAILURE) |
| 277 | 18 | 0.0278641030 | OFF/ON/ON | -1.67 <= x <= -1.00 (NO FAILURE) |
| 278 | 19 | 0.0274108370 | OFF/ON/ON | -1.00 <= x <= -0.33 (NO FAILURE) |
| 146 | 19 | 0.0150594141 | OFF/OFF/OFF | -1.00 <= x <= -0.33 (NO FAILURE) |
| 279 | 13 | 0.0005438601 | OFF/ON/ON | -0.33 <= x <= 0.33 (NO FAILURE) |
| 147 | 13 | 0.0003979661 | OFF/OFF/OFF | -0.33 <= x <= 0.33 (NO FAILURE) |
| 280 | 12 | 0.0003495440 | OFF/ON/ON | 0.33 <= x <= 1.00 (NO FAILURE) |
| 148 | 12 | 0.0002555170 | OFF/OFF/OFF | 0.33 <= x <= 1.00 (NO FAILURE) |
| Total: | 169 | 0.3750000000 | | |

Fig. 5. Summary analysis of level control system.

### 4.2. The DFT generation algorithm

The generation of DFTs relies on the DETs. We construct a fault tree by traversing the DET backwards (i.e., from failure to initiating event) and collecting all possible failure paths through the event tree.

The fault tree for a given top event, e.g., dryout, is an OR gate with one input for each failure scenario leading to the chosen top event. Each failure scenario is captured by an AND gate whose inputs represent the (timed) states the system goes through in the chosen failure scenario. For convenience, we will call these AND gates 'first-level ANDs'. Each state of the system is characterized by the state of each of the three control units (OK (on/off), Failed ON, Failed OFF), the level in the tank, and the time relative to the time of the failure. The top event occurs always at time $t = 0$, and time moves backwards in unit increments. For example, $t = -1$ refers to 1 time step before the top event occurred. We chose to capture such timed state of the system in the fault tree by using an AND gate whose 4 inputs are the state of each control unit and of the level tagged with an appropriate time stamp. We will call these AND gates 'second-level ANDs'.

An example DFT for the level controller system is presented in the next section.

## 5. Analysis of benchmark system

We have implemented a prototype tool (in Java) that allows us to generate the transition matrix of the discrete-time Markov chain modeling the benchmark system, and to use this matrix to explore the dynamic behavior of the system. Here we present the results of our analysis of the sample benchmark system. Note that for the purposes of illustration we make the following assumptions:

- $H = 3\,\text{m}$, $L = -3\,\text{m}$, $\text{hsp} = 1\,\text{m}$, and $\text{lsp} = -1\,\text{m}$ (see Fig. 1).
- $Q = 0.01\,\text{m}$ level change/minute (see Eq. (1)).
- Units are not repaired; unit failure rates do not depend on the mode of failure and they are constant. In particular, $\lambda_1 = 7.6\text{e}{-5}$ failure/min, $\lambda_2 = 9.5\text{e}{-5}$ failure/min, $\lambda_3 = 5.2\text{e}{-5}$ failure/min, where $\lambda_i$ is the failure rate for unit $i$.
- Time increment $\tau = 30\,\text{min}$.
- The water level has been discretized into 9 equal intervals, plus one for dryout and one for overflow.

### 5.1. Analysis of the failure behavior of the sample system

One of the features of the tool is the ability to generate all possible paths from a specified initial state to a sink state. Note that, in this example, all failure states of the sample system are sink states. To make this task computationally feasible, we generate only those paths through the state space that do not include cycles. However, at least for the simple system considered in this paper, we have computed exact probabilities associated with each path, including the possibility of cycling through some states an arbitrary number of times.

Starting from the state where the level is between $-\frac{1}{3}$ and $+\frac{1}{3}$ m and all the control units are operational, we generate

Table 1
One possible overflow scenario

| State | Unit 1 (inlet) | Unit 2 (inlet) | Unit 3 (outlet) | Level ($x$) |
|---|---|---|---|---|
| 4 | OK (on) | OK (off) | OK (on) | $-0.33 \leqslant x \leqslant 0.33$ |
| 70 | OK (on) | Failed ON | OK (on) | $-0.33 \leqslant x \leqslant 0.33$ |
| 71 | OK (on) | Failed ON | OK (on) | $0.33 \leqslant x \leqslant 1.00$ |
| 72 | OK (off) | Failed ON | OK (on) | $1.00 \leqslant x \leqslant 1.67$ |
| 94 | Failed ON | Failed ON | OK (on) | $1.00 \leqslant x \leqslant 1.67$ |
| 95 | Failed ON | Failed ON | OK (on) | $1.67 \leqslant x \leqslant 2.33$ |
| 96 | Failed ON | Failed ON | OK (on) | $2.33 \leqslant x \leqslant 3.00$ |
| 98 | Failed ON | Failed ON | OK (on) | $x > 3.00$ (O/F) |

all possible scenarios (paths) that terminate in a sink state. In Fig. 5 we present a summary of the analysis results. The first line shows the starting state. The three tables following the first line in Fig. 5 show, respectively, overflow, dryout, and other non-failure sink states that the system will reach in time. Each row in these tables shows an integer encoding of the end, sink state of the system,[1] the number of distinct paths from the initial state to the end state, the cumulative probability for those paths, and a more readable description of the end state[2] (configuration of system components and level interval).

Fig. 5 shows that there are 7 different failure states in which the system can overflow and only one failure state in which the system dries out. The probability of overflow is 50% and the probability of dryout is 12.5%. There is also a 37.5% probability that the system will end in a sink state without failure. There are 508 different paths leading to overflow (ignoring cycles in the path) and 64 different paths leading to dryout.

### 5.2. Analysis of a specific failure path

We can also use our tool to explore specific failure scenarios. Here is an example of one possible path to an overflow failure (see Table 1). The probability associated with this scenario (6.1%) takes into account the possibility of cycling through one or more states for an arbitrary number of times and reflects the probability that the system will fail this way as time goes to infinity.

The system starts with the water level in the nominal range and all components operational (OK). First, Unit 2 fails on. This causes the level to rise. When the level passes the high setpoint (1.0 m), Unit 1 correctly turns off, even though with Unit 2 failed on there is no way the level can decrease. Next Unit 1 fails on as well. Now the system is doomed. The level keeps rising until overflow occurs.

### 5.3. Generation of the DET

Finally, the prototype tool allows us to construct DETs with the algorithm described above. Starting from a normal state in which all the units are operational, the tool generates all possible configurations at the next time step (using 30 min as the time increment) keeping track of all the possible states the level variable may be in at that point in time and in that configuration of the control units. Fig. 6 below shows the tool window: the left pane shows a primitive representation of the event tree and the right pane shows the possible level states for the configuration and time step currently selected in the left pane with associated probabilities.

Instead of showing the events between branching points (as it is usually done when displaying ETs), the representation of the event tree in the left pane of the prototype shows the configuration of the control units at each branching point. The event(s) corresponding to a specific branch in the tree can be deduced by comparing the configurations to the left and to the right of the branch. For instance, if in the configuration at the left of a branch Unit 1 is operational and in the configuration to the right Unit 1 is failed off, we can deduce that the event that has occurred along that branch is the 'failure off' of Unit 1. We use a special notation to represent the configuration of the three control units at each branching point (or node) in the event tree. A 3-tuple is used where the state of each unit is either 'o' for operational, '+' for failed on, or 'x' for failed off. For example, x/+/o represents a configuration where Unit 1 has failed off, Unit 2 has failed on, and Unit 3 is operational.

The ET in the left pane is generated on demand. The top of the tree (displayed in the top-left corner of the left pane in Fig. 6) represents the normal configuration where all the units are operational (o/o/o). Whenever the user clicks one of the displayed nodes (branching points), the program generates all the possible configurations in which the system may evolve in the given time step. For example, there are 27 such possible distinct configurations after the first time step (3 control units each of which can be in one of 3 different states, operational, failed on, or failed off). By repeatedly clicking and expanding the tree nodes, the user can explore any possible scenario in the tree. For instance, on the (partial) event tree showed in Fig. 6, we have highlighted with boxes one possible path through the overflow scenario presented in Table 1.

The right pane in Fig. 6 shows that the system can be in one of six different states at that time (3.5 h into the simulation) and with that configuration of system components (+/+/o). In only one of these states the system has already failed, and, of course, the probability of this happening in such a short time is very small ($6.194 \times 10^{-8}$).

[1] Each integer encodes the state of the level process variable and the configuration of system components. The level is an integer between 0 and 10 indicating in which cell the level is. The configuration is an integer between 0 and 26: each of the 3 control unit/valve can be in one of 3 states: operational, failed on, or failed off, for a total of 27 different configurations. If the unit is operational, the state of the valve is determined uniquely by the state of the level variable and the control laws for the system. The total number of states in the example system is 297.

[2] The configuration of system components is expressed by a 3-tuple showing the state of each unit as either 'on' or 'off', if the unit is operational, or 'ON' to indicate a unit that has failed on, or 'OFF' to indicate a unit that has failed off. For example, OFF/ON/off represents a configuration where Unit 1 has failed off, Unit 2 has failed on, and Unit 3 is operational and off.

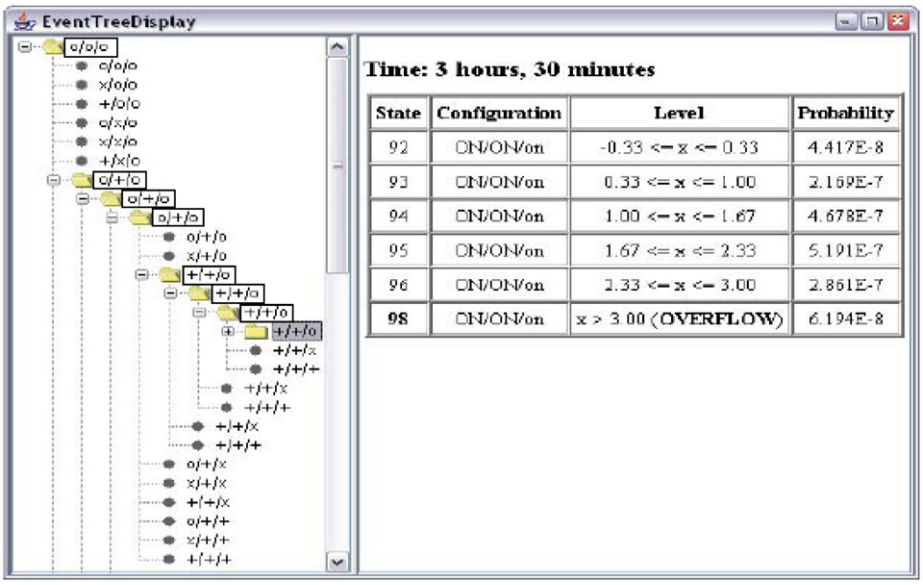


Fig. 6. Display of part of the DET.

Table 2
One possible dryout scenario

| Time | Unit 1 (in) | Unit 2 (in) | Unit 3 (out) | Level ($x$) |
|---|---|---|---|---|
| −8 | OK (on) | OK (off) | OK (on) | $-0.33 \leqslant x \leqslant 0.33$ |
| −7 | Failed OFF | OK (off) | OK (on) | $-0.33 \leqslant x \leqslant 0.33$ |
| −6 | Failed OFF | OK (off) | Failed ON | $-1.0 \leqslant x \leqslant -0.67$ |
| −5 | Failed OFF | OK (on) | Failed ON | $-1.33 \leqslant x \leqslant -1.0$ |
| −4 | Failed OFF | Failed OFF | Failed ON | $-1.33 \leqslant x \leqslant -1.0$ |
| −3 | Failed OFF | Failed OFF | Failed ON | $-2.0 \leqslant x \leqslant -1.67$ |
| −2 | Failed OFF | Failed OFF | Failed ON | $-2.33 \leqslant x \leqslant -2.0$ |
| −1 | Failed OFF | Failed OFF | Failed ON | $-3.0 \leqslant x \leqslant -2.67$ |
| 0 | Failed OFF | Failed OFF | Failed ON | $x \leqslant -3.0$ (DRY) |

### 5.4. Generation of the DFT

The DFTs can be generated from DETs. For a given top event, we trace backwards in time through the event tree collecting all the paths that resulted in the Top Event. Each path is converted into an AND of all the events that occurred along that path (scenario). These AND gates are then attached to a top-level OR gate representing all possible scenarios resulting in the top event under consideration. By using the DFT generation algorithm we can produce a fault tree that can be integrated into an existing PRA (see Section 6). Consider the possible path to failure (dryout) in Table 2.

This event sequence of a possible DET branch for the level controller shows the system starting at time $t = -8$ in a state where all control units are operational and the level is in the nominal range, specifically between $-\frac{1}{3}$ and $+\frac{1}{3}$ m. The outlet unit (Unit 3) is open and only the first inlet unit (Unit 1) is open. At $t = -7$, Unit 1 fails off (closed). The level starts to decrease, but before the level goes below the low setpoint (1.0 m), the outlet unit fails on (open) at time $t = -6$. At time $t = -5$, the level goes below the low setpoint and Unit 2 comes on to try and increase the level. But at time $t = -4$, Unit 2 fails off too. With the outlet unit open and both inlet units closed, the level continues to decrease until at time $t = 0$ the system experiences a failure by dryout.

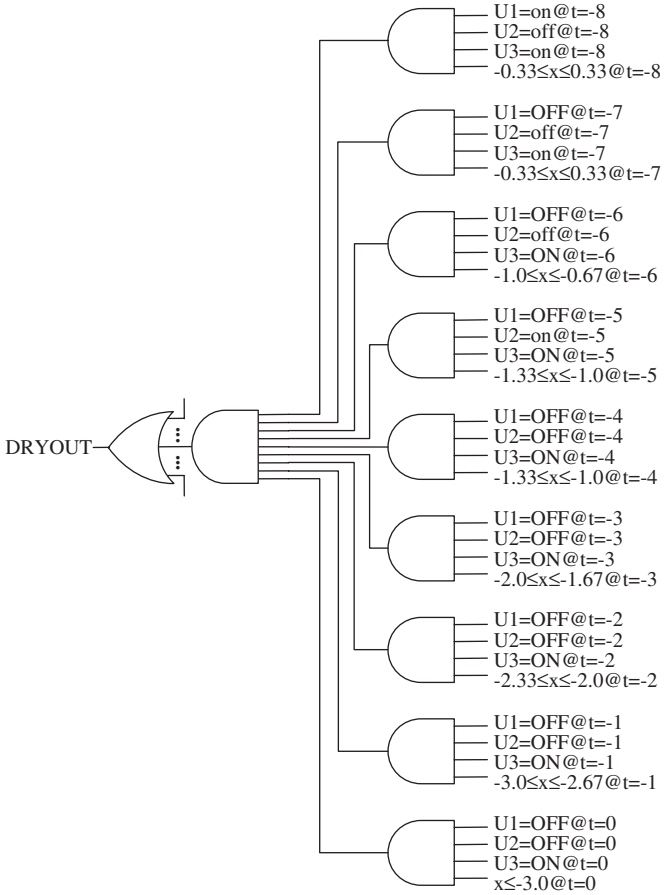


Fig. 7. First-level AND subtree.

This scenario is represented by the fault tree subtree in Fig. 7. As we can see, the first-level AND gate has 9 inputs (there are 9 states in the scenario). Each input is a second-level AND gate capturing one state in the scenario. Each second-level AND has four inputs that completely describe a state of the system, including a time stamp indicating the time at which the system was in the corresponding state. The time stamp allows us to keep track of the basic event ordering information.

The DETs generated from Markov models with this approach may become rather large and complex. However, modern ET/FT analysis software such as SAPHIRE [27] can efficiently solve complex models in a reasonable amount of time.

## 6. Integration into existing PRAs

The DETs generated may be easily incorporated into an ET/FT code such as SAPHIRE. The DETs may be described as AND event sequences, which can be modeled using fault trees. The fault trees may then be entered into SAPHIRE graphically using the fault tree editor. However, for large models it may be easier to construct a text file containing the fault tree logic and import this file into SAPHIRE. Other ET/FT codes such as CAFTA [30] and RISKMAN [31] have similar features.

In order to import the dynamic model into SAPHIRE, it must first be written as a properly formatted text file. A properly formatted file can be read into SAPHIRE using its MAR-D feature [27]. While the file may have any name desired, the file extension to be used will vary based on the type of tree that was generated. Since the model to be imported has been converted to a fault tree, a .FTL extension should be used. To illustrate the format to be used for a .FTL file, a sample fault tree is given below.

```
EXAMPLE, TOP-EVENT =
TOP_EVENT     AND     GATE1 GATE2 GATE3
GATE1         OR      EVENT1 EVENT2 EVENT3
GATE2         OR      EVENT4 EVENT5 EVENT6
GATE3         OR      EVENT7 EVENT8 EVENT9
```

This fault tree top event (TOP-EVENT) is an AND gate with three inputs (GATE1, GATE2, GATE3). Each gate is an OR gate with three separate basic events as inputs. This text format may be used to quickly enter a large fault tree into SAPHIRE. The file is imported through SAPHIRE's MAR-D toolset using the 'import' and 'fault tree logic' options. Fig. 8 below shows a segment of the .FTL file used to import the fault tree logic of the water level controller. Due to the large size of the file, it is not shown in its entirety.

Once the logic has been imported, the model will exist in the SAPHIRE database. Any basic events that have the same name as an existing basic event in the SAPHIRE database will be assumed to be the same event. Otherwise, the basic events will be created and added to the SAPHIRE database as the fault tree is imported. The user will then need to add appropriate failure information to these basic events.

Once the model is added to SAPHIRE, it may then be integrated into an existing PRA within SAPHIRE. Knowledge of the model and the existing plant are required in order to appropriately link the systems. To demonstrate

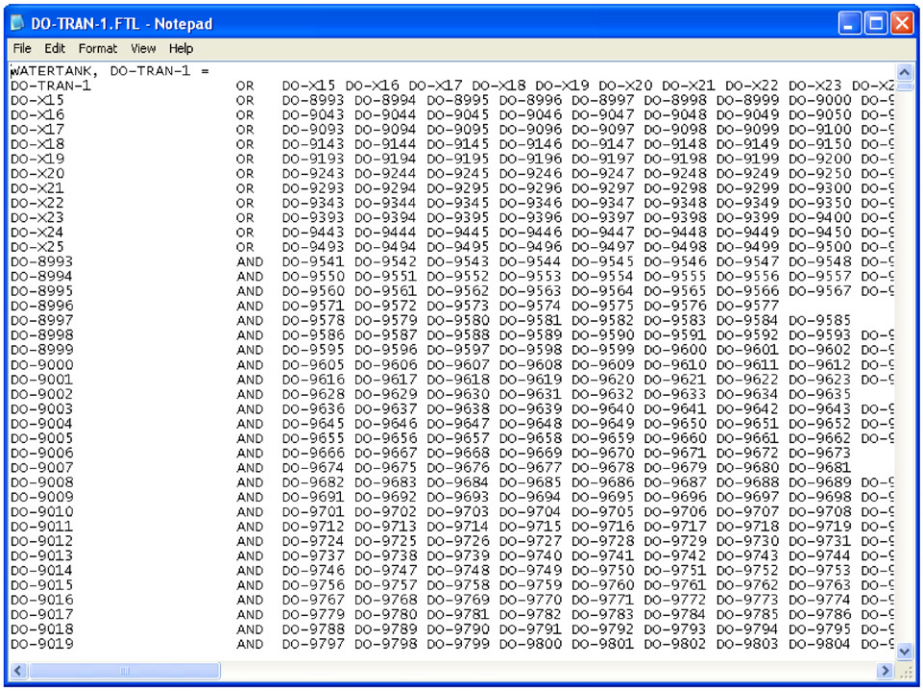


Fig. 8. Segment of .FTL file to import water level controller fault tree.

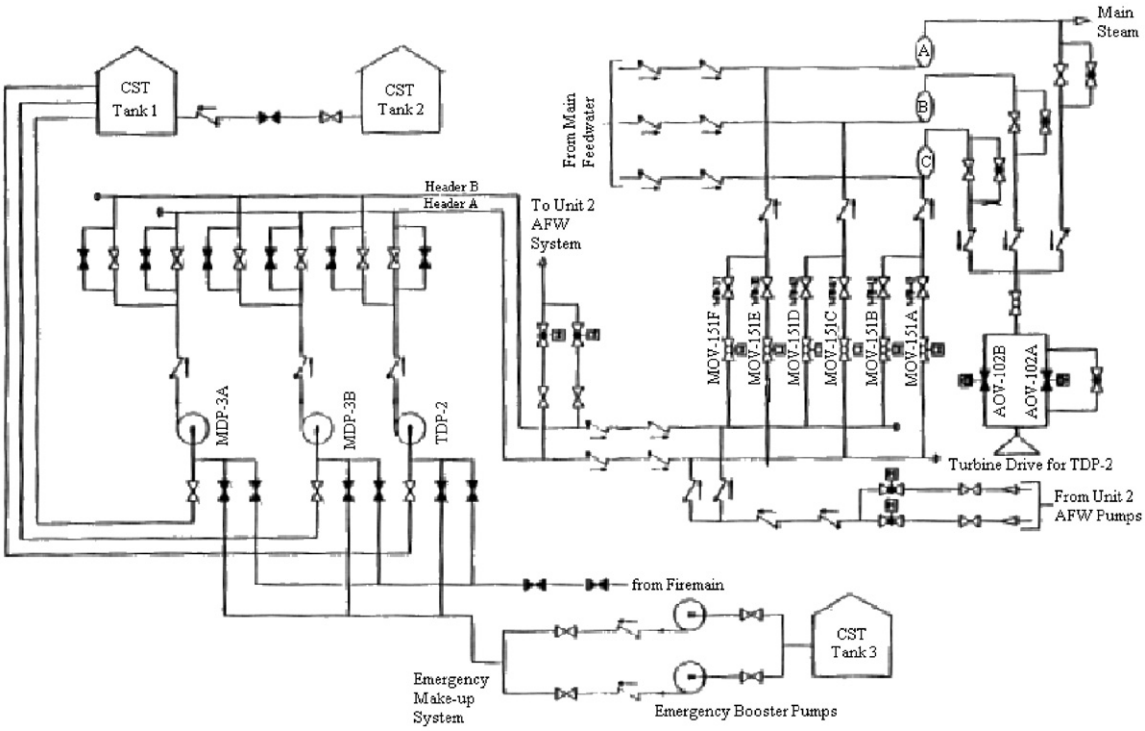


Fig. 9. Example plant AFW system [27].

how the models would be linked, consider the following example.

An example plant based on NUREG-1150 models has been modeled in SAPHIRE. This plant is a three steam-generator plant, as shown in Fig. 9. The auxiliary feedwater system (AFW) has been modeled in detail, with the exception of the control system which has not been included in this model [27]. This model will be linked to the simple level control system (which will be considered to be a feedwater controller). The controller will be tied to one steam generator. For the purposes of this illustration, we will assume that this controller is used during emergency conditions and controls the AFW system for the steam generator. Note that such a controller would not actually be used for the AFW system of a nuclear power plant, but these assumptions have been made due to model availability and this example is intended for demonstration purposes only.

As seen in Fig. 9, the example plant AFW system draws water from two condensate storage tanks (CST), with a third tank in reserve. Three pumps work in parallel to deliver water to the steam generators. Two of these pumps are motor-driven pumps (MDP), and the third is a turbine-driven pump (TDP). Six motor-operated valves (MOV) are used to control the water flow to the three steam generators. The valves MOV-151E and MOV-151F control the water flow to Steam Generator A, valves MOV-151C and MOV-151D control the water flow to Steam Generator B, and valves MOV-151A and MOV-151B control the water flow to Steam Generator C. Finally, two air-operated valves (AOV) control the rate of steam flow out of the steam generators.

The controller is added to the existing plant PRA as shown below in Fig. 10. The new controller (circled) is added as a Transfer gate and inputs to AFW1, insufficient flow to Steam Generator A. The remaining two steam generators are assumed to remain under analog control. Alternatively, additional controllers may be added to the model and linked to these steam generators. Two motor-operated valves, MOV-151E and MOV-151F, lead to Steam Generator A. These two valves will be assumed to be the inflow valves of the water level controller. The steam outflow valve, AOV-102A will be assumed to be the outflow valve from the controller model.

If components are common between the two models, it is important that SAPHIRE recognize this when it is generating the cut sets. Events added from the water level controller model have been time-tagged to retain some dynamic information. Furthermore, these components (such as the valves) are modeled as ‘failed open’ and ‘failed closed’. In the SAPHIRE plant PRA, the same valves may have different failure modes, such as ‘plugged’, ‘failed closed’, and common cause failures. The failure information for the air-operated steam outlet valve is shown in Fig. 11. Ideally, the failure information should not be lost when the controller model is added. This can be accomplished through SAPHIRE’s Recovery Rules feature [27]. Recovery Rules are a SAPHIRE toolset that allow for post-processing and cut set manipulation. They are typically used to add recovery events to the PRA or to

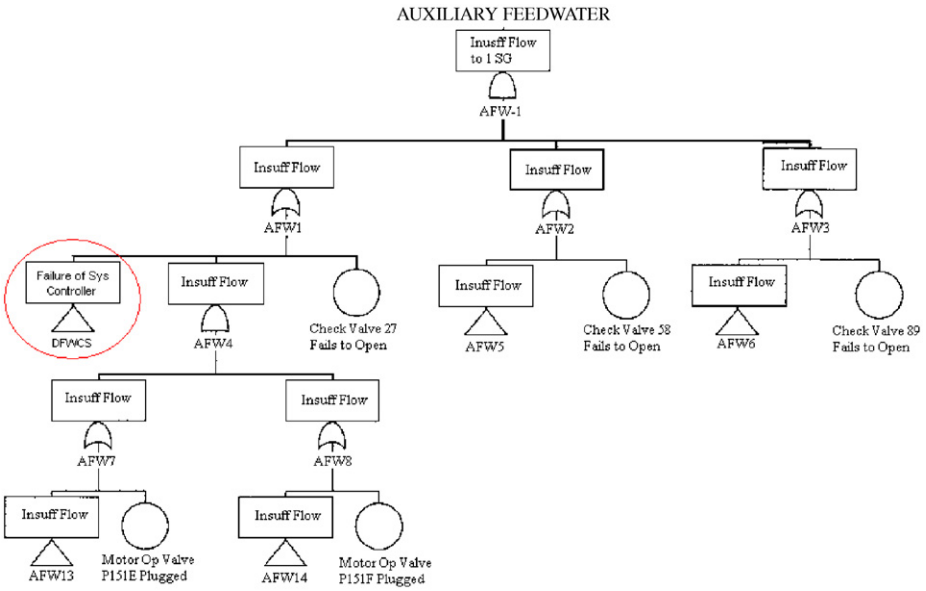


Fig. 10. Linking the model to an existing PRA.

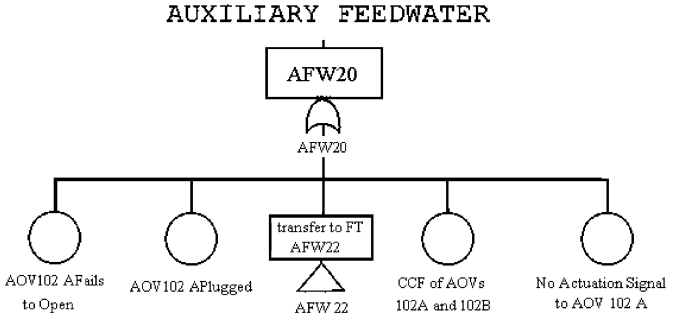


Fig. 11. Failure modes for steam outlet valve AOV-102A.

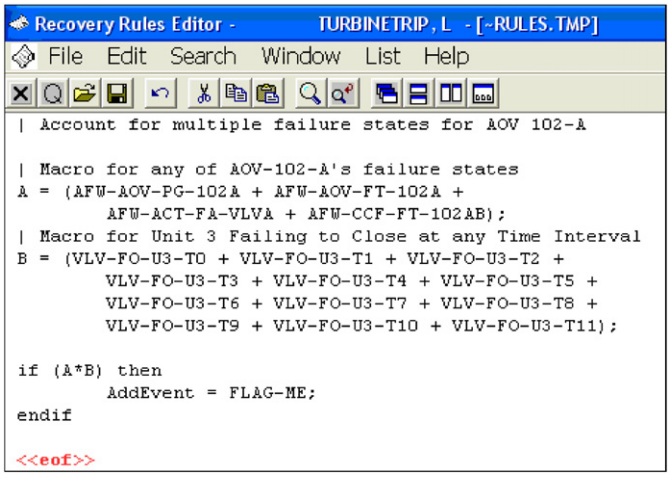


Fig. 12. SAPHIRE recovery rule for multiple failure modes.

remove mutually exclusive or otherwise impossible cut set sequences.

The Recovery Rule shown in Fig. 12 demonstrates how to account for multiple failure modes in both models, as well as time tagging of events. In this rule, 'A' is defined as the occurrence of the AOV-102A valve in any failure mode. 'B' then accounts for the outlet valve from the water level controller in any failed state (and any time tag). This rule will scan every cut set and add a dummy event 'FLAG-ME' to any cut set containing the valve in multiple failure modes.

The analyst can then search the cut sets for this flag and determine if any of these cut sets are inconsistent. For example, an inconsistent cut set might contain the valve in both the 'plugged' failure state and the 'failed open' failure state—which is physically impossible. Any cut sets determined to be inconsistent should then be deleted. Similar Recovery Rules were also written for the inflow valves to the steam generator. These rules allow for as much failure information from both models to be retained, while making it easy to find any inconsistencies that may appear.

## 7. Conclusion

In this paper, we have discussed a new approach to the generation of failure scenarios and their compilation into dynamic event trees and dynamic fault trees from a Markov model of a given system. In particular, we have presented two algorithms for the generation of DETs from a Markov model of the system and how to construct DFTs by traversing the generated DETs. We have also illustrated the use of these algorithms to analyze the behavior of a simple level control system and how the generated DFTs allow the integration of Markov models into existing PRAs using tools such as SAPHIRE [27].

Compared to conventional DET generation, the main strength of the approach presented is that, once the Markov model is generated, all possible failure scenarios for all possible initial conditions can be identified with very

little computational cost. Conventional DET generation techniques would require repetition of the DET generation procedure every time the initial conditions are changed.

The challenge with this approach is to provide a computationally feasible accurate description of the possible behaviors of the system. The generated trees tend to contain a large number of paths. We need to investigate techniques to extract the most useful information from the trees and to prune paths that may not be of interest for the given system and application.

## Acknowledgments

The research presented in this paper was partially supported by a contract from the Idaho National Laboratory (INL). The information and conclusions presented herein are those of the authors and do not necessary represent the views or positions of the INL. Neither the US Government nor any agency thereof, nor any employee, makes any warranty, expressed or implied, or assume any legal liability or responsibility for any third party's use of this information.

## References

[1] Hassan M, Aldemir T. A data base oriented dynamic methodology for the failure analysis of closed loop control systems in process plants. Reliab Eng Syst Saf 1990;27:275–322.
[2] Aldemir T. Quantifying setpoint drift effects in the failure analysis of process control systems. Reliab Eng Syst Saf 1989;24:33–50.
[3] Aldemir T, Siu NO, editors. Guest editorial. Reliab Eng Syst Saf 1996;52:181–4.
[4] Andow PK. Fault trees and failure analyses: discrete state representation problems. Trans IChemE-Chem Eng Res Des 1981;59a:125–8.
[5] March-Leuba J, Caccuci DG, Perez RB. Universality and aperiodic behavior in nuclear reactors. Nucl Sci Eng 1984;86:401.
[6] Siu N. In: Aldemir T, Siu N, Mosleh A, Cacciabue PC, Göktepe BG, editors. Dynamic approaches—issues and methods: an overview. Heidelberg: Springer; 1994. p. 3–7.
[7] Aldemir T, Miller DW, Stovsky M, Kirschenbaum J, Bucci P, Fentiman AW, et al. Current state of reliability modeling methodologies for digital systems and their acceptance criteria for nuclear power plant assessments, NUREG/CR-6901. Washington, DC: US Nuclear Regulatory Commission; 2006.
[8] Aldemir T. Utilization of the cell-to-cell mapping technique to construct Markov failure models for process control systems. In: Apostolakis G, editor. Probabilistic safety assessment and management: PSAM1. New York: Elsevier; 1991. p. 1431–6.
[9] Aldemir T. Computer-assisted Markov failure modeling of process control systems. IEEE Trans Reliab 1987;R36:133–44.
[10] Sharma TC, Bazovsky I. Reliability analysis of large system by Markov techniques. In: Proceedings of the annual reliability and maintainability symposium. New York: IEEE; 1993. p. 260–7.
[11] Lukic YD. Nuclear power plant core-protection-calculator reliability analysis. In: Proceedings of the annual reliability and maintainability symposium. New York: IEEE; 1996. p. 116–20.
[12] Aldemir T, Stovsky MP, Kirschenbaum J, Mandelli D, Bucci P, Mangan LA, et al. Dynamic reliability modeling of digital instrumentation and control systems for nuclear reactor probabilistic risk assessments, NUREG/CR-6942. Washington, DC: US Nuclear Regulatory Commission; 2007.
[13] Aldemir T, Miller DW, Stovsky M, Kirschenbaum J, Bucci P, Fentiman AW, et al. Current state of reliability modeling methodologies for digital systems and their acceptance criteria for nuclear power plant assessments, NUREG/CR-6901. Washington, DC: US Nuclear Regulatory Commission; 2006.
[14] Yucesan E, Jacobson SH. Computational issues for accessibility in discrete event simulation. ACM Trans Modeling Comput Simulation 1996;6:53–75.
[15] Schruben L, Yucesan E. Complexity of simulation models: a graph-theoretic approach. In: Evans GW, et al., editors. Proceedings of the 1993 winter simulation conference. New York, NY: ACM; 1993. p. 641–9.
[16] Krueger D, Kubler F. Intergenerational risk sharing via social security when financial markets are incomplete. Stanford, CA: Stanford Institute for Economic Policy Research; 2001.
[17] Backus D, Routledge B, Zin S. Exotic preferences for macroeconomists. Cambridge, MA: National Bureau of Economic Research; 2004.
[18] Noppen J, Aksit M, Nicola V, Tekinerdogan B. Market-driven approach based on Markov decision theory for optimal use of resources in software development. IEE Proc-Software 2004;151: 85–94.
[19] Bouissou M. Boolean logic driven Markov processes: a powerful new formalism for specifying and solving very large Markov models. In: PSAM6: Proceedings of the sixth international conference on probabilistic safety assessment and management, CD-ROM version. Amsterdam: Elsevier Science Ltd; 2002.
[20] Cojazzi G. The DYLAM approach to the dynamic reliability analysis of systems. Reliab Eng Syst Saf 1996;52:279–96.
[21] Acosta C, Siu N. Dynamic event trees in accident sequence analysis: application to steam generator tube rupture. Reliab Eng Syst Saf 1993;41:135–54.
[22] Kae-Sheng H, Mosleh A. The development and application of the accident dynamic simulator for dynamic probabilistic risk assessment of nuclear power plants. Reliab Eng Syst Saf 1996;52: 297–314.
[23] Izquierdo JM, Hortal J, Sanches-Perea J, Melendez E. Automatic generation of dynamic event trees: a tool for integrated safety assessment. In: Aldemir T, Siu N, Mosleh A, Cacciabue PC, Goktepe BG, editors. Reliability and safety assessment of dynamic process systems, vol. 120. Heidelberg: Springer; 1994. p. 135–50.
[24] Hakobyan A, Denning R, Aldemir T, Dunagan S, Kunsman D. A methodology for generating dynamic accident progression event trees for level 2 PRA. In: PHYSOR 2006 2006;B034:1–9.
[25] Andrews JD, Dugan J. Dependency modeling using fault-tree analysis. In: Proceedings of the 17th international system safety conference. Unionville, VA: The System Safety Society; 1999. p. 67–76.
[26] Cepin M, Mavko B. A dynamic fault-tree. Reliab Eng Syst Saf 2001;75:83–91.
[27] Smith CL, Knudsen J, Calley M, Beck S, Kvarfordt K, Wood ST. SAPHIRE basics: an introduction to probability risk assessment via the systems analysis program for hands-on integrated reliability evaluations (SAPHIRE) software. Idaho Falls, ID: Idaho National Laboratory; 2005.
[28] Belhadj M, Aldemir T. Some computational improvements in process system reliability and safety analysis using dynamic methodologies. Reliab Eng Syst Saf 1996;52:339–47.
[29] Russell S, Norvig P. Artificial intelligence: a modern approach. Englewood Cliffs, NJ: Prentice-Hall; 2003.
[30] CAFTA for Windows, Version 3.0c, SAIC, Los Altos, CA, 1995.
[31] RISKMAN 7.1 for Windows, ABS Consulting, Irvine, CA, 2003.