Available online at www.sciencedirect.com

ScienceDirect

Electronic Notes in Theoretical Computer Science 327 (2016) 27–47

Electronic Notes in Theoretical Computer Science

www.elsevier.com/locate/entcs

Dynamic Fault Trees with Rejuvenation: Numerical Analysis and Stochastic Bounds

Jean-Michel Fourneau

DAVID, Université de Versailles-St-Quentin
Versailles, France

Nihal Pekergin

LACL, Université Paris Est
Créteil, France

Abstract

This paper presents an extension of a methodology that we have introduced recently to analyze Dynamic Fault Trees (DFT). The failure time distributions of the components are obtained from measurements leading to discrete failure time distributions. The gate outputs in a DFT are numerically analyzed from the input failure time distributions. This paper presents one major extension in which maintenance operations are considered such that some components are replaced by new ones to increase the availability of the system.

Keywords: Fault Trees, Stochastic Bounds, Maintenance, Rejuvenation, Discrete Distributions.

1 Introduction

Fault Tree (FT) analysis is a standard technique used in reliability modeling [15]. Dynamic Fault Trees (DFT) are an extension of Fault Trees to model more complex systems where the duration and the sequences of events are taken into account (see for instance [2] or [16] for a presentation). Because of this representation of time and sequence of events, DFTs are much more difficult to analyze than static Fault Trees. The standard numerical approach combines the static analysis based on cut sets for static gates and a Markov chain approach to model dynamic gates. New techniques have been proposed (Monte Carlo simulation [11], process algebra [3]) but there is still a need for some efficient methods of resolution for large and complex DFTs.

Indeed, the Markovian approach leads as usual to the state space explosion which prevents the analysis of large systems while simulations of transient processes need a large number of replications of the sample-path computation to obtain an accurate estimator for the reliability (see [7] and references therein for a statement

http://dx.doi.org/10.1016/j.entcs.2016.09.022

of the problem and [14] for a tool based on the same set of methods). Thus, new resolution methods still have to be investigated.

We have proposed in [6] a numerical procedure to compute the distribution of the time to failure of a DFT or some stochastic bounds of this distribution which relies on the description of component failure processes by some discrete distributions, our model is not Markovian and it is not state-based unlike most of the tools (see [12] for a survey). Here we allow some rejuvenation process for the system components and we extend the numerical methods to take into account this maintenance activity.

Fault Trees are composed of a set of leaves which model the system components and some gates whose inputs are connected to the leaves or to the outputs of other gates. The value of the leaves is a boolean which is True, if the component is down, and False otherwise. The whole topology, when there is no replicated events, must be a tree. The root of the tree is a boolean value which must be True when the system has failed. The fault trees contain 3 types of gates: OR, AND and K/N (K out of N, or voting) gates. All of them are logical gates which are not needed to be presented here.

DFTs also contain four new gates: PAND (priority AND), FDEP (functional dependency), SEQ (sequential failures) and SPARE gates:

- SPARE gate. It is used to represent the replacement of a primary component by a spare with the same functionality. Spare components may fail even if they are dormant but the failure time distribution of a dormant component is lower in some sense than the failure time distribution of the component which is operational. A spare component may be "cold", if it cannot fail while it is dormant, or "hot" if the dormant has the same failure time distribution as an operating one, or "warm" otherwise. CSP, HSP and WSP gates are associated respectively with the cold, hot, and warm spare behaviors.
- FDEP. The FDEP gate has one main input connected to a component or to another gate and it has several links connected to components. When the main input becomes True, all the components connected by the links become True, irrespective of their current values. If a DFT contains a FDEP gate, its topology is not a tree anymore.
- PAND and STRICT-PAND (SPAND). The output of these gates becomes True when all of its inputs have failed in a preassigned order (from left to right in graphical notation). When the sequence of failures is not respected, the output of the gate is False. As discrete time models are considered, a distinction is made between PAND and SPAND when inputs become True at the same time.
- SEQ. The output of the SEQ gate becomes True when all of its inputs have failed in a preassigned order but it is not possible that the failure events occur in another order.

The DFT is represented by a function F (the so-called structure function [8]) which returns True when the system is down and False when it is operational. It is the value carried by the root of the DFT. The analysis of the DFT consists in computing the value of the structure function from the initial time (i.e. 0) up to

the mission time (denoted as MT in this paper).

As in [6], the component failure time is assumed to follow a discrete distribution. We directly use the experimental distributions obtained from the measurements as an input for the reliability of the system components without any fitting to theoretical models. It was shown in [6] how to compute the reliability of the system with some numerical algorithms associated with the gates and a global bottom-up numerical procedure. This work extends this approach to include the rejuvenation of some components with an external maintenance process. Even if there are some works in the literature on evaluation of DFT when the components can be repaired (see for instance [10]), the rejuvenation was not included in the DFT representation. Indeed there is no description of maintenance or repair in the normalized description of a DFT (see [16]).

The paper is organized as follows. Section 2 presents the algorithms to compute numerically the failure time distribution of a DFT or a stochastic bound of this distribution. Section 3 shows how the method can be adapted to consider rejuvenation of some components. In both sections some examples are given to illustrate the methodology. Section 4 is devoted to two examples taken from the literature.

2 Numerical analysis for the distribution of time before failure associated with a DFT

Unlike most of the models in the literature, our model is not state-based and we do not assume exponentially of the transitions to build a Markov chain. Instead, following [6] we model the distribution of the time to failure of the components with discrete random variables taking values in $\mathbb{R}^+$. d_X will denote the probability mass function (PMF) of random variable X and $\mathcal{H}_{d_X} = \{i : d_X(i) > 0\}$ will denote the support of distribution d_X.

The analysis is based on a decomposition into subtrees like many other techniques [5,7,10]. We first assume that the DFT does not contain neither replicated events nor FDEP gates. Therefore, the topology of the DFT is a tree and we only have to consider one tree. The main assumption is the independence of the failure events for the components of the model. Due to the tree topology the input distributions of any gate are independent. The reliability of the system is computed between time 0 and Mission Time (MT). Let $EoT \in \mathbb{R}^+$ be a symbolic time instant called End of Time. Clearly $EoT > MT$. EoT will be used to describe PAND gates.

2.1 Numerical analysis of gates in a DFT without replicated events nor FDEP gates

The independence of the inputs allows us to compute the distribution of the failure time for the output of a gate using the product of probabilities. Each gate $g \in \{$OR, AND, K/N, SEQ, CSP, HSP, PAND, SPAND$\}$ is associated with a function (say f) whose definition is given in Table 1 (for WSP, see [6]). Let I be the arity of f. For instance, the OR gate is associated with the "min" function. Indeed, an OR gate

is faulty when the first fault occurs.

Table 1
Description of the f_g function.

g	$f_g(v_1, \ldots, v_I)$	max output size
OR	$\min\{v_1, \ldots, v_I\}$	$\sum_{i=1}^{I} n_i$
AND	$\max\{v_1, \ldots, v_I\}$	$\sum_{i=1}^{I} n_i$
K/N	$\min\left\{v_i \mid \sum_{j=1}^{I} \mathbb{1}_{(v_j \leq v_i)} \geq k\right\}$	$\sum_{i=1}^{I} n_i$
HSP	$\max\{v_1, \ldots, v_I\}$	$\sum_{i=1}^{I} n_i$
PAND	$\mathbb{1}_{(v_1 \leq \ldots \leq v_I)} v_I + \mathbb{1}_{\neg(v_1 \leq \ldots \leq v_I)} EoT$	$n_I + 1$
SPAND	$\mathbb{1}_{(v_1 < \ldots < v_I)} v_I + \mathbb{1}_{\neg(v_1 < \ldots < v_I)} EoT$	$n_I + 1$
SEQ	$\min\left\{\sum_{i=1}^{I} v_i, EoT\right\}$	$\prod_{i=1}^{I} n_i$
CSP	$\min\left\{\sum_{i=1}^{I} v_i, EoT\right\}$	$\prod_{i=1}^{I} n_i$

Algorithm 1 Computation of the output distribution of gate g.

Input: Input distributions $(d_{V_i})_{1 \leq i \leq I}$ with supports $\mathcal{H}_{d_{V_1}}$,..., $\mathcal{H}_{d_{V_I}}$ with respective sizes $n_1, \ldots, n_I$.

Output: Output distribution d with support $\mathcal{H}_d$.

```
1:  H_d ← ∅;
2:  for all v_1 ∈ H_{d_{V_1}} do
3:      ...
4:      for all v_I ∈ H_{d_{V_I}} do
5:          v ← f_g(v_1, ..., v_I);
6:          if v ∉ H_d then
7:              H_d ← H_d ∪ {v}; d(v) ← 0;
8:          end if
9:          d(v) ← d(v) + ∏_{i=1}^{I} d_{V_i}(v_i);
10:     end for
11:     ...
12: end for
13: return d
```

Algorithm 1 presents the general computation of the output distribution of a gate g. It applies function f_g on all possible combinations of values $(v_i)_{1 \leq i \leq I}$ from the supports $(\mathcal{H}_{d_{V_i}})_{1 \leq i \leq I}$ of the respective input distributions $(d_{V_i})_{1 \leq i \leq I}$, adds the result v to the support of output distribution d and updates its probability $d(v)$ by adding the product of the probabilities $(d_{V_i}(v_i))_{1 \leq i \leq I}$ that correspond to values $(v_i)_{1 \leq i \leq I}$. Note that distribution d returned by Algorithm 1 has support $\mathcal{H}_d$ with size at most the size indicated in the last column of Table 1.

Algorithm 1 needs $\prod_{i=1}^{I} n_i$ evaluations of f and accesses to d. Algorithms with a better complexity are available. The output distributions of gates OR, AND, K/N, HSP, PAND and SPAND can be computed in $\sum_{i=1}^{I} n_i$ steps using a fusion of sorted lists and the output distributions of gates SEQ and CSP are associated with the convolution of discrete distributions which can be solved with a FFT based algorithm with $\sum_{i=1}^{I} n_i \times \log(\sum_{i=1}^{I} n_i)$ steps. The HSP is equivalent to an AND gate, and the CSP gate is equivalent to a SEQ gate. The PAND and the STRICT PAND gates have the same complexity as an AND gate.

Example 2.1 [OR and SEQ gates] Consider an OR gate with inputs d_a and d_b. The output distribution of the OR gate is given as:

d_a	0.25	0.1	0.35	0.15	0.15	
$\mathcal{H}_{d_a}$	0	3	5	9	11	
d_b	0.2	0.2	0.1	0.1	0.1	0.3
$\mathcal{H}_{d_b}$	3	4	5	9	14	16
$\mathrm{OR}(d_a, d_b)$	0.25	0.23	0.13	0.24	0.09	0.06
$\mathcal{H}_{\mathrm{OR}(d_a,d_b)}$	0	3	4	5	9	11

Consider now a SEQ gate with the same inputs d_a and d_b. The output distribution of the SEQ gate is given as:

$\mathrm{SEQ}(d_a, d_b)$	0.05	0.05	0.025	0.02	0.02	0.08	0.095
	0.035	0.04	0.03	0.105	0.03	0.09	0.01
	0.015	0.065	0.015	0.105	0.015	0.06	0.045
$\mathcal{H}_{\mathrm{SEQ}(d_a,d_b)}$	3	4	5	6	7	8	9
	10	12	13	14	15	16	17
	18	19	20	21	23	25	27

This example gives a nice intuition of the size explosion problem. The size of the output distribution may increase considerably at each gate.

Let us now evaluate a simple DFT with a tree topology. This algorithm can also be used to evaluate a subtree. Note that various subtrees will be used in the following to deal with PAND gates, replicated events or maintenance agents.

Algorithm 2 gives the analysis of a DFT without replicated events using a bottom-up approach based on the labels given in step 1. Due to the topological ordering, the distributions are always available when needed.

Algorithm 2 Exact computation of the output distribution of a DFT without replicated events.

Input: DFT T and mission time MT

Output: Output distribution

1: Label the gates using the topological order from the bottom to the top.
2: **for** all gate g in the tree, in the ascending order of the labels **do**
3: Evaluate the output distribution of gate g between 0 and MT according to the gate type and its inputs with Algorithm 1.
4: **end for**

2.2 Numerical analysis with replicated events or FDEP gates

Note that a FDEP gate is equivalent to replicated events as shown in Figure 1. In a first step of the analysis, we replace the FDEP gates by replicated events. Note that the trigger of the FDEP gate may be an event or a subtree, which can be evaluated in a first step.

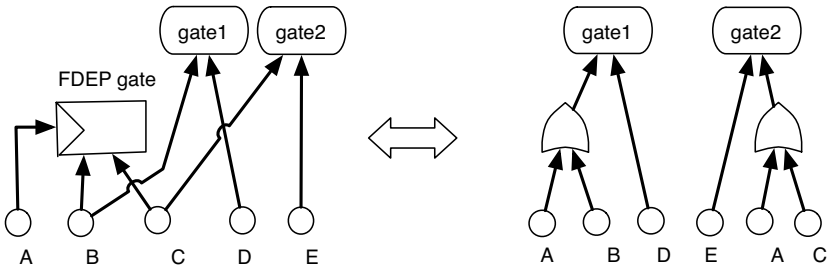

Fig. 1. Equivalence between a FDEP gate and multiple OR gates with duplicated leaves.

When some events are replicated, we cannot assume independence anymore among the input distributions of a gate. Thus, we have to condition on the failure times of the replicated events and use the total probability theorem to get the availability. We call the set of replicated leaves the conditioning set. Let n_R be the number of time instants in the joint distribution of the replicated events. The probability that the system is operational at time t conditioned on the failure instants of the replicated components is computed with Algorithm 1. However, we have to compute n_R such conditional probabilities before computing the reliability. To avoid unnecessary computation, we only consider conditional probabilities when needed. Clearly, it is sufficient to consider a subtree containing all the replicated leaves. Better strategies exist (see [6]).

2.3 Stochastic comparison and stochastic monotonicity: dealing with complexity issues

Assume that the inputs of a gate have distributions d_1 and d_2 with sizes n_1 and n_2. The size of the distribution of the failure time for the output of this gate can be as large as $n_1 \times n_2$. We use the stochastic monotonicity theory to reduce the size of the distributions while proving that they provide upper or lower stochastic

Algorithm 3 Exact computation of the output distribution of a DFT with replicated events.

Input: DFT T, and mission time MT
Output: Output distribution

1: Evaluate the subtrees rooted at the triggers of the FDEP gates with Algorithm 2 (not needed if the trigger is an event).
2: Convert the FDEP gate into replicated events
3: Build the smallest subtree containing all the replicated leaves.
4: **for** all the time instant u when the replicated leaves may fail **do**
5: Use Algorithm 2 to compute the distribution between 0 and MT conditioned on the failure of the replicated leaves at time u.
6: **end for**
7: Use the total probability theorem to analyze the subtree between 0 and MT.
8: Use Algorithm 2 to analyze the remaining tree and compute the probability that the system is operational at time t between 0 and MT.

bounds on the reliability of the system. Formally, for a given distribution d with support $\mathcal{H}_d$ of size $M \in \mathbb{N}$ ($|\mathcal{H}_d| = M$), two bounding distributions are built: d^l with support $\mathcal{H}_{d^l}$ of size $n \in \mathbb{N}$ such that $n < M$ ($|\mathcal{H}_{d^l}| = n$) and d^u with support $\mathcal{H}_{d^u}$ of size n ($|\mathcal{H}_{d^u}| = n$). Note that $\mathcal{H}_{d^l}$ and $\mathcal{H}_{d^u}$ have the same size but they are not necessarily the same set. Let us first define the stochastic comparison and the stochastic monotonicity (refer to Stoyan's book [9] for theoretical issues).

Definition 2.2 Let X and Y be two discrete random variables with d_X and d_Y as PMFs.

$$X \leq_{st} Y \Leftrightarrow \forall i \in \mathcal{H}_{d_X} \cup \mathcal{H}_{d_Y}, \ \sum_{k \geq i} d_X(k) \leq \sum_{k \geq i} d_Y(k)$$

Note that the notations $X \leq_{st} Y$ and $d_X \leq_{st} d_Y$ are interchangeably used in this paper.

Example 2.3 Consider discrete random variables X and Y with:

d_X	0.1	0.2	0.1	0.2	0.05	0.1	0.25
$\mathcal{H}_{d_X}$	1	2.1	3	4.8	5	6	10
d_Y	0.25	0.05	0.1	0.15	0.15	0.3	
$\mathcal{H}_{d_Y}$	2.1	3	4.8	5	6	10	

It can be checked that $d_X \leq_{st} d_Y$. Intuitively, the probability mass of d_Y is concentrated on higher states than that of d_X. In other words the cumulative distribution function (CDF) of X is always larger than the CDF of Y (see Figure 2).

Let us now turn to the stochastic monotonicity property applied to the DFT gates. Intuitively, if such a property holds, then when the inputs increase in the strong stochastic sense (i.e. $\leq_{st}$ ordering), so does the outputs (see reference [6] for proofs).

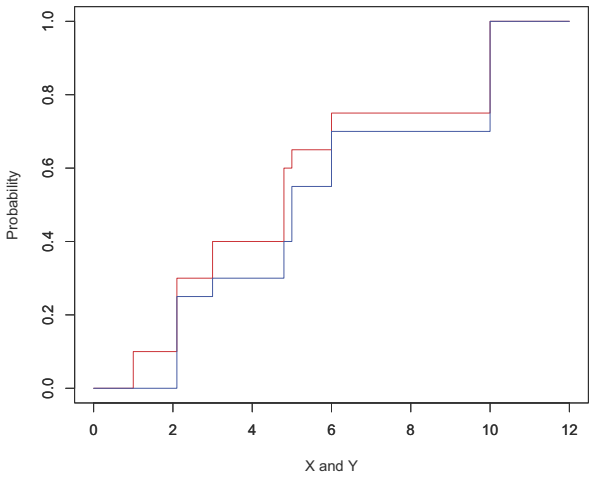

Fig. 2. Comparison of the CDF of Example 2.3 (X in red and Y in blue).

Definition 2.4 Let g be a DFT gate with I independent inputs $(V_i)_{1\leq i\leq I}$ and output Q. Gate g is said to be stochastically monotone, if the inputs $(V_i)_{1\leq i\leq I}$ are upper-bounded by independent bounds $(V_i^u)_{1\leq i\leq I}$, then the output Q^u with inputs $(V_i^u)_{1\leq i\leq I}$ provides the upper bound of the output Q with inputs $(V_i)_{1\leq i\leq I}$:

$$(\forall i, 1 \leq i \leq I : \; V_i \leq_{st} V_i^u) \implies Q \leq_{st} Q^u.$$

Theorem 2.5 *(see Lemma 1 and Lemma 2 in [6]) All the gates are stochastically monotone except for the PAND and the SPAND gates.*

Therefore, if during the evaluation of a gate, its input distributions are replaced by their stochastic bounds, a stochastic bound of the output distribution can be obtained. This property is extremely important to avoid that the sizes of the distributions become too large during the analysis. Let us consider an example to illustrate the problem.

It remains to explain how one can compute stochastic bounds of a discrete distribution. It is a rather intuitive task. A discrete distribution is a step function. A stochastic upper (resp. lower) bound with a smallest support is another step function with less steps and which is below (resp. above) the original one. Some algorithms have been presented in [1] and [6]. They differ by their complexity and the accuracy of the bounds. One of them is optimal according to the difference of rewards between the distributions and the others are faster. All of them provide stochastic bounds of the distribution. The modelers can use any of these algorithms with the ability to choose between the speed or the accuracy of the computation.

Example 2.6 [Monotonicity of the SEQ gate.] The output of SEQ(d_a, d_b) has been given in Example 2.1. Consider the upper bound d_a^u of d_a given in the following table. The output distribution of SEQ(d_a^u, d_b) is:

d_a^u	0.25	0.45	0.3				
$\mathcal{H}_{d_a^u}$	0	5	11				
$\text{SEQ}(d_a^u, d_b)$	0.05	0.05	0.025	0.09	0.115	0.045	0.13
	0.06	0.105	0.045	0.03	0.135	0.03	0.09
$\mathcal{H}_{\text{SEQ}(d_a^u, d_b)}$	3	4	5	8	9	10	14
	15	16	19	20	21	25	27

Figure 3 shows the cumulative probability distributions for $\text{SEQ}(d_a, d_b)$ and $\text{SEQ}(d_a^u, d_b)$. Clearly, $\text{SEQ}(d_a, d_b) \leq_{st} \text{SEQ}(d_a^u, d_b)$.

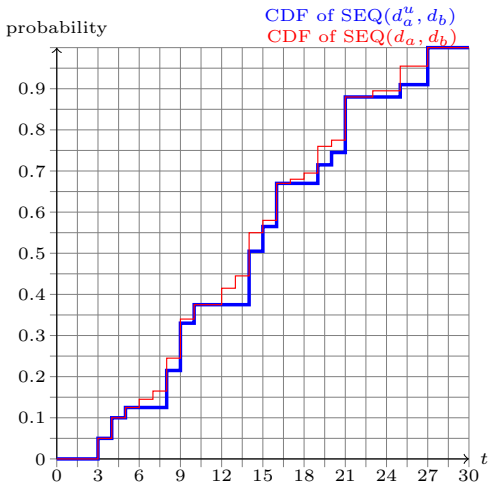

Fig. 3. Monotonicity of the SEQ gate. The exact distribution is in red while the upper bound is in blue.

As most of the gates (except the PAND and SPAND gates) are stochastically monotone, it can be proved that Algorithm 4 provides stochastic bounds for the reliability of a system modeled by a DFT [6]. The size of the bounding distributions is limited to n. Algorithm 4 prevents the bounding of any internal distribution in the subtree rooted with a PAND or SPAND gate which must be evaluated first.

3 Analysis of a DFT with some maintenance processes

We first describe the model for the rejuvenation of the components. Then, we present the numerical algorithm to compute the availability at time t when t varies from 0 to the mission time. Finally, we prove that like our original approach in [6] one can derive stochastic bounds of the availability. Remember that due to the maintenance process, we now consider the availability at time t rather than the reliability at time t.

Algorithm 4 Bounded computation of the output distribution of a DFT without replicated events.

Input: DFT T, $n \in \mathbb{N}$ and mission time MT
Output: Output distribution

1: Build all the subtrees rooted in the PAND gates.
2: **for** all the subtrees rooted in a PAND gate **do**
3: Use Algorithm 2 to compute the distribution at the output of the subtree between 0 and MT
4: If the size of the output distribution is larger than n, use one of the compression methods [1,6] to reduce its size to n and replace the subtree by a leaf with this distribution.
5: **end for**
6: Label the remaining gates using the topological order from the bottom to the top.
7: **for** all remaining gates g in the ascending order of the labels **do**
8: Evaluate the output distribution of gate g with Algorithm 1
9: If the size of the output distribution is larger than n, reduce its size to n.
10: **end for**

3.1 Agents and Model of rejuvenation

We assume that there exist R maintenance agents. They play the same role as the repair blocks in [10] but they are modeled by a set of time instants and a set of leaves rather than a Petri net. Each agent $r, 1 \leq r \leq R$ is described by two sets: $\mathcal{T}_r$, the set of maintenance instants of Agent r, and $\mathcal{L}_r$, the set of the components (leaves) which are maintained by Agent r. Note that $\forall t \in \mathcal{T}_r \ : \ t \leq MT$. Agent r acts as follows:

- At each instant $t \in \mathcal{T}_r$, Agent r replaces each component which belongs to $\mathcal{L}_r$ by a new one. The agents do not take into account if the component is faulty or not before replacing it. We assume that the rejuvenation is instantaneous. As the component is replaced by a new one, the failure time distribution is reset at the maintenance instants.
- Several agents may work at the same time instant: for any agents r and s, it is not required that $\mathcal{T}_r \cap \mathcal{T}_s = \emptyset$.
- Some components may never be replaced and some components may be replaced by several agents. Thus it is not required that the subsets $(\mathcal{L}_r)_{1 \leq r \leq R}$ have to be a partition of the set of leaves.
- Finally, a fault may occur at the same time instant as a maintenance process. Without loss of generality, we assume that the failures take place before the maintenance.

Note that coordinated maintenance policies such as the ones described in [10] can be described by a sequence of agents acting at closed time instants.

We build the set of maintenance time instants: $\mathscr{T} = \cup_{r=1}^{R} \mathcal{T}_r \cup \{MT, 0\}$ and we

index it such that:

$$\mathscr{T} = \{0 = t_0 < t_1 < t_2 < \ldots < t_{p-1} < t_p = MT\}.$$

$\mathcal{S}_i$ will be the set of components replaced at time t_i: $\mathcal{S}_i$ is the union of subsets $\mathcal{L}_j$ for all j such that $t_i \in \mathcal{T}_j$, or more formally: $\mathcal{S}_i = \cup_{t_i \in \mathcal{T}_j} \mathcal{L}_j$ (as there is no intervention at instant $t_0 = 0$, $\mathcal{S}_0 = \emptyset$).

The gates that may be replaced by an agent will be classified as static or dynamic as follows:

Definition 3.1 In this paper, gates {OR, AND, K/N, HSP} are denoted as static while gates {SEQ, WSP, CSP, PAND, SPAND} are defined as dynamic. The FDEP gates are transformed into replication of events as seen previously. Thus, they are not considered in the classification.

We now consider the paths from an arbitrary component (say leaf u) to the root of the DFT. As the leaf may be replicated or may trigger a FDEP gate, more than one path may exist. Thus, in general, we have to consider several paths. Let $\mathcal{P}_u$ be the set of paths $P(u)$ from leaf u to the root of the DFT. A path $P(u)$ is an ordered sequence of gates ending at the root of the DFT. Let $\mathcal{G}_u$ be the set of dynamic gates in $\mathcal{P}_u$.

Definition 3.2 A leaf u is said to be static if along all the paths in $\mathcal{P}_u$, all the gates are static (i.e. $\mathcal{G}_u$ is empty). A leaf u is said to be dynamic if there exists at least one path $P(u)$ which contains at least one dynamic gate.

Adding maintenance agents to a DFT may lead to semantics problems which, to the best of our knowledge, have not been solved in the literature. Thus, we consider in this paper a set of policies which satisfy some consistency constraints.

First, let us define a relation among the leaves of the DFT (or the components of the system). Let $G = (V, E)$ be a graph such that V is the set of leaves of the DFT and $e = (f, h)$ is an edge of G if $\mathcal{G}_f \cap \mathcal{G}_h \neq \emptyset$.

Definition 3.3 A maintenance agent (say j) is said to be dynamic-compliant if the following conditions holds:

- if a dynamic leaf (say i) is in $\mathcal{L}_j$, then all the leaves in the connected component of i in G are also in $\mathcal{L}_j$,
- there is no constraint on the static leaves of $\mathcal{L}_j$.

Remark 3.4 If the DFT has no replicated leaves, the connected component of G are associated with subtrees rooted in some dynamic gates. We say that these dynamic gates and these subtrees are associated with the maintenance agents.

In the following we consider dynamic-compliant agents for maintenance. Intuitively, it means that all the leaves connected to a dynamic gate are either replaced by the agent or ignored. This is the key idea to establish the following lemma.

Lemma 3.5 *Let g be a dynamic gate. After a maintenance by a dynamic-compliant agent at time t_i, the distribution of the output of gate g remains the same (if the leaves in the subtree rooted in g are not replaced) or is shifted by t_i (if the leaves are replaced).*

Proof. Consider the leaves in the subtree rooted in g. As g is dynamic and the agent is dynamic-compliant, the leaves of the subtree are all replaced or none of them are replaced. In the first case, as all the components of the subtree are new at time t_i, the initial distribution of the time to failure for g is now shifted by t_i. In the other case, the distribution of the time to failure for g remains the same. □

Remark 3.6 Our methodology allows us to analyze static Fault Trees without any constraints.

3.2 Computing the availability for a system with rejuvenation

For the sake of readability, we first assume that there is no replicated events in the DFT. Due to these assumptions, the structure of the DFT is a tree. The replication is studied in the next section.

The algorithm proceeds by a loop on time intervals $[t_i, t_{i+1})$ where $t_i \in \mathscr{T}$. At time t_i, components in $\mathcal{S}_i$ are replaced by new ones, and their distributions of time to failure have to be shifted by t_i to reflect the rejuvenation. For static leaves it is sufficient to use Algorithm 2 with updated parameters. But dynamic leaves which are replaced require a distinct method. We analyze the dynamic leaves taking into account the assumptions on the agents and on the decomposition of the DFT into subtrees. As usual once a subtree has been evaluated, it is replaced by a virtual leaf with the associated distribution. In a first step, we evaluate the subtrees associated with the agents. After each replacement, the initial distribution of failures for this subtree is translated by the current time to obtain the new distribution. As all dynamic leaves which are replaced between 0 and MT are analyzed during the subtree evaluation, the remaining DFT only contains static leaves and dynamic leaves which are never replaced by the agents.

Let d_c be the initial time to failure distribution for leaf c (component c). We compute at each time interval $[t_i, t_{i+1})$, a distribution e_c^i which results from the history of the component between 0 and t_i. Concerning the leaves of the remaining DFT, we clearly have three cases:

- They are static and they have been replaced at time t_i. Their distribution is shifted by t_i. Let $\otimes$ denote the convolution of distributions:

$$e_c^i = d_c \otimes t_i.$$

- They have not been replaced at time t_i and they are static, theoretically we do not have to change their distributions. However in order to speed up the computation time, we modify distribution e_c^{i-1} to minimize the size of the support while being sure that the computations on the interval $[t_i, t_{i+1})$ are still exact. In fact for static gates, at a given instant, one needs to know if inputs are faulty or not and

the temporal order of failures is not important. Thus it is possible to keep as information only the probability that a fault has been occurred before t_i. More formally, operator ϕ is applied on distribution e_c^{i-1} at time t_i as follows: Let $\alpha(t_i)$ be the biggest time instant in the support of e_c^{i-1} such that $\alpha(t_i) < t_i$. The part of the distribution before t_i is aggregated on instant $\alpha(t_i)$: the support of the distribution now begins at $\alpha(t_i)$ and the probability at time $\alpha(t_i)$ is the summation of the probability in e_c^{i-1} before t_i. Clearly such a modification of the distribution decreases the size of its support. Therefore it leads to a faster computation when Algorithm 2 is called. The distributions computed between 0 and t_i may be false before $\alpha(t_i)$ but at time t_i the failures have occurred with the correct probabilities and the computations for time interval $[t_i, t_{i+1})$ correctly take into account the aging process for the non replaced components.

$$e_c^i = \phi(e_c^{i-1}, \alpha(t_i)).$$

- They are dynamic and they will never been replaced, we do not change the distribution:

$$e_c^i = e_c^{i-1}.$$

Example 3.7 Consider the following input distribution d. Let us compute $\phi(d, 7)$. Clearly, $\alpha(7)$ is 6, so:

d	0.11	0.22	0.01	0.06	0.03	0.14	0.08	0.04	0.07	0.24
$\mathcal{H}_d$	1	3	4	5	6	8	9	14	15	16
$\phi(d,7)$					0.43	0.14	0.08	0.04	0.07	0.24
$\mathcal{H}_{\phi(d,7)}$					6	8	9	14	15	16

The main problem of the evaluation of the DFT is that, due to the dynamic gates, one cannot evaluate the distribution in time interval $[t_i, t_{i+1})$ because one must take into account the events before t_i (remember for instance the semantics of a PAND gate). Therefore, one must build the distribution from time 0 while only time interval $[t_i, t_{i+1})$ is required to be computed. Let us now present the algorithm.

Theorem 3.8 *Algorithm 5 computes the point availability of a DFT with maintenance by dynamic compliant agents.*

Proof. At each step, Algorithm 2 is used to get the distribution between 0 and t_i to obtain the result for time interval $[t_{i-1}, t_i)$. Indeed, one must compute the distribution between 0 and t_i to take into account the aging of the components which have not been replaced and whose behavior in $[t_{i-1}, t_i)$ depends on the past. For all cases, the output distribution of failure time in $[0, t_{i-1})$ is irrelevant but it is needed to insure the correctness of the result for time interval $[t_{i-1}, t_i)$.

We still have to prove that as the maintenance agents are dynamic-compliant, the evaluation of the availability in time interval $[t_{i-1}, t_i)$ is correct. Basically, Lemma 3.5 explains how to compute the distribution for the subtree rooted in a dynamic

Algorithm 5 Exact computation of the point availability of a DFT without replicated events and with maintenance agents.

Input: DFT T, a set of time instants $\mathscr{T}$ and a list of sets $(\mathcal{S}_i)_{0\leq i<p}$
Output: Point availability a

1: **for all** subtrees rooted in the dynamic gates (say g) associated with all the maintenance agents **do**
2: Compute d_g the distribution of the time to failure for gate g using Algorithm 2 with MT as mission time and d_c as input distributions for the leaves in the subtree
3: **end for**
4: Copy all the distributions d_c to e_c.
5: Use Algorithm 2 with t_1 as mission time and e_c as input distributions to build the failure time distribution for interval $[0, t_1)$.
6: **for** $i \leftarrow 1$ to $p-1$ **do**
7: **for all** subtrees rooted in the dynamic gates (say g) associated with the maintenance agents operating at time t_i **do**
8: shift d_g by t_i to represent the rejuvenation of the subtree: $e_g = t_i \otimes d_g$.
9: **end for**
10: **for all** component $c \in \mathcal{S}_i$ in the remaining DFT once the subtrees have been evaluated **do**
11: **if** c is static **then**
12: Replace e_c by $t_i \otimes d_c$.
13: **end if**
14: **end for**
15: **for all** $c \notin \mathcal{S}_i$ **do**
16: **if** c is static **then**
17: Replace e_c by $\phi(e_c, \alpha(t_i))$.
18: **end if**
19: **end for**
20: Use Algorithm 2 with t_{i+1} as mission time and input distributions e_c (for leaves) or e_g (for subtrees rooted in the dynamic gates associated with the maintenance agents).
21: Extract from the distribution the part between t_i and t_{i+1}, and use it the compute the point availability a for the same time interval.
22: **end for**

gate associated with the agents. The output distributions for these subtrees are computed with Algorithm 2. The subtrees are replaced by virtual leaves associated with these distributions.

Then, we have to deal with the remaining tree which by construction only contains static leaves and dynamic leaves which are never replaced between 0 and MT. Basically, we now consider a virtual system with distributions e_c^i for component c. If a component (say c) is replaced, its distribution becomes d_c shifted by t_{i-1}. Otherwise the distribution does not have to change (we show at the end of this proof that Instruction 17 does not change the result).

If a replaced component is static, its distribution is only used for the evaluation of static gates. We compute the distribution for the tree associated with these gates by Algorithm 2 between time 0 and t_i using the shifted distributions for the replaced components and the previously modified (or original) distributions for the components which are not replaced. Finally, remark that:

- Instruction 18 only modifies the distributions for static gates on time interval $[0, t_{i-1})$ while giving a correct probability of being operational at time t_{i-1}. As the gates are static, this will insure that the distributions in $[t_{i-1}, t_i)$ are correctly defined.
- In Instruction 2, MT is an upper bound of the time duration. It is possible to improve this bound.

□

3.3 Replication of events

The replication of events does not require any new techniques, even if the model may be more complex to describe. The classification of leaves as static or dynamic is done after we have replaced the FDEP gates with replication of events. We have to consider two cases according to the interaction between the maintenance agents and the events which are used for the conditioning in Algorithm 3.

- if the agent does not replace any of the components which are in the conditioning set, we use a new version of Algorithm 5 where calls to Algorithm 2 in Instructions 2 and 21 are replaced to calls to Algorithm 3 to accommodate the replication of events. Note that Instructions 1 to 3 of Algorithm 3 have to be executed only once as the tree and the conditioning distribution do not change with the maintenance operation.
- if a component in the conditioning set is replaced by the agent at time t_i, we have to restrict ourselves:
  - · if one of the component of the conditioning set is replaced by the agent, then all the components in the conditioning set are replaced at the same time.
  - · if the components which are replaced are the leaves of a subtree rooted in the trigger of a FDEP gate, then this subtree only contains static gates.

  We proceed as follows:

(i) shift the distributions of replaced component by t_i,
(ii) evaluate the new conditioning distribution,
(iii) proceed as in the previous case, using the new conditioning distribution.

Note that due to the assumptions, the support of the conditioning distribution begins at a time instant larger than t_i.

3.4 Stochastic monotonicity and rejuvenation

This subsection shows that the reduced-size stochastic bounds of a DFT with rejuvenation can be also derived using the same methodology. For a DFT without rejuvenation the relationship between the output distribution of its root and the

availability at time t, $\mathcal{A}(t)$ of the underlying system is given as follows: the availability of the system at time t corresponds to the probability that the root of the DFT is FALSE at time t. Recall that the output of a DFT without rejuvenation is a discrete random variable Q representing the time to failure of the DFT. Then the availability of the DFT is:

$$\forall t, \;\; \mathcal{A}(t) = 1 - \sum_{\tau \in \mathcal{H}_Q | \tau \leq t} d_Q(\tau)$$

It follows from the properties of the $\leq_{st}$ order that if the output distributions are comparable, then the corresponding availabilities are also comparable [9].

$$Q \leq_{st} Q^u \implies \mathcal{A}(t) \leq_{st} \mathcal{A}^u(t)$$

In order to derive bounds for the systems with maintenance, we assume that the maintenance dates are the same for the original and the bounding models. The analysis of the DFTs with rejuvenation can be done through Algorithm 5 by analyzing a DFT without rejuvenation between consecutive maintenance dates. Therefore by considering bounding systems in each time interval, a global bound can be computed for DFTs with rejuvenation. Note that it is not necessary to use the same size parameter for each time interval.

4 Examples

In this section two examples are presented. The first one is the so called Hypothetical Example Computer System (HECS) introduced in [13] for the reliability of fault tolerant computer based systems. In Figure 4, the block diagram of subsystems of HECS is given. The DFTs of the subsystems (memory module, processing module, bus system, application/interface modules) are presented in Figures 5 and 6. Note that if any of the four subsystems fails, the HECS will fail. Memory module consists of five memory units $(M1, M2, M3, M4, M5)$ which are connected to the bus system via two memory interface units $(MIU1, MIU2)$. $M1$ and $M2$ (resp. $M4, M5$) are associated with $MIU1$ (resp. $MIU2$), while $M3$ is connected to both interface units for redundancy. There must be at least three memory units with their associated interfaces. In the processing module (S_1), the processors $A1$, $A2$ and A are identical. $A1$ and $A2$ are active and if one of them fails, it is replaced by the cold spare A. The processing module thus the HECS fail when all processors fail. There are two identical redundant bus $(B1, B2)$ in the bus module. The three application/interfaces $C1, C2, C3$ must be all available.

We assume that the maintenance agents act on the components of module S_4 (application/interface module). The mission time is fixed at instant $MT = 500$ and it will be assumed that $EoT = 501$. Let d_1, d_2 and d_3 be the respective failure time distributions of components C_1, C_2 and C_3. We consider two maintenance agents. Agent 1 acts on components C_1 and C_3 by replacing both of them by new ones at instants $t_1 = 200$ and again at $t_3 = 400$. Agent 2 replaces component C_2 at instant

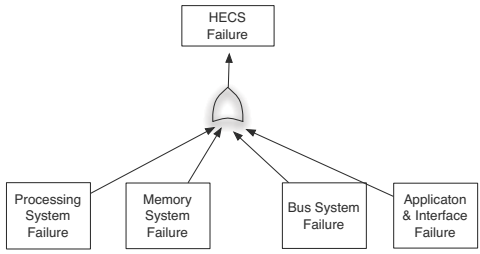

Fig. 4. Hypothetical Example Computer System.

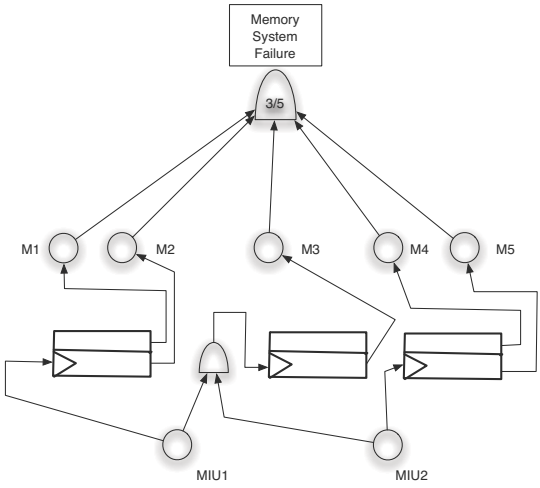

Fig. 5. Memory module (S_2).

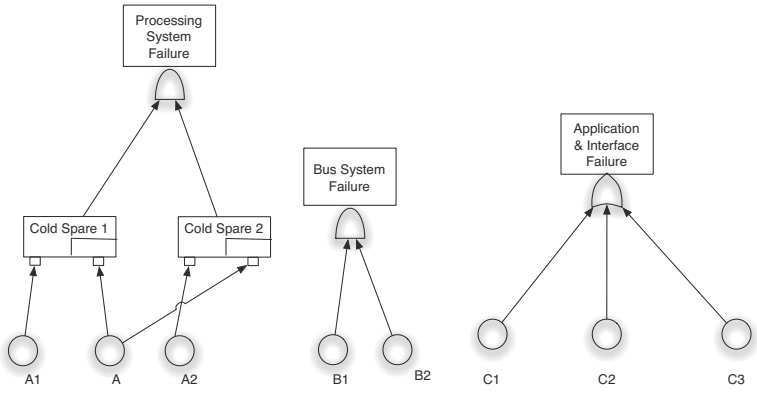

Fig. 6. Processing module (S_1), Bus system (S_3) and Application/Interface modules (S_4).

$t_2 = 280$. Thus $\mathcal{T}_1 = \{t_1, t_3\}$, $\mathcal{L}_1 = \{C_1, C_2\}$, and $\mathcal{T}_2 = \{t_2\}$ and $\mathcal{L}_2 = \{C_2\}$. As subsystems S_1, S_2, and S_3 are never replaced, their distributions of time before failure are computed using Algorithm 2 in a preliminary step. However the analysis of subsystem S_4 must be performed for time intervals according to the interventions of maintenance agents. For a given time interval, the output distribution of the whole system is then computed by means of the already computed output distributions

of subsystems S_1, S_2, S_3 and the output distribution of subsystem S_4 computed with input distributions taking into account rejuvenations during this time interval. This example requires the analysis of four time intervals ($p = 4$):

$[0, t_1)$: all the components start working together, and the output distribution of subsystem S_4 in this interval is calculated by applying Algorithm 2.

$[t_1, t_2)$: due to the intervention of Agent 1 at t_1, components C_1 and C_3 are replaced with new ones and their behaviors are described by $t_1 \otimes d_1$ and $t_1 \otimes d_3$. The output distribution of subsystem S_4 is computed by the updated input distributions for C_1, C_3 and and the global distribution in this interval is calculated by combining these distributions with C_2. The result is a distribution that covers the interval $[0, t_2)$ but only the part between t_1 and t_2 is considered.

$[t_2, t_3)$: due to the intervention of Agent 2 at t_2, component C_2 is replaced with a new one whose behavior is described by $t_2 \otimes d_2$. The output distribution of S_4 in this interval is calculated by combining it with the distributions of $t_1 \otimes d_1$ and $t_1 \otimes d_3$ respectively for components C_1 and C_3. The result is a distribution that covers the interval $[0, t_3)$ but only the part between t_2 and t_3 will be considered.

$[t_3, MT]$: due to the intervention of Agent 1 at t_3, the components C_1 and C_3 are replaced with new ones and their behaviors are described by $t_3 \otimes d_1$ and $t_3 \otimes d_3$ and the output distribution of S_4 in this interval is calculated by combining them with the distribution of $t_2 \otimes d_2$ for component C_2. The result is a distribution that covers the interval $[0, MT]$ but only the part between t_3 and MT is considered.

Figure 7 depicts the availability of the HECS with time. Note that the availabilities are the same before the first intervention.

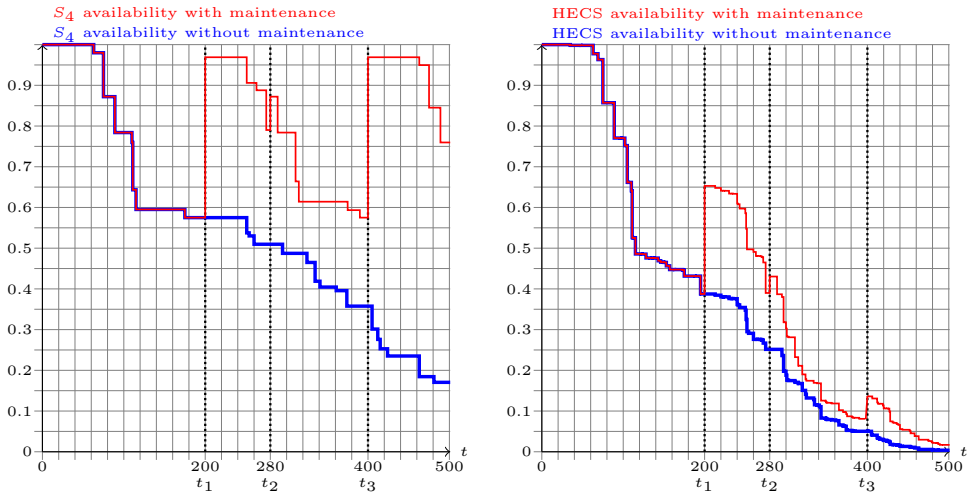

Fig. 7. Availability of subtree $S4$ (left) and availability of the HECS system with/without maintenance (right).

4.1 A more complex example

We consider the cardiac assist system (CAS) illustrated in Figure 8 taken from [4]. This system consists of three separate modules (i.e. CPU, motor, and pump), and

the fault in any of the subsystems yields to the unavailability of the CAS system (OR gate on the top level). In the CPU unit, the primary (P) and the backup (B) CPU units are subject to the common failure coming from the trigger unit (OR gate). The failure of the crossbar switch (CS) or the system supervision (SS) triggers the fault making the CPU unit out of work (CPU FDEP). In the motor section, primary motor unit MA has a cold spare MB which can be activated by means of the switch MS. Due to the PAND gate, if MS fails before MA, the motor unit fails when MA fails. On the other case, since the MB has been already turned on, the failure of MS is not taken into account, and the motor unit fails due to the failure of the MB. There are two primary pumps PA and PB sharing a cold spare PS. The pump system and thus CAS system become unavailable when all pumps become unavailable.

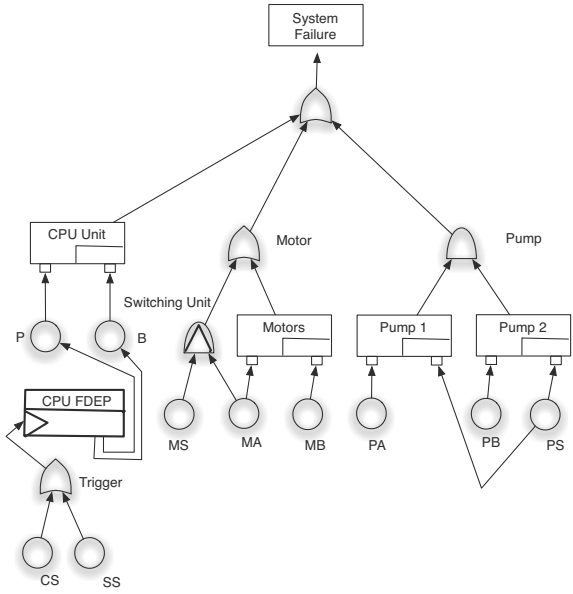

Fig. 8. Cardiac Assist System.

We assume that there exist one agent which repairs the CPU subsystem (the processors, the crossbar and the system supervision). Note that the agent satisfies the assumptions on the replicated events in section 3.3. The agent operates at time 200, 280 and 400. The evolution of the availability with time is depicted in Fig. 9.

5 Conclusion

This paper extends the numerical analysis based on stochastic bounds of a DFT to take into account some maintenance process. By considering discrete random variables for component failure time distributions, algorithms were given to compute the overall point availability with respect to time. For safety critical systems it is important to determine maintenance policies to guarantee the required availability. The proposed methodology can be generalized to find the best maintenance dates to

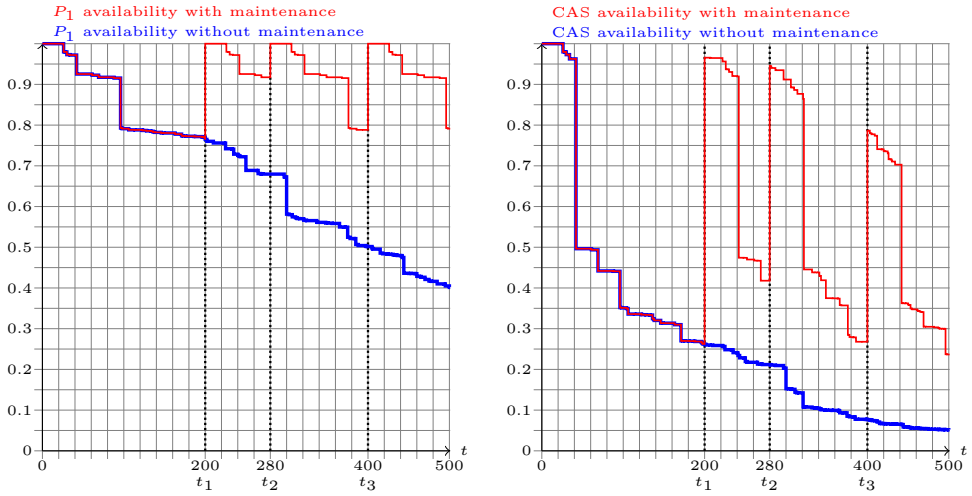

Fig. 9. Availability of CPU subsystem (left) and availability of the CAS system with/without maintenance (right).

reach the maximum availability of the system during a period of time. The method has been implemented in a software tool which will be available soon.

Acknowledgement

This work was partially supported by grant ANR MARMOTE (ANR-12-MONU-0019). We thank M. Dahmoune and D. Vekris for their helpful comments.

References

[1] F. Aït-Salaht, H. Castel-Taleb, J. Fourneau, and N. Pekergin. Stochastic bounds and histograms for network performance analysis. In *Computer Performance Engineering - 10th EPEW Italy*, volume 8168 of *LNCS*, pages 13–27. Springer, 2013.

[2] J. Bechta Dugan, S. J. Bavuso, and M. Boyd. Dynamic fault-tree models for fault-tolerant computer systems. *Reliability, IEEE Transactions on*, 41(3):363–377, Sep 1992.

[3] H. Boudali, P. Crouzen, and M. Stoelinga. Dynamic fault tree analysis using input/output interactive markov chains. In *The 37th Annual IEEE/IFIP International Conference on Dependable Systems and Networks, DSN, UK*, pages 708–717. IEEE Computer Society, 2007.

[4] H. Boudali, A. P. Nijmeijer, and M. I. A. Stoelinga. Dftsim: A simulation tool for extended dynamic fault trees. In *Proceedings of the 2009 Spring Simulation Multiconference*, SpringSim '09, pages 1–8. Society for Computer Simulation International, 2009.

[5] Y. Dutuit and A. Rauzy. A linear-time algorithm to find modules of fault trees. *IEEE Transactions on Reliability*, 45(3):422–425, 1996.

[6] J.-M. Fourneau and N. Pekergin. A numerical analysis of dynamical fault trees based on stochastic bounds. In *12th Int Conference on Quantitative Evaluation of Systems, QEST15*. IEEE, 2015.

[7] R. Manian, J. Bechta Dugan, D. Coppit, and K. Sullivan. Combining various solution techniques for dynamic fault tree analysis of computer systems. In *High-Assurance Systems Engineering Symposium, 1998. Proceedings. Third IEEE International*, pages 21–28, Nov 1998.

[8] G. Merle, J.-M. Roussel, and J.-J. Lesage. Algebraic determination of the structure function of dynamic fault trees. *Rel. Eng. & Sys. Safety*, 96(2):267–277, 2011.

[9] A. Muller and D. Stoyan. *Comparison Methods for Stochastic Models and Risks*. Wiley, New York, NY, 2002.

[10] D. C. Raiteri, G. Franceshinis, M. Iacono, and V. Vittorini. Repairable fault tree for the automatic evaluation or repair policies. In *Int. Conf. on Dependable Systems and Networks, DSN'04*. IEEE, 2004.

[11] K. D. Rao, V. Gopika, V. S. Rao, H. Kushwaha, A. Verma, and A. Srividya. Dynamic fault tree analysis using monte carlo simulation in probabilistic safety assessment. *Reliability Engineering & System Safety*, 94(4):872 – 883, 2009.

[12] E. Ruijters and M. Stoelinga. Fault tree analysis: A survey of the state-of-the-art in modeling, analysis and tools. *Computer Science Review*, 15:29–62, 2015.

[13] M. Stamatelato and W. Vesely. Fault tree handbook with aerospace applications, nasa, 2002.

[14] K. J. Sullivan, J. B. Dugan, and D. Coppit. The Galileo fault tree analysis tool. In *Digest of Papers: FTCS-29, The Twenty-Ninth Annual International Symposium on Fault-Tolerant Computing, USA*, pages 232–235. IEEE Computer Society, 1999.

[15] K. S. Trivedi. *Probability and Statistic with Reliability, Queueing and Computer Science Applications, Second Edition*. Wiley, 2002.

[16] W. Vesely, F. Goldberg, N. Roberts, and D. Hassl. Fault tree handbook, nureg-0492, technical report, united states nuclear regulatory commission, 1981.