DFTCalc: A Tool for Efficient Fault Tree Analysis (extended version)*

Florian Arnold[1], Axel Belinfante[1], Freark Van der Berg[1], Dennis Guck[1], Mariëlle Stoelinga[1]

[1] Department of Computer Science, University of Twente, The Netherlands
{f.arnold,d.guck,a.f.e.belinfante,m.i.a.stoelinga}@utwente.nl
f.i.vanderberg@student.utwente.nl

Abstract. Effective risk management is a key to ensure that vital assets like our nuclear power plants, medical equipment, and power grids are dependable. Also, risk management is often required by law. Fault Tree Analysis (FTA) is a widely used methodology here, computing important dependability measures like system reliability and availability. This paper presents DFTCalc, a powerful tool for FTA that provides (1) efficient fault tree modelling via compact representations; (2) effective analysis, allowing a wide range of dependability properties to be analysed; (3) efficient analysis, via state-of-the-art stochastic techniques; and (4) a flexible and extensible framework, where gates can easily be changed or added. Technically, DFTCalc is realised via stochastic model checking, an innovative technique offering a wide plethora of powerful analysis techniques, including aggressive compression techniques to keep the underlying state space small.

1 Introduction

Risk analysis is a key feature in reliability engineering: in order to design and build medical devices, smart grids, and internet shops that meet the required dependability standards, we need to assess how dependable these systems are, and take appropriate measures if they are not dependable enough. This analysis is most useful when carried out at design time. Then important reliability decisions are made concerning the system architecture, the level of redundancy and spare management.

Fault Trees. Fault tree analysis (FTA) is a graphical technique that is often used in industry [27]. Fault trees (FTs) model how component failures lead to system failures: The leaves of a FT are basic events (BEs) that represent component failures, e.g. the failure of a motor. The other nodes express how failures

* This research has been partially funded by the NWO under the project ArRangeer (12238), and by the DFG/NWO bilateral project ROCKS (DN 63-257) and by the European Commission's FP7 under the project TREsPASS (318003).

propagate through the system via AND and OR gates. For instance, a car fails, if either the motor, the battery or the starting mechanism fails. Quantitative fault tree analysis derives the probability of a system failure from the probabilities of the components' failure. There are two important models: Discrete time FTs equip each BE with a probability p, representing the probability that the component fails within a certain discrete time interval. In this paper we consider continuous FTs. Here, each BE is equipped with a probability distribution f showing how the failure behaviour evolves over time, i.e. $F(t)$ represents the probability that the BE is still running at time point t. The root of the tree, called the top-level event, represents a system failure. FTA typically computes for a given FT the *system reliability,* i.e. the probability that the system has not failed within a given mission time T, the *mean time to failure* (MTTF), i.e. the expected time of a failure to occur, and the *availability*, i.e. the time that the system is up in the long run.

Dynamic Fault Trees (DFTs) extend standard (or static) fault trees with a number of intuitive gates. These gates facilitate the modelling of often recurring concepts in reliability engineering: spare management, functional dependencies, and order-dependent behaviour. We focus on DFTs without maintenance.

DFTCalc. DFTCalc is a powerful tool for modelling and analysis of DFTs. It can efficiently model DFTs and provides means to compute various dependability metrics, given BEs whose failure probabilities are given by exponential and phase type distributions. The major innovation of DFTCalc is the deployment of stochastic model checking (SMC) techniques [3]: SMC is an innovative technique to systematically explore the state space of a stochastic system and provides a wide plethora of powerful analysis techniques, with fully-fledged tool support. By deploying SMC, DFTCalc can handle DFTs with BEs that are statistically dependent; in fact, the FDEP gate has specifically been designed to model interdependent events. Repairs, however, have not yet been included.

The main problem in time-dependent reliability analysis is its complexity: The state space of models of real systems can grow very large [21] and, thus, highly efficient techniques are needed to yield results in a reasonable amount of time. Furthermore, an accurate modelling of all dependencies in these inherently complex systems requires an ever growing diversity of new gates. DFTCalc constitutes an architectural framework that addresses both challenges, and thereby yields four major advantages:

- *Increased modelling power.* Compared to earlier DFT tools, DFTCalc's input language is more powerful and imposes fewer syntactic restrictions: DFTCalc allows any DFT to be a spare component or a trigger, and is no longer restricted to BEs as in [22]. This is a great advantage in practice, since spare components and triggers are often complete subsystems.
- *Increased analytical power.* SMC enables DFTCalc to analyse a wide range of dependability metrics, namely those expressed in a large subset of the logic CSL [1]. Also, as argued in [6], certain DFTs give rise to non-determinism. These can be handled by DFTCalc.

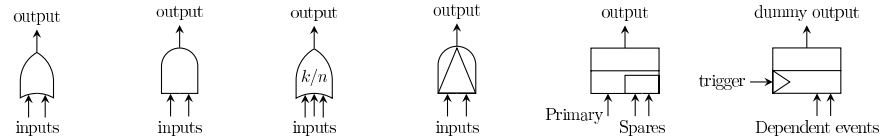


Fig. 1. Dynamic fault tree gates.

- *Efficiency.* DFTCalc uses compositional aggregation techniques (Section 3) that lead to significant speed ups of several orders of magnitude.
- *Flexibility.* The compositional aggregation approach makes the framework very extendable. In order to change the behaviour of a gate or add new gate types, we only need to provide the underlying, relatively simple model in the form of an input/output interactive Markov chain (Section 2.2).

Related work. A wide range of FTA methods exists: Classically, one obtains the minimal cut sets in the FT [4]. That is, those minimal sets of BEs whose failure results in a system failure. This enables to order components based on their structural importance. Further, with additional probabilistic information one can compute the system reliability. A popular technique is to exploit Bayesian networks, which used both in discrete time [9] and in continuous time [8]. Our approach focuses on continuous timed systems. Therefore, we will translate DFTs into continuous time Markov chains (CTMCs) and use state of the art techniques as described in [1,2]. This allows us to compute reliability measures by use of efficient techniques for transient analysis of CTMCs.

A wide number of commercial and academic tools for static fault tree analysis are available. Some are merely drawing tools, while others provide probabilistic analysis, like the popular FaultTree+ package from Isograph [19]. Dynamic FTA is supported by tools like Windchill [24], NASA's Galileo/ASSAP software [12], and the simulation tool DFTSim [10]. A first implementation of DFT analysis using I/O-IMCs was realized in Coral [7], the predecessor of DFTCalc.

Organisation of the paper. Section 2 presents DFTCalc's modelling and analysis capabilities and Section 3 the architecture and internal structure. In Section 4 we provide experimental results and Section 5 concludes the paper.

2 Dynamic fault trees: modelling and analysis

2.1 Dynamic Fault Trees

DFT modelling. Dynamic fault trees (DFTs) model the failure propagation in complex systems. The leaves of a DFT are labeled with *basic events* and the non-leaves with *gates*. The root is called *top-level event*.

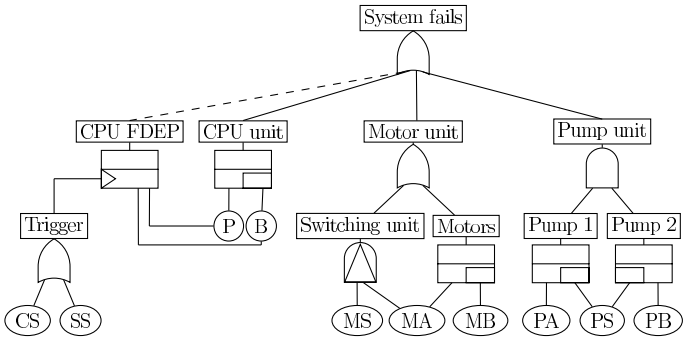


Fig. 2. The cardiac assist system DFT.

Basic events. A basic event (BE) represents the failure behaviour of a basic system component, and can be in three different modes: *dormant*, *active* and *failed*. The dormant mode indicates that a component is not in use, but acts as a stand-by or spare; the active mode indicates that the component is in use and works normally; finally the failed mode indicates that the component has failed. In active mode, a component fails with a certain rate λ which is the parameter of an exponetial distribution. In dormant mode the failure rate is decreased by a *dormancy factor* $\alpha \in [0, 1]$. In case $\alpha = 0$ the BE cannot fail (cold BE) and in case $\alpha = 1$ the failure rate is the same as in active mode (warm BE). Alternatively, the failure behaviour can be specified by a phase type distribution, see Sec. 2.3. In that case, the dormancy factor reduces each rate of the phase type distribution.

Gates. A gate expresses how component failures induce a system failure. Gates consist of one or more inputs, and one output. Figure 1 depicts the DFT gates.

(a) The OR gate fails when at least one input fails.
(b) The AND gate fails when all of its inputs fail.
(c) The VOTING gate fails when at least k out of n inputs fail.
(d) The PAND gate fails when all of its inputs fail from left to right.
(e) The SPARE gate consists of a *primary* input and one or more *spare* inputs. At system start, the primary is active and the spares are in dormant mode. When the primary input fails, one of the spare inputs is activated and replaces the primary. If no more spares are available, the SPARE gate fails. Note that a spare component can be shared among several spare gates.
(f) The FDEP (functional dependency) gate consists of one *trigger* event and several *dependent events*. When the trigger event occurs, all dependent events fail. The FDEP has a "dummy" output, which is represented by a dotted line and ignored in calculations.

Example 1. Fig. 2 depicts a DFT representing a cardiac assist system (CAS) [9] consisting of three subsystems: the CPU, the motor and pump units. If either

one of these subsystems fails, then the entire CAS fails, as modelled by the top level OR gate. The CPU unit consists of a primary (P) and a backup (B) CPU, as indicated by the SPARE gate. The primary and backup CPU are subject to a common cause failure, modelled by the CPU FDEP gate: if either the crossbar switch (CS) or the system supervisor (SS) fails, the primary and backup CPU become unavailable. The motor unit consists of a primary (MA) and a backup (MB) motor. If the primary fails, the motor switching component (MS) will turn on the backup motor. Because of the PAND gate the failure of the switching component can then be ignored. Finally, the pump unit consists of two pumps (PA and PB), which share a common cold spare (PS).

2.2 Input/output interactive Markov chains

DFTCalc uses input/output interactive Markov chains (I/O-IMCs) [6] to formally encode the semantics of DFT gates and leaves. I/O-IMCs extend interactive Markov chains (IMCs) [17] by integrating features from input/output automata. An I/O-IMC consists of a number of states which are connected via transitions. As in interactive Markov chains, transitions are classified as either *Markovian transitions* or *interactive transitions*. Markovian transitions represent a system delay. They are labeled with rates λ indicating that the transition can be taken after an exponentially distributed delay with parameter λ. In other words, the time until a Markovian transition is taken, represented by random variable X, is defined by the cumulative distribution function (CDF)

$$\mathbb{P}(X \leq t) = 1 - e^{-\lambda t}, \quad \text{for any } t \in \mathbb{R}^+.$$

On the other hand, interactive transitions are executed instantly. They are labeled with different kinds of actions:

(a) *Input actions* (denoted $a?$) can only be taken, if another I/O-IMC executes an output action $a!$; we say that $a?$ requires *synchronization* on $a!$. The action is thus possibly subject to delays.
(b) *Output actions* (denoted $a!$) cannot be delayed and have to be taken immediately. They emit the output signal $a!$ on which corresponding input actions can synchronize.
(c) *Internal actions* (denoted $a;$) cannot be delayed, quite like output actions. However, they are not visible to other I/O-IMCs, i.e. they do not require synchronization.

I/O IMCs are *input-enabled*, meaning that all states in an I/O-IMC can respond to all input signals from any other I/O-IMC in the considered system. These concepts are formalized in the following definition.

Definition 1 (Input/Output Interactive Markov Chain). *An input/output interactive Markov chain is a tuple* $\mathcal{I} = (S, s_0, Act, \rightarrow, \dashrightarrow)$ *where* S *is a set of states,* $s_0 \in S$ *is the initial state,* Act *is a finite set of actions (or signals), where* $Act = \{Act^I, Act^O, Act^{int}\}$ *with* Act^I *the set of input actions,* Act^O *the set of output actions and* Act^{int} *the set of internal actions, and*

- $\longrightarrow \subseteq S \times Act \times S$ *is a set of interactive transitions, and*
- $\dashrightarrow \subseteq S \times \mathbb{R}_{>0} \times S$ *is a set of Markovian transitions.*

I/O-IMC s are input enabled, such that $\forall s \in S. \alpha \in Act^I, \exists s' \in S. (s, \alpha, s') \in \longrightarrow$.

We abbreviate $(s, \alpha, s') \in \longrightarrow$ by $s \xrightarrow{\alpha} s'$ and $(s, \lambda, s') \in \dashrightarrow$ by $s \overset{\lambda}{\dashrightarrow} s'$. Further, we write $\alpha?$ iff $\alpha \in Act^I$, $\alpha!$ iff $\alpha \in Act^O$ and $\alpha;$ iff $\alpha \in Act^{int}$.

One of the key properties of I/O-IMCs is that they are compositional. Complex models consisting of various interacting I/O-IMCs can be aggregated in a stepwise, hierarchical manner to obtain one I/O-IMC representation of the whole system. We denote with $I_1 || I_2$ the parallel composition of I/O IMCs I_1 and I_2. Then $I_c = I_1 || I_2$ is again an I/O-IMC (with the Cartesian product of I_1 and I_2 as state space) expressing the joint behavior of its constituents:

(a) If an action does not require synchronisation, then I_1 and I_2 evolve independently.
(b) If an action $a?$ on an interactive transition requires synchronisation, then it can only be taken at the time when another I/O-IMC performs output action $a!$.

This behaviour is defined formally in the following definition.

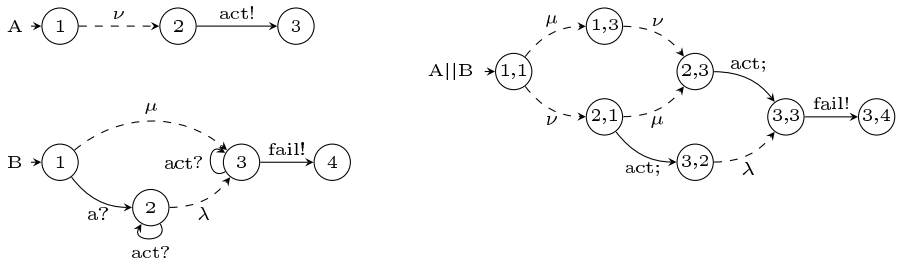
Definition 2 (Parallel Composition). *Let* $\mathcal{I}_1 = (S_1, s_{0,1}, Act_1, \longrightarrow_1, \dashrightarrow_1)$ *and* $\mathcal{I}_2 = (S_2, s_{0,2}, Act_2, \longrightarrow_2, \dashrightarrow_2)$ *be I/O-IMCs.* $\mathcal{I}_1$ *and* $\mathcal{I}_2$ *are composable, if* $Act_1^O \cap Act_2^O = Act_1^{int} \cap Act_2 = Act_1 \cap Act_2^{int} = \emptyset$. *The parallel composition of* $\mathcal{I}_1$ *and* $\mathcal{I}_2$ *is then defined by:*

$$\mathcal{I}_1 || \mathcal{I}_2 = (S_1 \times S_2, (s_{0,1}, s_{0,2}), \{(Act_1^I \cup Act_2^I) \setminus (Act_1^O \cup Act_2^O), Act_1^O \cup Act_2^O, Act_1^{int} \cup Act_2^{int}\}, \longrightarrow, \dashrightarrow)$$

where $\longrightarrow$ *and* $\dashrightarrow$ *are defined as the smallest relation satisfying*

(a) $s_1 \xrightarrow{\alpha} s_1'$ *and* $s_2 \xrightarrow{\alpha} s_2'$ *and* $\alpha \in Act_1 \cap Act_2$ *implies* $(s_1, s_2) \xrightarrow{\alpha} (s_1', s_2')$
(b) $s_1 \xrightarrow{\alpha} s_1'$ *and* $\alpha \notin Act_2$ *implies* $(s_1, s_2) \xrightarrow{\alpha} (s_1', s_2)$ *for any* $s_2 \in S_2$
(c) $s_2 \xrightarrow{\alpha} s_2'$ *and* $\alpha \notin Act_1$ *implies* $(s_1, s_2) \xrightarrow{\alpha} (s_1, s_2')$ *for any* $s_1 \in S_1$
(d) $s_1 \overset{\lambda}{\dashrightarrow} s_1'$ *implies* $(s_1, s_2) \overset{\lambda}{\dashrightarrow} (s_1', s_2)$ *for any* $s_2 \in S_2$
(e) $s_2 \overset{\lambda}{\dashrightarrow} s_2'$ *implies* $(s_1, s_2) \overset{\lambda}{\dashrightarrow} (s_1, s_2')$ *for any* $s_1 \in S_1$.

The first three constraints define the behaviour for interactive transitions: (a) describes the synchronisation between an input action $\alpha?$ and an output action $\alpha!$ where (b) and (c) describe the independent evolving of either I/O-IMC. The last two constraints (d) and (e) show that the I/O-IMCs can delay independanly. Hence, no synchronisation over Markovian transitions is done. This behaviour is justified by the memoryless property of exponential distributions: if two Markovian transitions with rate λ and μ competing for execution, then the remaining delay of the μ-transition after taking the λ-transition is exponentially distributet with rate μ.
After the parallel composition of two I/O-IMCs we can hide those actions which are no longer subject to further synchronisation. This is formally defined in the following.



(a) Two examples of I/O IMCs

(b) Parallel composition of A and B, hiding signal act

Fig. 3. Composition and hiding of I/O IMCs

Definition 3 (Hiding). *Let* $\mathcal{I} = (S, s_0, Act, \rightarrow, \dashrightarrow)$ *be a I/O-IMC and* $A \subseteq Act^O$ *a set of output actions. We define hide* A *in* $\mathcal{I}$ *as the I/O-IMC given by* $\mathcal{I} \setminus A = (S, \{Act^I, Act^O \setminus A, Act^{int} \cup A\}, \rightarrow, \dashrightarrow, s_0)$.

Example 2. Consider the two I/O-IMCs A and B in Fig. 3(a), where A describes an activation after a delay given by rate ν and B describes a dormant BE with dormant failure rate μ and active failure rate λ. States are depicted by circles, initial states by an incoming arrow, Markovian transitions by dashed lines, and interacting transitions by solid lines. While A has only one possible path from the initial to the final state, B can either directly reach state 3 by a Markovian transition, or move via state 2. The path taken is determined by a race condition: If the delay generated by the Markovian transition is shorter than the delay caused by the synchronization on action a, then B executes the transition from state 1 to state 3. When composing A and B we synchronize on action *act*. Since B has *act* as input, it has to wait for A's output action *act*!. All Markovian transitions and non-synchronising signals are essentially interleaved during composition. All synchronising signals are transformed into internal actions and thereby hidden.

2.3 Phase-type distribution

In the most generic case the failure times of components are represented via exponential distributions which capture the non-aging of mechanical elements with its memoryless property. Besides, we also have to resort to a different family of distributions to model the failure time accurately. In DFTCalc we use phase-type distributions to solve this problem. A *phase-type* (PH) distribution is a probability distribution constructed of several exponential distributions. An important property of PH distributions is that they can approximate any continuous distribution with arbitrary precision [20]. Therefore, DFTCalc allows the user to annotate the basic components with any suitable distribution. Formally, a phase-type distribution represents the distribution of the time until absorption in an absorbing CTMC [25]. An example is given in Fig. 4. We will focus on

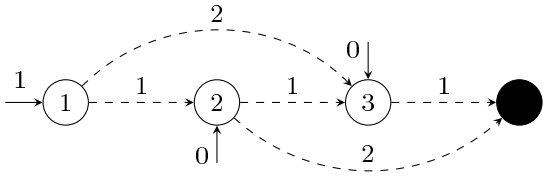


Fig. 4. CTMC representation of an APH distribution. The absorbing state is coloured in black.

acyclic phase-type (APH) distributions, that is, the representing Markov chains are acyclic.

Definition 4 (Continuous time Markov chain). *A continuous time Markov chain (CTMC) is a tuple* $\mathcal{C} = (S, \mathbf{Q}, \overrightarrow{\pi})$*, where* S *is a finite set of states,* $\mathbf{Q} : (S \times S) \to \mathbb{R}$ *is an infinitesimal* generator matrix*, and* $\overrightarrow{\pi} : S \to [0,1]$ *is the initial probability distribution on* S*.*

For any two states $s, s' \in S$, $\mathbf{Q}(s, s')$ represents the rate of the transition from s to s'. By definition $\mathbf{Q}(s, s') \geq 0$ for all $s, s' \in S$ with $s \neq s'$, and $\mathbf{Q}(s, s) = -\sum_{s \neq s'} \mathbf{Q}(s, s')$ otherwise. The negative diagonal value, $E(s) = -\mathbf{Q}(s, s)$ is called the *exit rate* of state s.

Definition 5 (Acyclic phase-type distribution). *Let* $\mathcal{C} = (S, \mathbf{Q}, \overrightarrow{\pi})$ *be a CTMC with* $S = \{s_0, s_1, \ldots, s_n\}$*. If the last state* $s_n \in S$ *is absorbing, i.e.* $E(s) = 0$*, and all other states* $s_i \in S$*, for* $0 \leq i < n$*, are transient, i.e. there is a nonzero probability that* s_i *will never be visited once it is left, then* $\mathcal{C}$ *describes an acyclic phase-type (APH) distribution.*

In other words, an APH distribution describes the time of absorption of a CTMC $\mathcal{C}$ without cycles and exactly one absorbing state. APH distributions are often represented in the following way: If $\mathcal{C}$ represents an APH distribution, its infinitesimal generator matrix can be written as

$$\mathbf{Q} = \begin{bmatrix} \mathbf{A} & \overrightarrow{A} \\ \overrightarrow{0} & 0 \end{bmatrix}.$$

Since we consider CTMCs with an acyclic graph structure $\mathbf{A}$ is an upper triangular matrix. Further, $\mathbf{A}$ is called *APH-generator* and is non-singular. We will map the initial distribution $\overrightarrow{\pi}$ to a new initial distribution $\overrightarrow{\alpha}$ for all transient states, such that $\alpha(s) = \pi(s)$ for all $s \in S \setminus \{s_n\}$. The pair $(\overrightarrow{\alpha}, \mathbf{A})$ is then the representation of an APH distribution.

Example 3. Consider the CTMC $\mathcal{C}$ depicted in Fig. 4. By definition, $\mathcal{C}$ represents a APH distribution with

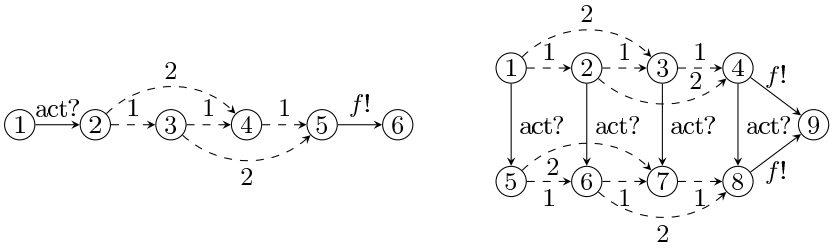


Fig. 5. (a) I/O-IMC representation of an active BE with APH distribution; (b) I/O-IMC representation of an dormant BE with APH distribution.

$$\mathbf{Q} = \left(\begin{array}{ccc|c} -3 & 1 & 2 & 0 \\ 0 & -3 & 1 & 2 \\ 0 & 0 & -1 & 1 \\ \hline 0 & 0 & 0 & 0 \end{array} \right) = \left(\begin{array}{c|c} \multicolumn{2}{c|}{\mathbf{A}} & \overrightarrow{A} \\ \hline \multicolumn{2}{c|}{\overrightarrow{0}} & 0 \end{array} \right)$$

$\overrightarrow{\pi} = [1,0,0,0] = [\overrightarrow{\alpha}, 0]$. Hence the corresponding APH distribution is given by $APH = (\overrightarrow{\alpha}, \mathbf{A})$.

COX normalform. The *COX* normalform [13] is a special representation of an APH distribution. In a COX representation of an APH, every state, apart from the absorbing state, has a transition to the next state, possibly a transition to the absorbing state, and no other transitions. Furthermore, it has descending exit rates and a Dirac initial distribution, that is, the state with the highest exit rate is the only possible starting state. Any APH can be transformed into a COX representation, by first applying the so-called spectral polynomial algorithm to yield a canonical ordered bidiagonal representation in time $\mathcal{O}(n^3)$, where n is the size of the given APH representation. Once an ordered bidiagonal representation is obtained, we can transform it into a COX representation in $\mathcal{O}(n)$ [25].

Phase type distribution in BEs. We now define how to model the failure behaviour of basic components with APH distributions. That is, instead of annotating the BEs with failure rates as for the BE B in Fig. 3(a), we annotate them with the CTMC representation of the APH distribution which determines the component's failure probability over time. To embed this feature into the I/O-IMC formalism, it is necessary that there is a unique initial state, i.e. the initial probability distribution is a Dirac distribution, such that we can plug in the APH distribution instead of the exponential distribution. An example for the I/O-IMC of a BE with APH distribution is presented in Fig. 5.

We can also model the failure behaviour of dormant BEs with APHs. Dormant BEs have a different failure distribution, depending on whether they are

still in dormant mode or have been activated. We model these two phases by linking two APHs which describe the distribution of a failure over time. The dormant APH distribution is obtained by applying a dormancy factor α to all rates in the CTMC that represents the APH distribution in dormant mode. The corresponding I/O-IMC starts with this CTMC, and once the component is activated, the I/O-IMC moves to the equivalent state in the APH that represents the active mode. The assumption behind this model approach is that the aging of the component in dormant mode is memorised when the component is activated [7].

Representation of Phase type distributions in BEs. Due to the use of I/O-IMCs we can only embed those APHs that have a Dirac initial probability distribution. When deriving an APH representation of a certain distribution, for instance by fitting algorithms [18], they usually do not fulfill this requirement. However, there are algorithms available to transform any APH distribution into a COX representation [13], an APH representation with a Dirac initial distribution.

2.4 DFT analysis

DFTCalc can compute a number of different reliability metrics, namely all metrics that can be expressed as reachability properties in the logic CSL. This includes properties such as:

(1) *Timed-Reliability*: the probability that the system fails until a given time point T or in a given interval $[T, T']$;
(2) *Mean time to failure*: the expected time to a system failure;
(3) *Reliability*: the probability that the system fails in the long-run.

In case of non-determinism, we calculate the minimum and maximum values for the above metrics. Each of these properties can either be evaluated from the initial state (i.e. the system is fully functional), or by *setting evidence* (i.e certain components have failed already).

DFTCalc fruitfully exploits the technique of *compositional aggregation,* see Fig. 8. Whereas traditional FTA methods translate a DFT into a large and monolithic CTMC, we do this in a stepwise fashion: First, DFTCalc translates each element (i.e., gate or BE) into an input-output interactive Markov chain (I/O-IMC), implementing the methodology from [6,7]. Then, we obtain the underlying CTMC by composing all I/O-IMCs. We compose these I/O-IMCs one-by-one, and employ aggressive state space compression technique in each step, to keep the state space minimal.

3 DFTCalc's structure: architecture and interface

Architecture. DFTCalc combines dedicated code and the state-of-the-art model checkers CADP, MRMC and IMCA:

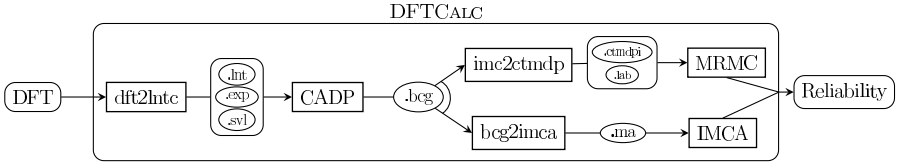


Fig. 6. The DFTCalc tool-chain.

CADP [15] supports construction, minimisation and analysis of IMCs. It compiles and generates the state space from a LotosNT [26] specification. The compositional verification engine of CADP can compose a network of communicating IMCs. The tool set also enables minimisation modulo strong and branching bisimulation.

MRMC [21] is a model checker for discrete-time and continuous-time Markov reward models. It supports the verification of properties expressed by the logics PCTL and CSL as well as their reward extensions CSRL and PCTRL. There is also a CTMDP extension available[1] which provides recent analysis techniques based on [11].

IMCA [16] is a tool for the quantitative analysis of IMCs. In particular, it supports the verification of IMCs against unbounded reachability, time- and interval-bounded reachability, expected-time objectives, and long-run average objectives.

The architecture of DFTCalc is displayed in Fig. 6 and the processing steps in Fig. 7: First, `dft2lntc` translates a DFT in Galileo format into `.lnt` format, a process calculus enriched with data that is input to CADP. Technically, this step transforms each DFT element into an I/O-IMC representing the element's behaviour. Additionally, a `.exp` file is generated that defines the interaction between components. The clear distinction between local component and global system information together with the compositional semantics of I/O-IMCs makes DFTCalc highly flexible: New components can be added or existing components adapted by specifying their behaviour in `.lnt` format and adding them to the tool's library. In the next step, the CADP tool set [15] uses the compositional aggregation method to generate the state space of the system, which is a I/O-IMC representation of the whole DFT. The output of CADP is a `.bcg` file. This format is translated either into a `.ctmdpi` file, which is input to the Markov Reward Model Checker MRMC [21], or into an `.ma` file, which is the input of the Interactive Markov Chain Analyzer IMCA [16]. Finally, the requested dependability metrics are computed.

Compositional aggregation. Compositional aggregation of I/O-IMCs lies at the heart of DFTCalc. As depicted in Fig. 8, after transforming each DFT element into an I/O-IMC, we iteratively compose the obtained I/O-IMCs. We

[1] `http://depend.cs.uni-sb.de/tools/ctmdp/`

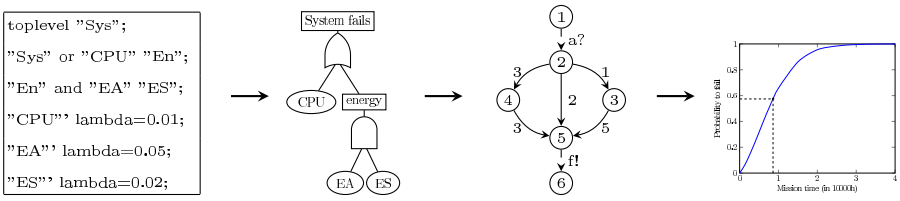


Fig. 7. Graphical overview of the processing steps in DFTCalc.

take two I/O-IMCs, compose them, hide all action labels that are no longer needed for synchronisation, and then minimise the composition via bisimulation minimisation. This process continues until a single I/O-IMC remains. The order of the aggregation process heavily influences the number of states in the obtained I/O-IMC, and is determined by a smart heuristic. Compositional aggregation yields reductions up to several orders of magnitude [7]. Given a DFT D, the involved steps to obtain one minimal I/O-IMC representation of D are:

(1) Each element in D is transformed into its corresponding I/O-IMC. The interaction between components in D is preserved by matching input and output signals: Once failed, basic events in D send output signals and attached gates receive the corresponding input signal. In this way, the signals are passed on bottom-up from the leafs of D to its root. This step results in a set of I/O-IMCs.
(2) Two interacting I/O-IMCs are composed in parallel with Def. 2. The order in which the I/O-IMCs are chosen and composed is determined by a smart heuristic which calculates for each possible aggregation a metric based on the number of outgoing transitions of the product state [14].
(3) All signals that are no longer necessary for composition, i.e. those signals which require no more synchronisation, are hidden and transformed into internal actions.
(4) The composed I/O-IMC is minimised using branching bisimulation [17].
(5) If more than one I/O-IMC is left, go to Step 2; otherwise continue with Step 6.
(6) The aggregated I/O-IMC is analysed with respect to the chosen metrics. Usually, we do not have non-determinism in the aggregated model and can obtain a CTMC, otherwise we obtain an IMC.

Web Interface. DFTCalc can be used by downloading a stand-alone version, and via a web interface. Both are accessible at `http://fmt.cs.utwente.nl/tools/dftcalc/`. DFTCalc is open source, but requires a license for CADP, which is free for academic institutions. The web interface is realised with the use of PUPTOL [5] and extends the downloadable version with a GUI as well as the plot function and is shown in Fig. 9. It allows the user to (1) input DFT models via a text screen, the topmost box in Fig. 9; (2) select the dependability metrics. This can be (a) the reliability for one or more mission times x, or (b) the probability on a system failure during an interval $[T1, T2]$, or (c) the mean

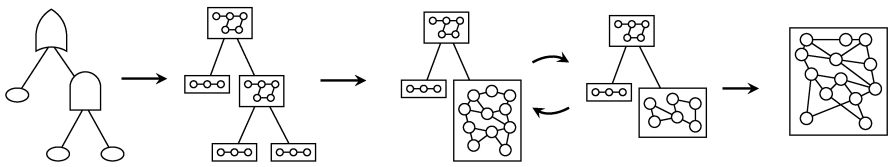


Fig. 8. Graphical overview of the compositional aggregation of DFT models.

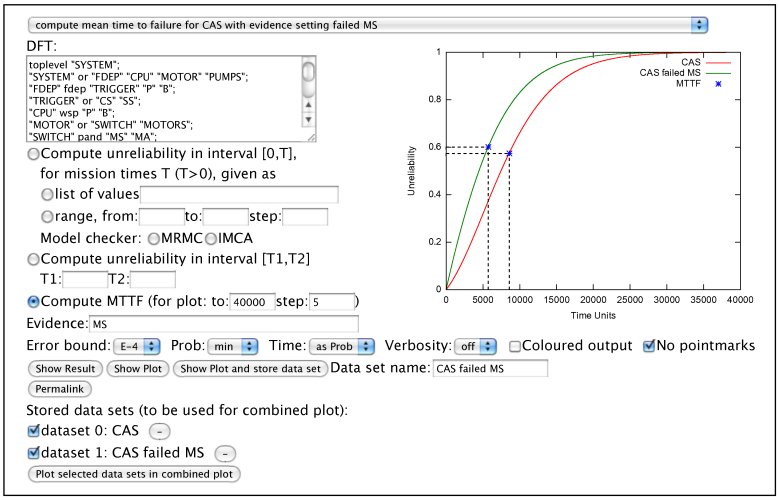


Fig. 9. DFTCalc web-tool interface.

time to failure; (3) set various options: which model checker to use; the error bound, the level of verbosity, and whether to color output. The results can be given either by numbers, via the button *show result*, or as a plot, via the button *plot result*. The input and configuration of the web interface can be saved via the button *permalink*.

4 Case Studies

We show the applicability of DFTCalc for three case studies: a multiprocessor computing system (MCS) [23,7]; the cardiac assist system (CAS) [9,8] from Figure 2; and a fault-tolerant parallel processor (FTPP) [7]. The MCS and CAS models were originally developed for discrete time models [23,9], but were analyzed — as we do — for continuous time models in [7,8]. The reliability data for the MCS and CAS case studies are given in Table 1.

All our experiments were conducted on a single core of a 2.7 GHz Intel Core2Duo processor with 2GB RAM running on Linux. Figure 11 presents the increasing failure probability over time for two systems as well as their expected

(a) Reliability of the MCS components.

Components	Failure rate (λ)	dormancy factor (α)
N	$2.0e^{-9}$/h	–
P1, P2	$5.0e^{-7}$/h	–
PS	$6.0e^{-6}$/h	–
D11, D21	$8.0e^{-5}$/h	–
D12, D22	$8.0e^{-8}$/h	0.5
M1, M2	$3.0e^{-8}$/h	–
M3	$3.0e^{-8}$/h	0.5

(b) Reliability of the CAS components.

Components	Failure rate (λ)	dormancy factor (α)
CS, SS	$2.0e^{-5}$/h	–
P	$5.0e^{-5}$/h	–
B	$5.0e^{-5}$/h	0.5
MS	$1.0e^{-6}$/h	–
MA, MB	$1.0e^{-4}$/h	–
PA, PB	$1.0e^{-4}$/h	–
PS	$1.0e^{-4}$/h	0.1

Table 1. Reliability data for MCS and CAS.

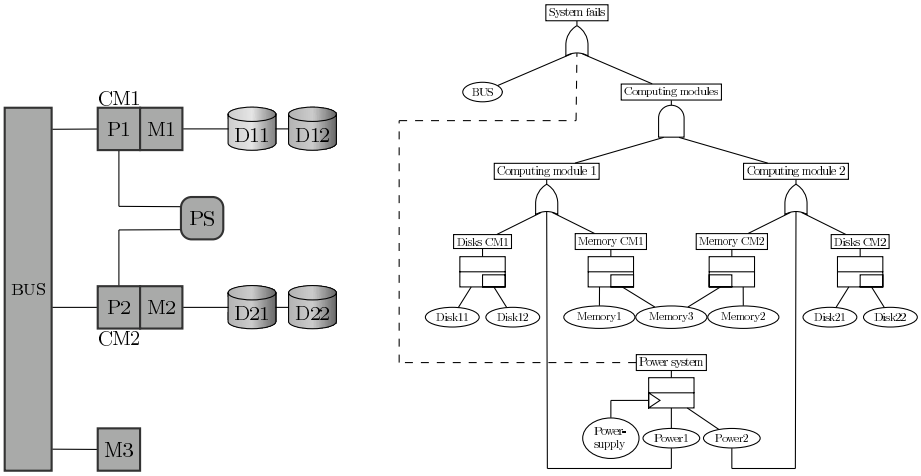
failure time. Table 2 compares *Coral* and DFTCalc: DFTCalc is up to three times faster than Coral and thereby also faster than earlier tools like Galileo [7].

Multiprocessor computing system (MCS). Figure 10(a) depicts the physical description of the MCS. The system consists of two computing modules CMi (i=1 or 2). They are connected via a bus, powered by a power supply (PS) and share a spare memory module M3. Each computing module consists of a processor Pi, a memory module Mi and two hard drives, a primary Di1 and a backup Di2. The system fails, if either both computing modules fail, the communication bus fails or the power supply fails. A computing module fails, if both hard drives fail, or the processor fails or the memory fails (where the shared spare memory is not available or also fails). The corresponding DFT is depicted in Figure 10(b). We also analyzed an extension of the MCS (4CM) which contains two more copies of the computing modules connected to the same bus with an own power source and a separate spare memory.

Fault-tolerant parallel processor (FTPP). The FTPP-n [7] models a redundant computer system consisting of four groups of n processors and is depicted for $n = 4$ in Fig. 12. The system consists of $4n$ processors divided into four logical groups. Each group is equippted with a shared cold spare. Furthermore, a network element physically connects one processor of each group to the system. If a network element fails, all connected processors become unavailable. In our set-up, all network elements have a failure rate equal to 0.017, and all processors have a failure rate equal to 0.11.

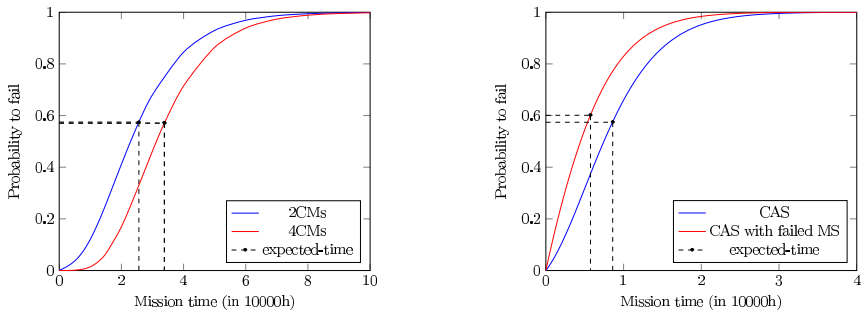
Results. The scalability of DFTCalc is shown in Table 2. We provide several comparisions with the predecessor Coral. For each case study we computed the probability that the system will fail until a given mission time t. The computation time includes each analysis step, the transformation of the DFT into I/O-IMCs, their compositional aggregation and minimisation as well as the analysis. In this experiment we used the model checker MRMC. The average speed up of DFTCalc to Coral is approximately a factor of 2.

Fig. 11 depicts the failure probability over time as well as the expected time to failure for the MCS and CAS case studies. Fig. 11(a) provides the failure probability until a mission time of 100000 hours for the 2CM and the 4CM



(a) Architecture of the MCS (b) DFT of the MCS.

Fig. 10. The multiprocessor computing system (MCS).



(a) Failure probability of the MCS over time. (b) Failure probability of the CAS over time.

Fig. 11. Reliability plots for MCS and CAS.

system. Fig. 11(b) depicts the failure probability for a fully functional CAS system and for one with a broken motor switch until a mission time of 40000 hours.

5 Conclusion

We have presented an efficient tool chain which allows to model and analyse DFTs with a number of prominent dependability metrics. The flexible architecture of DFTCalc exploits state-of-the-art techniques to compose, compress and analyse DFTs, and is easily extendable. We have conducted several case studies demonstrating DFTCalc's high performance in the analysis of DFTs.

As future work, we aim to enable the allocation of a cost structure to DFT elements. This will support the practical decision making process by allowing to analyse reliability requirements as an investment. We will further make use of DFTCalc's flexible architecture and implement additional gates to broaden it's application range to other formalisms like attack trees. The basic components will be annotated with repair rates to allow a realistic modelling of the long-term behaviour of complex systems.

Model	Tool	Time (s)	P(fail)	States	Transitions	Speedup
MCS 2CMs, t=10000	Coral	131.492	0.998963	18	55	1
	DFTCalc	55.395	0.998963	18	55	2.37371
MCS 4CMs, t=10000	Coral	339.752	0.997927	151	992	1
	DFTCalc	201.461	0.997927	151	992	1.68644
CAS, t=10000	Coral	135.155	0.0460314	16	50	1
	DFTCalc	51.267	0.0460314	16	50	2.64794
FTPP-4 , t=1	Coral	491.114	0.0192186	142	923	1
	DFTCalc	234.905	0.0192186	72	386	2.09069
FTPP-5, t=1	Coral	730.761	0.0030616	2167	27438	1
	DFTCalc	603.630	0.0030616	400	3369	1.21061

Table 2. Results of the case studies.

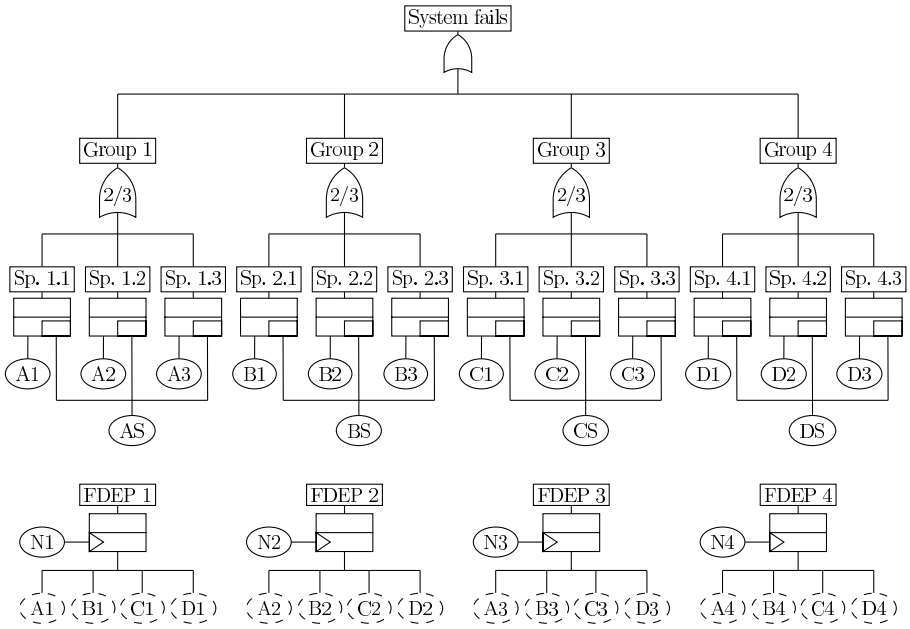


Fig. 12. The FTPP-4 case study.

References

1. C. Baier, B. Haverkort, H. Hermanns, and J.-P. Katoen. Model-checking algorithms for continuous-time Markov chains. *IEEE TSE*, 29(6):524–541, 2003.
2. C. Baier, H. Hermanns, J.-P. Katoen, and B. R. Haverkort. Efficient computation of time-bounded reachability probabilities in uniform continuous-time Markov decision processes. *Theoretical Computer Science*, 345(1):2 – 26, 2005.
3. C. Baier and J.-P. Katoen. *Principles of Model Checking*. MIT Press, 2008.
4. R. E. Barlow and F. Proschan. *Statistical theory of reliability and life testing: probability models*. Holt, Rinehart and Winston, 1975.
5. A. F. E. Belinfante and A. Rensink. Publishing your prototype tool on the web: PUPTOL, a framework. Technical Report TR-CTIT-13-15, Centre for Telematics and Information Technology, University of Twente, Enschede, June 2013.
6. H. Boudali, P. Crouzen, and M. I. A. Stoelinga. Dynamic fault tree analysis using Input/Output interactive Markov chains. In *DSN*, pages 708–717, 2007.
7. H. Boudali, P. Crouzen, and M. I. A. Stoelinga. A rigorous, compositional, and extensible framework for dynamic fault tree analysis. *IEEE TDSC*, 7:128–143, 2010.
8. H. Boudali and J. Dugan. A continuous-time bayesian network reliability modeling and analysis framework. *IEEE transactions on reliability*, 55(1):86–97, 2006.
9. H. Boudali and J. B. Dugan. A Bayesian network reliability modeling and analysis framework. *IEEE Transactions on Reliability*, 55:86–97, 2005.
10. H. Boudali, A. P. Nijmeijer, and M. I. A. Stoelinga. DFTSim: A simulation tool for extended dynamic fault trees. In *ANSS 2009*, page 31, 2009.
11. P. Buchholz, E. M. Hahn, H. Hermanns, and L. Zhang. Model checking algorithms for ctmdps. In *CAV*, pages 225–242, 2011.
12. D. Coppit and K. Sullivan. Galileo: a tool built from mass-market applications. In *International Conference on Software Engineering*, pages 750–753, 2000.
13. D. R. Cox. A use of complex probabilities in the theory of stochastic processes. *Proceedings of the Cambridge Philosophical Society*, 51(2):313–319, 1955.
14. P. Crouzen and F. Lang. Smart reduction. In *Proceedings of the 14th international conference on Fundamental approaches to software engineering: part of the joint European conferences on theory and practice of software*, FASE'11/ETAPS'11, pages 111–126, 2011.
15. H. Garavel, F. Lang, R. Mateescu, and W. Serwe. CADP 2011: A toolbox for the construction and analysis of distributed processes. *International Journal on Software Tools for Technology Transfer*, pages 1–19, 2012.
16. D. Guck, T. Han, J. P. Katoen, and M. Neuhausser. Quantitative timed analysis of interactive Markov chains. In *NFM*, volume 7226 of *LNCS*, pages 8–23, 2012.
17. H. Hermanns. *Interactive Markov Chains*. Springer, Berlin, 2002.
18. A. Horváth and M. Telek. Phfit: A general phase-type fitting tool. In *Proceedings of the 12th International Conference on Computer Performance Evaluation, Modelling Techniques and Tools*, TOOLS '02, pages 82–91, London, UK, UK, 2002. Springer-Verlag.
19. Isograph. Fault Tree +. `www.isograph-software.com/2011/software/`.
20. M. A. Johnson and M. R. Taaffe. The denseness of phase distributions. School of Industrial Engineering Research Memoranda 88-20, Purdue University, 1988.
21. J.-P. Katoen, I. S. Zapreev, E. M. Hahn, H. Hermanns, and D. N. Jansen. The ins and outs of the probabilistic model checker MRMC. *Perf. Eval.*, 68(2):90–104, 2011.

22. R. Manian, J. Bechta Dugan, D. Coppit, and K. Sullivan.
23. S. Montani, L. Portinale, A. Bobbio, M. Varesio, and D. Codetta-Raiteri. A tool for automatically translating dynamic fault trees into dynamic Bayesian networks. In *RAMS*, pages 434–441, 2006.
24. PTC. Windchill FTA. `http://www.ptc.com/product/relex/fault-tree`.
25. R. Pulungan. *Reduction of Acyclic Phase-Type Representations*. PhD thesis, Universität des Saarlandes, Saarbruecken, Germany, 2009.
26. S. Sighireanu, A. Catry, D. Champelovier, H. Garavel, F. Lang, G. Schaeffer, W. Serwe, and J. Stöcker. Lotos nt user's manual (version 2.7), 2012.
27. W. E. Veseley, F. F. Goldberg, N. H. Roberts, and D. F. Haasl. Fault tree handbook, NUREG-0492. Technical report, NASA, 1981.