Fault tree analysis of software-controlled component systems based on second-order probabilities

Marc Förster, Mario Trapp
Fraunhofer-Institut für Experimentelles Software-Engineering (IESE)
Fraunhoferplatz 1
67663 Kaiserslautern, Germany

{marc.foerster | mario.trapp}@iese.fraunhofer.de

Abstract — Software is still mostly regarded as a black box in the development process, and its safety-related quality ensured primarily by process measures. For systems whose lion share of service is delivered by (embedded) software, process-centred methods are seen to be no longer sufficient. Recent safety norms (for example, ISO 26262) thus prescribe the use of safety models for both hardware and software. However, failure rates or probabilities for software are difficult to justify. Only if developers take good design decisions from the outset will they achieve safety goals efficiently. To support safety-oriented navigation of the design space and to bridge the existing gap between qualitative analyses for software and quantitative ones for hardware, we propose a fault-tree-based approach to the safety analysis of software-controlled systems. Assigning intervals instead of fixed values to events and using Monte-Carlo sampling, probability mass functions of failure probabilities are derived. Further analysis of PMF lead to estimates of system quality that enable safety managers to take an optimal choice between design alternatives and to target cost-efficient solutions in every phase of the design process.

Embedded systems, fault tree analysis, probability intervals, safety, simulation, software.

ACRONYMS & ABBREVIATIONS

(A)SIL (Automotive) safety integrity level
BDD Binary decision diagram
(C)FT (Component) fault tree
FIT Failures in time (10^9/hi)
FM Failure mode
FMEA Failure modes and effects analysis
FTA Fault tree analysis
MCS Minimal cut set
PI Prime implicant
PMF Probability mass function
s- Stochastically
SD Standard deviation

I. INTRODUCTION

The contribution of software to the functionality of embedded systems is ever increasing. Nevertheless, software components are still mostly regarded as a black box by the development and safety process, and their safety-related quality ensured mainly by process measures. For systems whose lion share of service is delivered by means of (embedded) software, process-centred methods are seen to be no longer sufficient [44]. Safety norms (for example, the forthcoming ISO 26262 [26], the MISRA guidelines [32]) substantiate this paradigm shift by prescribing the use of safety models and analyses both for hardware and software. Typically, this is a logical composition/decomposition, combining inductive (FMEA) and deductive (FTA) procedures.

The failure behaviour of hardware alone on the one side and software-controlled systems on the other is very different. Since software represents a behaviour specification, wrong behaviour reveals systematic faults. Software does not fail randomly but will invariably fail again in the same way under the same conditions. While for mass-produced hardware parts it is possible to assign an (exponentially distributed) failure probability with a rate of, for example, 40.17 FIT, for software a similar assumption does not seem entirely realistic. Approaches to alleviating the problem have been developed in the field of software metrics, trying to predict software quality from measurable attributes of resources, processes and software development artifacts themselves [16] [17]. Many of them, however, still treat software as a black box with unknown relation between incoming, internal and outgoing failures. Software fault trees [12] [22] [30] occupy a sort of middle ground here, in that they do capture failure dependences by combinational propositions but abstract from the intricacy of state-based behaviour (which, for example, [19] [20] explicitly model).

This article presents an approach to the safety analysis and optimisation of safety-critical and software-controlled systems by evaluating FT models with a large number of random samples from probability intervals instead of just one concrete probability value. During parts of the development process it will not even be totally clear how functions will be distributed among hard- and software parts. Our method deliberately employs uncertainty about eventual system design and the failure behaviour of system components to arrive at quantitative propositions about failure probability that are more realistic than traditional ones, and at the same time more instructive. It extends related work by indicating how to use second-order probability distributions to characterise model properties and by demonstrating cost optimisation with sample subset analysis.

The essential objective is to support the shifting of safety models and analyses from the post mortem stage to the very beginning of the design phase, promoting them to first-class artifacts and using them as the backbone of a safety-oriented development process.

This work was funded by the German Federal Ministry of Education and Research (BMBF) in the context of the ViERforES project (01 IM08003 B).
Section II gives an introduction to the compositional type of fault trees we use to model and analyse embedded component systems; section III explains basics of interval arithmetic for Boolean functions and gives the details on experimentation setup; section IV describes basic measures we derive from models and analyses and introduces diagrams of top event distributions, interval-based importance measures and different random sampling methods; in section V we propose a safety optimisation technique for software-controlled systems, based on fault trees with interval-valued variables; section VI reviews related work, and section VII concludes our findings and summarises ongoing and future activities.

II. FTA FOR COMPONENT SYSTEMS

In a top-down, component-oriented development process, every design decision on one level critically determines the range of choices available on all subordinated hierarchy levels. Some decisions may be delayed until their consequences become clearer, but must have to be made in the face of uncertainty. Only if developers take good decisions from the beginning will they achieve their safety goals effectively and efficiently. To successfully navigate the potentially infinite design space, at any point in this process we would like to know the answer to the question ‘If I take this decision rather than that, what is the likelihood of the system/component/service turning out “good”?’. If goodness is defined as the probability of the system, component or service not failing or the analysed failure mode not occurring, answering the question requires the determination of the probability of probabilities, that is, second-order probabilities.

A. The component fault tree concept

Our understanding of the component concept closely follows [3]. Thus, components are seen as not just the deployable results of a development process but abstract entities and first-class development artifacts in their own right. During system development, components serve as specifications for their successive refinement/realisation by a set of lower-level (sub)components/specifications [13] [14]. This hierarchical, iterative-incremental process is repeated until, finally, specifications are fine-grained enough to be automatically transformed to machine-executable specifications.

Complementary to functional component models, there are non-functional ones, such as, for safety-critical systems, failure models to demonstrate and analyse safety. For this purpose, we use a FT variant, component fault trees [27], for failure models. They possess, in principle, the same expressive power as conventional FT but are structurally adapted to the component concept. Components are most often influenced by and dependent upon other components, thus, they cannot usually be mapped to a basic event or an independent FT branch (module), as it is traditionally done. For this reason, CFT allow the association of arbitrary connected subsets of FT nodes with entities of the modelled system [27].

Figure 1 shows the abstract view of a CFT model (an entire model is, in fact, a fault forest; here, just one tree for one output failure mode is shown; for gates as well as for components and subcomponents, black nodes represent outports while white nodes represent inports; (sub)component ports may be grouped, indicated by a surrounding frame, representing different FM of a signal). The top-level component A with FM/top event TOP is realised by four subcomponents, B, C, D and E, which are interconnected according to the data/failure flow. The CFT of subcomponents B, C, D are shown in a glass box view. The failure flow between components is communicated via FM-typed ports which can be grouped into higher-order failure modes (for example, ports 2, 6 and 10). Just as the specifications of B, C and D are part of A’s realisation, they in turn will be further broken down into their own subcomponents as the (top-down) development process moves on (imagine a simple specification of A.TOP to be, for example, the disjunction of all externally visible import variables, $1 \lor 2 \lor 3 \lor 4 \lor 6 \lor 7 \lor 8 \lor 9 \lor 10$).

![Figure 1. Exemplary (fragment of a) component fault tree](image)

Consistent refinement of the failure model requires that a realisation $R_L = \{ S_{1L}, S_{2L}, \ldots, S_{nL} \}$ of hierarchy level $L$ should imply its superordinated specification $S_{2L}$:

$$R_L \Rightarrow S_{2L} \Leftrightarrow \neg R_L \lor S_{2L}.$$  \hspace{1cm} (1)

Whenever implication (1) is not fulfilled, there is a potential hazard on level $L$ which is not captured on higher levels (if a subcomponent is able to produce an internal failure that independently contributes to a higher-level one, that failure must be part of its specification). In this way it is guaranteed that a component’s FT completely covers subordinated failure...
behaviour; thus, it represents a logical upper bound. In component-oriented development, (1) plays a vital role for safety analyses because it ensures that neither qualitative nor quantitative results can be exceeded later or on lower levels. Thus, decisions may rely on early analyses, and the safety/design space can actually be navigated.

B. Bridging the gap between qualitative & quantitative analyses

In the development of safety-critical and software-controlled systems, even if there is a software fault tree (FT) model already present, the question of assigning reasonable probabilities remains. Because of that, only some standards do mention quantitative analyses for software, but it would be useful to have them all the same. Otherwise, analyses for hardware (quantitative and qualitative) and software (purely qualitative) are hard to integrate to get a comprehensive picture, and to derive measures and priorities on which to base design decisions.

III. FTA WITH INTERVAL-VALUED VARIABLES

Uncertainty about possible variability is the overriding reason for the popularity with safety analysts of the ‘what-if’ game: What, if this component was ten times as likely to fail; what, if the failure rate of that signal were very low; what, if we cut off a fault tree branch altogether... So, what, if we play this game a million times? Starting from the fault tree of a certain top-level failure mode, its leaves (variables, but not necessarily ‘basic events’ in the classical sense) are assigned probability intervals (think of SIL or ASIL). Sampling every variable’s interval many times and evaluating the FT based on those values results in an approximation of a probability distribution. The distribution is displayed in a histogram, showing the probability mass function of the probability of the failure mode of interest occurring. (Semi)quantitative FTA in the proposed form supports continuous analysis and prioritisation, complementing the purely logical results of a qualitative analysis with measures that are useful for efficient refinement, continuous analysis, prioritisation and feedback.

Today, stochastic Boolean functions are usually evaluated by transforming them to a BDD [1] [8] [29] equivalent, which is canonical for any fixed variable order. The Shannon expansion [40] used to build BDDs ensures that BDD calculations can be optimally efficient.

A. From Boolean logic functions to Boolean polynomials

Annotating s-independent Boolean variables, \( X_i \in \{ \text{false, true} \} \), with probabilities, \( P_i = P(X_i = \text{true}) \), enables a straightforward mapping of binary logic to stochastic algebra.

\[
\begin{align*}
&f = \bigvee_{m \in M} C_m, \quad C_m = \bigwedge_{i \in I_m} X_i^j \\
P(f) = \sum_{m \in M} \prod_{i \in I_m} P_i^{j_i}, \text{ with} \\
I_m \subseteq \{1, \ldots, n\}, j \in \{0, 1\} \\
X_i^0 = \overline{X_i}, X_i^1 = X_i, P_i^0 = 1 - P_i, P_i^1 = P_i 
\end{align*}
\]

(2)

The transformation is consistent with stochastic algebra if and only if \( C_m \land C_n = \{ \} \), \( m \neq n \) and, thus, \( P(C_m \lor C_n) = P(C_m) + P(C_n) \), which is guaranteed by representing \( f \) in BDD form, with every path corresponding to one conjunction \( C_m \). The sum of products on the right-hand side is a multilinear Boolean polynomial [7] [38].

B. Combining interval algebra with Boolean polynomials

Extending standard numeric algebra to intervals is straightforward, too [33]:

\[
X = [x_l, x_u], \quad Y = [y_l, y_u], \quad \bullet \in \{+,-,\cdot,\div\} \\
X \bullet Y = [\min(x_l y_l, x_l y_u, x_u y_l, x_u y_u), \max(x_l y_l, x_l y_u, x_u y_l, x_u y_u)]
\]

provided that, if \( \bullet = \div, 0 \notin Y \). Uniting Boolean polynomials with stochastic interval algebra, however, poses a significant challenge because it requires symbolic interval operations. If we ‘naively’ calculate, for example, \( A \lor B = [a_l, a_u] \lor [b_l, b_u] \), according to its BDD representation and (2) and (3), the prerequisite that \( A \) can only take one value at a time is ignored, yielding bounds for the result that are too wide or even outside the \([0, 1]\) range for probabilities altogether [11]. If a polynomial can be transformed to a representation where every variable appears just once, numeric interval algebra may be applied directly; since we would like to use BDDs for calculations, though, this is not an option.

There exist approaches to cope with the repeated-variable problem. [11] applies constrained mathematics, [9] propose variants of Skelboe’s technique [42], while [36] represent uncertainty by distribution mean and standard deviation (see Section VI). Our requirements of (a) comprehensive and, within limits, exact characterisation of the actual top event distribution and (b) direct use of BDDs for efficiency reasons, disposed us not to follow one of the approaches mentioned but to rather use ‘plain old random sampling’ in the present research.

C. Stochastic analysis of FT with interval-valued variables

The probability distribution of top event probabilities (hence, second-order probabilities) is determined by randomly sampling one probability value from every variable’s probability interval, calculating the corresponding top event probability, and doing that a large number of times. The set of resulting probabilities was distributed among equally spaced histogram bins, and the relative number of samples per bin used to derive the probability of the bin’s associated probability interval. As with BDD/FT, probability intervals of variables were created at random, ranging between \(10^2\) and \(10^6\). For the rendering of histograms shown here we typically used between 500,000 and one million samples of the top event probability, combined with 500 bins.

For a more general handling of probability intervals, we work with a parameterised representation, such that an interval \( X = [x_l, x_u] \) is given by \( f_{x_i} = x_l + c_i(x_u - x_l), c_i \in [0, 1] \). The choice of \( c_i \) determines a concrete variable value \( x \in X \). As a starting point, the stochastic distribution of variable probabilities was chosen to be uniform over their intervals because uniformity is
consistent with a least-knowledge assumption (Laplace’s principle) [18]. Nevertheless, the method allows using any conceivable distribution if one should have reason to do so.

Measures that can be calculated immediately without the need to resort to random sampling are (a) the expectation of the probability of the analysed failure mode, if expectations of the distribution of variable probabilities are known, and (b) tight upper and lower bounds for the resulting top event distribution, if the fault tree is coherent so it represents a monotonic function. For the expectation $E$ of the Boolean polynomials we use, the following (simplified) formula applies:

$$\begin{align*}
    f = \sum_{i=1} X_i \Rightarrow E(f) &= \sum_{i=1} E(X_i), \\
    \text{given that all } X_i \text{ are s-independent, which is true for FT/BDD variables by definition. For } X_i \text{ with uniform distributions over the probability interval } [x_{\alpha_i}, x_{\beta_i}], \text{ their expectation is the interval centre, } x_d + 0.5(x_{\beta_i} - x_{\alpha_i}), \text{ such that setting all } \alpha_i \text{ to 0.5 and evaluating the fault tree with the corresponding variable values yields its expectation. If the fault tree is coherent, tight bounds for the top event probability are calculated likewise, setting the } \alpha_i \text{ to zero (lower bound) and one (upper bound).}
\end{align*}$$

### IV. BASIC ANALYSES

For our investigation of the characteristics of top event probabilities for FTs with interval-valued basic events/variables two ways of BDD generation were employed:

- building, for a given number $n$ of variables, a random BDD by use of a routine that yields all Boolean $n$-variable functions with equal probability, and
- building, for a given number $n$ of variables, a random FT (without meshes) with a specified and/or gate ratio and a specified number of not gates which was then transformed to a BDD.

As was to be expected, using one or the other way had a significant impact on BDD size. By choosing a random function one will virtually always end up with a nonmonotonic one since the number of all Boolean $n$-variable functions grows much faster with $n$ than the number of monotonic Boolean functions with $n$ variables (the Dedekind number). Nonmonotonic functions, in turn, are more prone to BDD blowup than their monotonic counterparts. Setting the number of not gates to zero in the second generating method would give us coherent FTs, and thus monotonic functions, whose BDDs were usually manageable in size until around 100 variables. For $n = 100$, BDD sizes typically vary between half a million and ten million nodes, with occasional explosions into unknown ranges (generation was cancelled whenever the BDD library began to reserve more than 50 million nodes; so far, we do not use variable reordering for BDD size reduction).

With random (nonmonotonic) functions, the upper limit was significantly reduced to about 20 variables. Inserting not gates in random places into an FT also increased BDD size, but in a manageable way until about half a dozen negations. We conjecture that this effect was even stronger would we allow meshes in the FT structure (repeated events) that, together with not gates, would give rise to xor-like structures very easily. For the time being, we restrict ourselves to monotonic functions/coherent FTs mostly. We believe that, in principle, having the option of using negations is desirable [2] [41], but the BDD largeness problem could prove insurmountable. Still, the techniques presented can be applied to any Boolean function.

### A. Second-order probability mass functions

The combination of random functions with random variable probability intervals and uniform random sampling enabled us to get a comprehensive view of the different kinds of resulting probability mass functions of top event probabilities. To our knowledge, this has been the first attempt to characterise such functions in a general fashion. Note that distributions do not appear ‘diluted’ and spread out over a large range without real gestalt, peaks or valleys, even with many variables and wide intervals. On the contrary, they possess distinct ‘personalities’ every time.

**Figure 2.** Some basic PMF shapes of top event distributions

PMF shapes are primarily determined by the Boolean function they represent and, to a lesser degree, by width and location of the probability intervals of variables. There are typical forms that appear again and again: five basic classes can be identified that we call ‘bell’, ‘shark’, ‘triangle’, ‘(left) halfpipe’ (Figure 2, clockwise from upper left; derived from functions with ten variables) and ‘box’ (not shown here). There are also many hybrids such as the ‘fat bell’ (bell plus box), ‘fat shark’ or ‘trapeze’ (fat triangle), but the basic shapes they vary are obvious in most cases (Figure 3 presents more variants).

In simulations the sample mean was always in close agreement with the sampling-independent expectation. As an additional basic measure, sample median and standard deviation were calculated (standard deviation appears to be more useful than variance for very small values because it is of a similar order of magnitude as the mean). Distribution shapes facilitate intuitive comprehension of model properties. In a first approximation it can be said that the more the main probability mass rests close to the upper bound, the more likely it is that a
hierarchical refinement (realisation) of the safety model that conforms to the analysed fault tree (specification) will exhibit a relatively high probability of the top event in question (Section II), if no further measures are taken. If the maximum appears at the upper bound (Figure 3, top) this suggests that safety goals will be compromised easily for the analysed mode because the upper bound is not stable and will be exceeded easily when the range/interval of a variable is extended or shifted towards greater values. Note that this conclusion may turn out differently for two models that would be assigned the same number (that is, upper bound or expectation) using conventional single-point estimates, and also for two models representing the same Boolean function.

According to component characteristics, variable probabilities may be interval valued and uniformly distributed, fixed (for example, [25] demands that the safety classification of software components should be based on the assumption that a failure which is possible will indeed occur) or, for hardware parts, time dependent. All these distributions can be expressed by a PMF and sampled accordingly for an integrated analysis.

B. Characteristic probability mass functions

In case our uncertainty about the probability range of a variable is maximal, it makes sense to choose the unity interval [0, 1]. If this is done for all variables of a function, sampling gives us a histogram we call characteristic PMF. It solely depicts the stochastic behaviour of the failure function itself under the assumptions of variable independence and distribution uniformity, without recurrence to specific probability ranges of a variable, and its range is, of course, also the unity interval.

Figure 3 shows some examples derived from functions with 15 variables (from top): ‘right halfpipe’, ‘right jump’ (jump is a shark variant with a gradual transition between positive and negative curvature), ‘dome’ and ‘left jump’. The general appearance of the characteristic PMF remains similar when other probability intervals are assigned to variables, as long as they are of comparable range and magnitude. The transition from right halfpipe to left jump in the four characteristic PMF of Figure 3 demonstrates the role of the failure function in determining failure probability distribution.

The more conjunctions (representing, for example, the application of mitigation measures) there are and the higher their position in the tree, the lower is the mean of resulting failure probabilities: 0.72, 0.6, 0.49, 0.3 (from top to bottom). This is confirmed by a MCS cardinality analysis of the functions (Table I).

| Table I. MCS of failure functions (Figure 3) |
|-----------------|-----|-----|-----|-----|-----|-----|-----|-----|
|                | 2   | 3   | 4   | 5   | 6   | 7   | 8   |
| R halfpipe      | 10  | 2   | -   | -   | -   | -   | -   |
| R jump          | -   | 10  | 50  | -   | -   | -   | -   |
| Dome            | -   | -   | 24  | -   | 24  | -   | -   |
| L jump          | -   | -   | 12  | -   | 30  | -   | 12  |

While the right halfpipe (‘bad’) comes from a function with ten MCS of two variables and two MCS of three, the left jump (‘good’) function features twelve MCS of size four, 30 of size six and another twelve of size eight. The dome makes up its larger number of size-three MCS in comparison with the right jump by having none with four variables. This analysis gives an idea of the expressiveness of (characteristic) PMF.

C. Interval-based importance measures

Figure 4. Examples of criticality PMFs (two different variables out of ten)

Other measures can be extended to the interval domain as well, resulting in, for example, the PMF of importance [5] and criticality. Figure 4 shows the criticality of two different
variables of the same failure function. For these diagrams, criticality was calculated per sample and then, as with other PMF, transformed into a histogram. Similar to probabilities, importances can be characterised by sample mean, median and SD/variance and assessed in the same way as their “sharp” counterparts in classical FTA. The mean value of the left one is 0.32, with a SD of 0.167 while the criticality of the other variable has a mean of 0.037 and a SD of 0.028 (rounded).

Clearly, criticality distributions are more useful in a design process than simple figures, as the definite realisation is only established at its end. It is all the more so when the samples producing a distribution are divided into subsets (for example, those with greatest or least value) which can be examined more closely. Section V demonstrates a subset analysis applied to probability samples.

D. Boundary sampling

If the foremost interest is in the extreme values of the probability function, especially when it is nonmonotonic so its tight bounds are hard to determine, the likelihood of their being “hit” can be increased by restricting alpha parameters to \{0, 1\} (because Boolean polynomials are linear, their partial derivative for any variable is constant, such that extreme values of the entire function require that every variable should assume either its highest or lowest value). This technique, which we call boundary sampling, produces PMF that look different from those with unrestricted sampling (Figure 5). Function values have accumulation points, separated at times by intervals containing few samples, or none at all.

The spectrum-like structure of boundary PMF mirrors the influence of variables and variable combinations that dominate the location of a sample, with other combinations shifting it to a lesser and lesser degree.

E. Effort

While the effort of evaluating a FT not just once but a million, or at least many, times may seem daunting, we dare to say that it is, in fact, not very significant. In any case, calculation effort will be much less critical for project costs than developer manpower.

The required Monte-Carlo simulation is linear in BDD size and number of samples and trivially parallelisable, so by using, for example, cloud computing resources it can be speeded up by many orders of magnitude. For effort reduction, approximations with fewer samples should be evaluated first; additionally, divide-and-conquer strategies can be applied, similar to the standard method of evaluating independent FT branches and combining the results, by resampling the PMF of FT modules.

V. ADVANCED ANALYSIS

A PMF result may be subjected to further analysis since we record for every function value the combination of variable (alpha) values that produces it. Examination of sample subsets with a given probability threshold facilitate safety-oriented cost optimisation.

As an example, we analysed a CFT model (Figure 1; its PMF is the ‘shark’ of Figure 2). The probability intervals associated with variables are listed in column two of Table II. Based on these, the true probability range (rounded) of the top event is \([1.0001 \times 10^6, 2.0012 \times 10^5]\). With one million samples (their calculation taking about 18 seconds for the BDD with 30 nodes), the sample range covered 99.48% of the true range. Expectation is \(8.03 \times 10^6\), sample mean and median were \(8.02 \times 10^6\) and \(7.97 \times 10^6\), respectively, with a standard deviation of \(3.4 \times 10^6\).

<table>
<thead>
<tr>
<th>Var</th>
<th>Pr. interval</th>
<th>(\emptyset)</th>
<th>(\alpha)</th>
<th>Sample 1, (\alpha)</th>
<th>Sample 2, (\alpha)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>([10^3, 10^4])</td>
<td>0.1928</td>
<td>0.0065</td>
<td>0.0012</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>([10^4, 10^5])</td>
<td>0.4946</td>
<td>0.7283</td>
<td>0.5599</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>([10^4, 10^5])</td>
<td>0.5028</td>
<td>0.2057</td>
<td>0.9086</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>([10^5, 10^6])</td>
<td>0.5083</td>
<td>0.335</td>
<td>0.4909</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>([10^5, 10^6])</td>
<td>0.229</td>
<td>0.0198</td>
<td>0.7988</td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>([10^6, 10^7])</td>
<td>0.4989</td>
<td>0.2062</td>
<td>0.3855</td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>([10^6, 10^7])</td>
<td>0.0192</td>
<td>0.0552</td>
<td>0.0488</td>
<td></td>
</tr>
<tr>
<td>8</td>
<td>([10^6, 10^7])</td>
<td>0.4962</td>
<td>0.6478</td>
<td>0.2113</td>
<td></td>
</tr>
<tr>
<td>9</td>
<td>([10^6, 10^7])</td>
<td>0.4977</td>
<td>0.2499</td>
<td>0.9643</td>
<td></td>
</tr>
<tr>
<td>10</td>
<td>([10^7, 10^8])</td>
<td>0.5021</td>
<td>0.8567</td>
<td>0.887</td>
<td></td>
</tr>
</tbody>
</table>

Within the range of the function \(f\), \(1.5 \times 10^6\) was fixed as the upper limit for top event probability (safety goal) that should not be exceeded. Of the whole set of one million, 5585 samples fulfilled that goal in one particular simulation run. Average alpha values of the subset, for each variable, are listed in column three of Table II. It is obvious that the most influential variables are 7, 1 and 5 (bold type) as they need to deviate significantly from the average of 0.5 for the function value to fall below the threshold.

A minimal cut set analysis confirms this: MCS\((f) = \{7, \{1, 5\}, \{2, 9\}, \{6, 9\}, \{3, 5, 8\}, \{4, 9, 10\}\}\), where the MCS with by far the greatest upper probability bounds are \{7\} and \{1, 5\}. The only variable with a definite maximum alpha (of 0.0552) is 7, as
it represents a single point of failure, all other variables in the sample subset could, in principle, take almost any alpha value between zero and one, though not necessarily at the same time because of the required correlation between members of the same MCS (as reflected by alpha averages). Column four of Table II shows the alpha values of the sample (no. 1) that got closest to the threshold without exceeding it.

We defined a fictitious cost function \( f_c \) whose only requirements were that it reflected increasing cost with decreasing probability associated with a variable of a sample as well as different cost factors for different variables to be realistic. We used

\[
f_c(\text{sample}_i) = \sum_{j \in \text{PMF}} \sqrt{\text{value}(v_j(i))}
\]  

(5)

To calculate the cost of a sample, for every variable, its number (1 to 10 in this example) is divided by the square root of the sample’s variable value within its probability interval (determined by its alpha values), and the results are summed up. The PMF according to this cost function is shown in Figure 6. Costs within the sample subset in question range from 10542 to 21552, with mean 13347, median 12967 and SD of 1569 (rounded). The rightmost column of Table II contains the alpha values of the sample with least cost from the subset.

The probability of variable 5 can even be more than one order of magnitude higher than with sample 1, while the top event probability still meets the safety goal of \( 1.5 \times 10^{-6} \).

For this example we employed the specifications of subcomponents B, C and D to analyse a possible realisation of component A. In a distributed development project, optimal failure probabilities of subcomponents and required services can be derived from top event thresholds and communicated to subcontractors by the manufacturer. Instead of (and in addition to) determining constraints on the probability of incoming failures and delegating responsibility for failure detection and mitigation to the environment of A, improvements can be made by refining internal specifications. Alternatives can then be compared and optimised in the described way to serve as vantage points for the next iteration.

We have, of course, not started with approximation by intervals to arrive at a dozen significant digits. Therefore, numbers should be taken with a grain of salt and be used as expectations of, again, intervals, rather than as fixed suprema. In any case, the rationale behind the process is not to come up with a definite conception in one step but consists in hierarchically constraining failure models and associated ranges of likelihood, taking into account trade-offs and alternatives, until they zero in on an effective and cost-efficient solution.

VI. RELATED WORK

Uncertainty and approximate reasoning is seen as an important challenge in all areas of risk analysis, and a multitude of approaches exist to tackle it. Prominent among them are possibilistic methods using concepts from fuzzy logic and belief theory [15] [21] [39] [43] [45], probabilistic ones employing interval analysis and distribution properties [4] [6] [10] [11] [28] [31] [35] [36], and techniques based on order-of-magnitude reasoning [23] [34] [37]. Since we take a probabilistic approach here, we briefly review select publications from that domain.

Page and Perry [36] propose mean and standard deviation of random variables ‘as an alternative to fuzziness’ to express uncertainty in FT models. Associating basic events with a mean value and its standard deviation and propagating these through the tree, mean value and bounds of the SD of the top event’s probability can be obtained. By use of the Chebyshev inequality, the bounds for top-event SD are improved further. Cooper and Ferson [11] [10] use constrained calculation to derive tight bounds on Boolean functions with intervals. Calculations preserve variable identity by representing them in interval-centre-epsilon form and solving corresponding equation systems. Lodwick [31] follows a similar path, employing Lagrangian formulae and their extension to inequalities by the Kuhn-Tucker theorems. Carreras and Walker [9] take variants of Skelboe’s technique [42], using linear and geometric grids to subdivide intervals such that the excess produced by numeric interval calculations is reduced, and determine top event distributions in histogram form.

Of the mentioned approaches, ours is closest to [9] in that actual distributions are generated by random sampling of probability intervals. In contrast to [9], we use point sampling...
and standard FT evaluation instead of interval arithmetic. The main reason is that, although grids serving as interval subdivisions reduce the excess caused by interval arithmetic, they produce, at the same time, stochastic artifacts. These are visible as dents and wobbles in histograms where there should be none. We conjecture that artifacts are caused mainly by the combination of two error sources: (a) the assumption that output intervals are uniformly distributed and (b) the accumulation points of Boolean functions, as they become obvious with boundary sampling (Section IV.D).

For monotone functions, we also use interval besides point sampling, but calculate lower and upper bound separately such that result bounds are exact. The two error sources described above remain, however, and we are currently conducting further tests to give a more detailed account of the effects. In the context of subset analysis, interval sampling has the advantage of yielding interval-valued results directly.

VII. CONCLUSION & FUTURE WORK

In this article we present extensions of FTA for interval-valued variables to support hardware- and software-oriented safety analysis at an early stage. We use random (Monte-Carlo) sampling to generate second-order PMF, while subsequent analysis of sample subsets leads to a new optimisation method. The techniques demonstrated here allow to (a) bridge the existing gap between qualitative and quantitative safety analyses, integrating hardware and software analyses in a (semi)quantitative way, (b) use probability and importance measures under conditions of uncertainty, (c) capture variability inherent in a design process to leverage constructive, as opposed to after-the-fact, application of FTA and (d) evaluate safety-cost tradeoffs for development guidance.

PMF histograms afford our approach a strong visual aspect by giving an intuitive display of safety properties that has not been on hand in embedded systems engineering. These diagrams have the potential to be used by safety analysts in much the same way as spectrum diagrams by an electrical engineer. Implementation of the described analyses and visualisations in our industry-proven toolchain (a part of it is available at [24]) for safety-oriented, component- and model-based embedded systems design is complete. Further investigation of the relation between PMF shape and failure functions is on its way, as well as of the use of PMF and intervals as abstractions of failure functions.

Putting brute force behind the ‘what-if’ game opens up new possibilities for failure analysis, facilitating the design and safety engineering of highly complex systems by powerful visualisations and optimisations. Thereby, a whole landscape of answers becomes apparent, and consequently, of questions that may be asked. It features the seamless integration of fault tree analysis for software and hardware parts of a system, from day one of the design phase. The resulting safety-oriented design space navigation enables safety managers to choose a good and effective, if not optimal, solution every time. We hope and expect that the concepts proposed here will lead to a more effective and efficient safety process for the development of safety-critical and software-controlled systems.

REFERENCES


