

Modeling with Extended Fault Trees

Kerstin Buchacker

Institut für Informatik 3, Friedrich-Alexander-Universität

Martensstr. 3, 91058 Erlangen, Germany

kerstin.buchacker@informatik.uni-erlangen.de

Abstract

In the areas of both safety and reliability analysis the precise modeling of complex technical systems during development and for evaluation purposes is of great importance. Traditionally, fault tree models have been used to accomplish this, and, more recently, stochastic Petri-net models have begun to be employed. To provide engineers with an intuitive high-level modeling interface to Petri-nets, this paper introduces an approach combining extended fault trees for the description of the system and stochastic Petri-nets for the evaluation and analysis of the model.

1. Introduction

Growing reliability and safety requirements make it necessary to provide efficient methods and tools for modeling and evaluating large technical systems. Such tools should on the one hand have solid mathematical and formal foundations while on the other hand presenting an easily understandable modeling paradigm to the user. Fault tree models are clearly structured, but unable to model some aspects of system behavior, such as multi-state components or failure and repair dependencies between different parts of the system. It seems natural to remove these drawbacks of the fault tree modeling paradigm by combining it with stochastic Petri-nets, which are especially well suited for modeling complex stochastic dependencies. This paper presents a modeling paradigm based on fault trees and stochastic Petri-nets as an approach to solve these issues.

The rest of the paper is structured as follows: Section 2 introduces the example system used in the remainder of the paper for demonstrative purposes. Section 3 briefly introduces fault trees and stochastic Petri-nets and gives a short overview of related work on approaches using a combination of boolean and state-space-based modeling paradigms. Section 4 introduces the extended fault tree modeling paradigm. Section 5 describes how extended

fault tree models are solved. The results for the example described in Sec. 2 are presented in Sec. 6. Section 7 concludes the paper.

2. Example System

The extended fault tree modeling interface will be introduced using the cooling system described in [35] and shown in Fig. 1. The cooling system consists of two parallel pump trains in series with a heat exchanger. Each pump train is made up of a pump, with a manual valve and a filter before, and a check valve and manual valve positioned after the pump. The flow of water to/from the heat exchanger also passes through valves and a filter. If necessary, the heat exchanger may be bypassed. The bypass is operated with two motor-driven valves. If one of the pumps fails, the failure rate of the remaining pump increases due to the higher load. All components are repairable (or may be exchanged). Only one pump may be repaired at a time. The system is said to have failed, when it no longer produces adequate cooling.

3. Related Work

Sections 3.1 and 3.2 briefly explain fault trees and stochastic Petri-nets as the representatives for boolean and state-space based methods on which the method presented in Sec. 4 is based. Section 3.3 gives an overview and references to published work combining boolean and state-space-based methods.

3.1. Fault Trees

Fault trees (FT) have been widely used for both safety and reliability analysis since the early 1960s and are supported by a rich body of research [22, 27]. They provide a compact representation of a system and can be easily specified and understood by humans. *Fault tree analysis (FTA)*

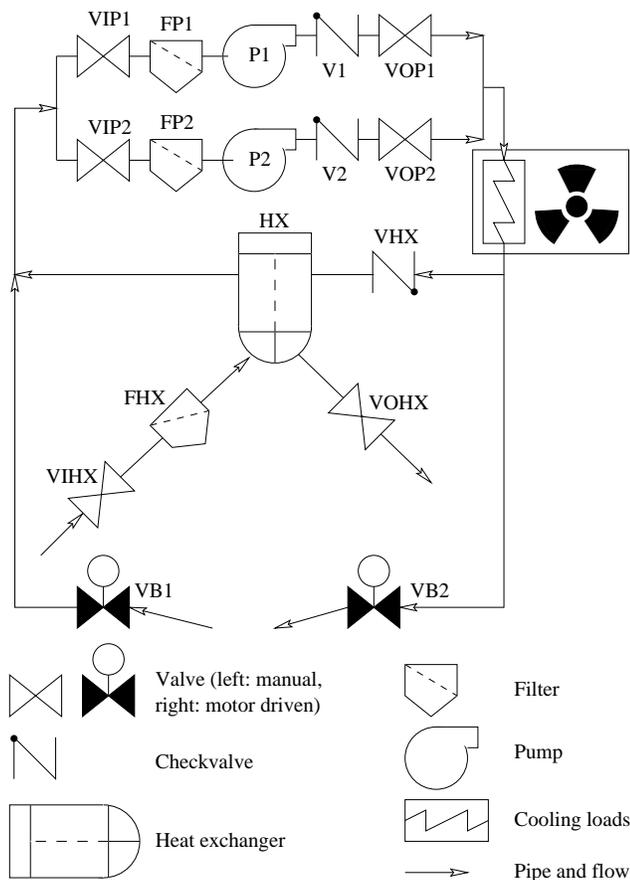


Figure 1. Block Diagram of Cooling System

answers the questions "What are the causes for an (undesired/catastrophic) event? How high is the probability that this event occurs?" The fundamental concept in FTA is the translation of a physical system into a structured logic diagram, the FT, in which certain specified causes lead to one specified *top-event* (TE) of interest. Mathematically speaking, a FT describes the interaction of a number of basic system components as a boolean function. Its graphical representation — due to the possibility of an event occurring as input to more than one gate — is not a tree in the strict graph-theoretic meaning, but a directed acyclic graph. The direction of the arcs is usually not shown explicitly, it is implicitly assumed to be from cause (input) to effect. The TE of a FT-model are usually strongly undesired system states that can occur as a result of subsystem functional faults. Examples for such undesired system states are complete system failure (reliability analysis) or system states that pose a threat to the environment (safety analysis). The FT is built top-down starting from the undesired TE. All intermediate events contributing to the TE are expanded recursively. The recursion ends at the basic system components which repre-

sent the finest resolution on the level of abstraction chosen for the model. FTA considers *all* possibilities of occurrence of a *single* event. A qualitative analysis yielding component *cutsets* is often performed to gain a better understanding of the system behavior. A cutset is a set of components such, that if all components in the cutset fail, the undesired TE occurs. For the probability of the TE to be calculated (quantitative FTA), failure probabilities or failure functions must be known for all basic components. In addition, all components must be boolean and pairwise stochastically independent. Real-world systems hardly ever comply with this condition, and the assumption of stochastic independence in the model can lead to results, which do not reflect the real behavior of the modeled system very closely.

Even though FT are not a state-space-based model, they do have system states. These system states are identified by unique combinations of failed and intact components. The probability of finding the system in a certain state Z is the product of the probabilities of finding each component c in its failed or intact state respectively. Let M be the set of those components c , which are in a failed state in Z :

$$P\{\text{System in state } Z\} = \prod_{c \in M} P\{c \text{ failed}\} \prod_{c \notin M} (1 - P\{c \text{ failed}\})$$

Obviously this behavior only holds true for pairwise stochastic independent behavior of the components. Trivial FT solution methods explicitly evaluate the above equation for all possible system states and then sum up the probabilities for those system states, for which the top event holds true. The non-trivial analytical FT solution methods do not explicitly enumerate all possible combinations of failed and intact components, which would be 2^n for a FT with n distinct basic components. Exact methods include algorithms based on binary decision diagrams [8] and modularization [7, 14, 21]. Approximative methods are most often based on truncation [22].

A number of articles concerning the extension of fault trees with multi-failure-mode components has been published. [15, 26, 28, 29] are concerned with dual-failure-mode components only, non-degrading components with one intact and several failure states are treated by [6], whereas [20, 34, 36] all focus on degrading components. In this paper, FT with such multi-state components are called *multi-state* FT (mFT). Since extended fault trees allow both non-degrading and degrading multi-state components (see Fig. 3 for an example), an approach based on [6, 20, 34, 36] is used [5]. The papers cited here also present boolean extensions to FTA for the solution of mFT. Multi-state components introduce the problem, that even though the components are still pairwise stochastic independent, single events, such as "component A is in state B" may not be. Specifically, single events belonging to the same component

are stochastically dependent, since the exclude each other (a multi-state component, too, must be in exactly one state at any point in time).

3.2. Stochastic Petri-Nets

Stochastic Petri-nets (SPN) are well suited for modeling complex and time-dependent stochastic interactions between events. Although the need for flexible formal methods for reliability analysis has been recognized, SPN are only recently beginning to spread from research to industry applications. A formal introduction to the Petri-net nomenclature can be found in [25], for SPN refer to [1]. Here a short informal overview of net syntax and semantics suffices (Fig. 2). A SPN is most often shown as a bipartite graph consisting of *transitions* (often representing actions) which are influenced by certain conditions (represented by *places*). The completion of an action (*firing* of a transition) moves indicators (*tokens*) around and thus changes the state of the system and which conditions hold. *Arcs* connecting places and transitions model the complex relationships between them. A certain distribution of tokens in places is called a *marking*. An (often random) non-zero firing delay is associated with *timed* transitions, whereas for *immediate* transitions the firing delay is zero and they instead have a firing probability. A transition is *enabled* and may fire, when the number of tokens in its input places is larger or equal to the weights of the corresponding input arcs and the number of tokens in any places connected to the transition by inhibitor arcs contains less tokens than the inhibitor arc weight. When several timed transitions are enabled in parallel, the one with the shortest firing delay fires. In Fig. 2 (top) transition T1 is enabled. The firing of T1 will remove the two tokens from place P2 and put a single token into P3 (compare Fig. 2, bottom). This in turn will enable the two immediate transitions T2 and T3 in parallel. In the case of the parallel enabling of several immediate transitions, the one to fire is chosen based on the firing probability of the transitions involved. Additionally, immediate transitions always have priority over timed transitions. The firing of a transition equals the change from one marking to the next. The probability of finding the system in a certain state, i.e. the SPN in a certain marking, can be calculated using either analytical methods based on the theory of stochastic processes (for many classes of SPN Markov chains suffice) or simulation.

SPN, like all state-space-based approaches, suffer from the drawback that for an exact analytical solution, the complete state-space must be generated. Contrary to FT, the size of the state-space can not usually be predicted from the number of SPN-elements and the local rules governing the transition firings. Thus, relatively small and simple SPN can have extremely large state-spaces.

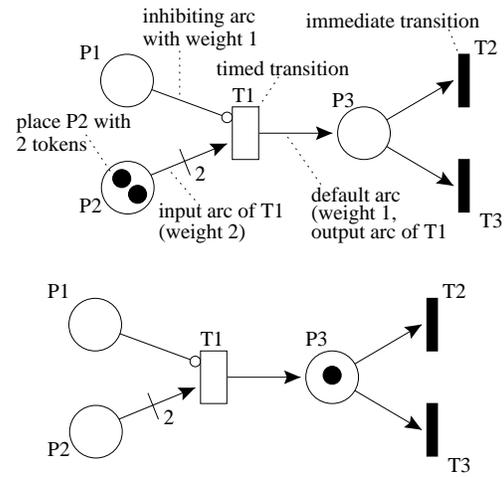


Figure 2. SPN elements and firing

3.3. Previous Published Work

Considering the above it seems only natural to combine FT, as a representative for boolean methods, with state space based methods such as SPN or Markov chains and exploit the advantages of both modeling paradigms. Previous work treating the combination of FT and SPN in hierarchical models includes [12, 30, 32]. Whereas [30] is rather theoretic, [32] describes a graphical modeling interface based on hierarchical reliability block diagrams (RBD). Each block of the RBD can be refined recursively. A basic block may either be a single component, or a k-of-n-system. The model is automatically solved, choosing the appropriate boolean or state-space-based method for the solution of each basic block. [12] models fault-tolerant systems (including soft- and hardware in the model) using FT for the software and Markov chains for the hardware.

Work about the transformation of traditional FT into Petri-Nets may be found in [16, 23, 37]. [37] constructs a Petri-Net for a FT for an airbag-inflator and shows how to include error-detection into the resulting Petri-Net. [16, 23] describe general algorithms for transforming traditional FT into SPN. Neither address the common SPN problem of state-space explosion.

An approach of transforming a certain class of high-level fault trees into SPN is proposed in [3].

The extension of FT with additional gates, cold-spare- (CSP), functional-dependency- (FDEP), sequence-enforcing- (SEQ) and priority-and-gate was first proposed in [10] and refined in a number of papers, the more recent of which are [13, 17, 24]. The term *dynamic* FT was coined for these models. Dynamic FT make the modeling of certain stochastic dependencies between the components

possible. A CSP-gate is used to model cold-spares dependencies. All inputs to this gate must be basic components. The output of a CSP-gate becomes true when all units (primary and spares) have failed. The SEQ-gate must also have basic components as inputs. It forces its inputs to fail in a predefined sequence. The SEQ-gate was combined with the FDEP-gate in some of the authors' previous papers to model cold-spares dependencies. The dynamic FT model is solved analytically by transforming those parts of the model containing stochastic dependencies into Markov chains. The dynamic FT models given in the papers are all systems containing only boolean, non-repairable basic components.

4. Introducing Extended Fault Trees

This section describes the modeling interface of extended FT (eFT). eFT extend traditional FT by allowing multi-state components and stochastic dependencies, namely repair and failure dependencies. The different approaches to extend FT with multi-state components have been combined and formalized into a coherent framework making it possible to use both degrading and non-degrading components (either of which may be repairable). Apart from allowing multi-state-components, eFT differ from dynamic FT by allowing not only CSP-dependencies but a number of different repair dependencies as well as failure dependencies changing not the state but the failure rate of the affected component.

Using the eFT modeling interface, the modeler can describe the system structure, system component properties and stochastic interaction of basic system components in a clearly structured and intuitive way. The mapping of the eFT model-description and generation of the underlying SPN, which may be quite complex, is done automatically and is completely transparent to the modeler. The eFT modeling paradigm is introduced using the cooling system described in Sec. 2.

The eFT model consists of three parts: Firstly, the definition of components (Sec. 4.1), secondly the structure of the system modeled using an eFT (Sec. 4.2) and thirdly information about the components' stochastic interactions, which cannot be modeled by a traditional FT (Sec. 4.3).

4.1. Modeling the Components

The eFT modeling paradigm allows degrading and non-degrading components with a single intact state and several fault states to be modeled. The fault states are either mutually exclusive (non-degrading components) or represent different states of degradation. Switches or valves with the states "intact", "stuck closed" and "stuck open" are examples for the first case. Pumps or motors, on the other hand, may be modeled as degrading components with states such

as "intact", "1/2 power", "1/4 power" and "stopped." Figure 3 shows the diagrams of possible state changes for one of the pumps of the cooling system modeled as degrading component and one of the non-degrading valves. The term *fault states* here is understood to include all of a component's failed and degraded states and excluding the intact state.

The behavior of a component concerning its failures over time is modeled by failure and repair rates associated with each fault state. Obviously a component with mutually exclusive fault states cannot change directly from one fault state to another. Conversely, degrading components can change directly from a degraded state to an even more degraded state or may fail spontaneously from not fully degraded states into the fully degraded state with a certain failure rate. Additionally a repair rate may be associated with every fault state. Repair will reset both types of components into their intact state. In Fig. 3 dashed lines represent state changes due to repair, whereas solid lines are associated with failure. Due to the way the behavior of a component is defined, a component must be in exactly one state at any point in time.

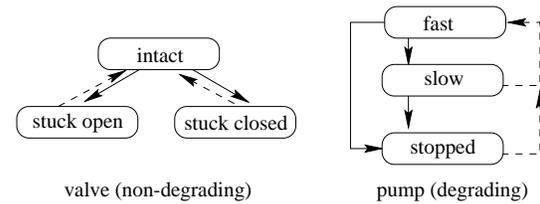


Figure 3. Possible state changes of a non-degrading and a degrading component

4.2. Modeling the System Structure

The structure of the system is mostly modeled as classical FT extended by the ability to have multi-state components as leaves. Figure 4 shows the structure of the eFT for the cooling system from Sec. 2. The functional interaction of the components is modeled using boolean gates. In principle, any boolean function is possible as gate function, but usually only a few are used. The eFT introduced in this paper include "and"-, "or"-, "not"- and "k-of-n"-gates. The two-state components of classical FT are modeled by boolean variables taking the value 1 (true) when the component is in the failed state, and 0 (false) when the component is in the intact state. Since the eFT allows multi-state components, a single boolean variable per component is not enough. Therefore multi-state components have one boolean variable per state. To identify which of the variables of the component is an input to a gate on a higher

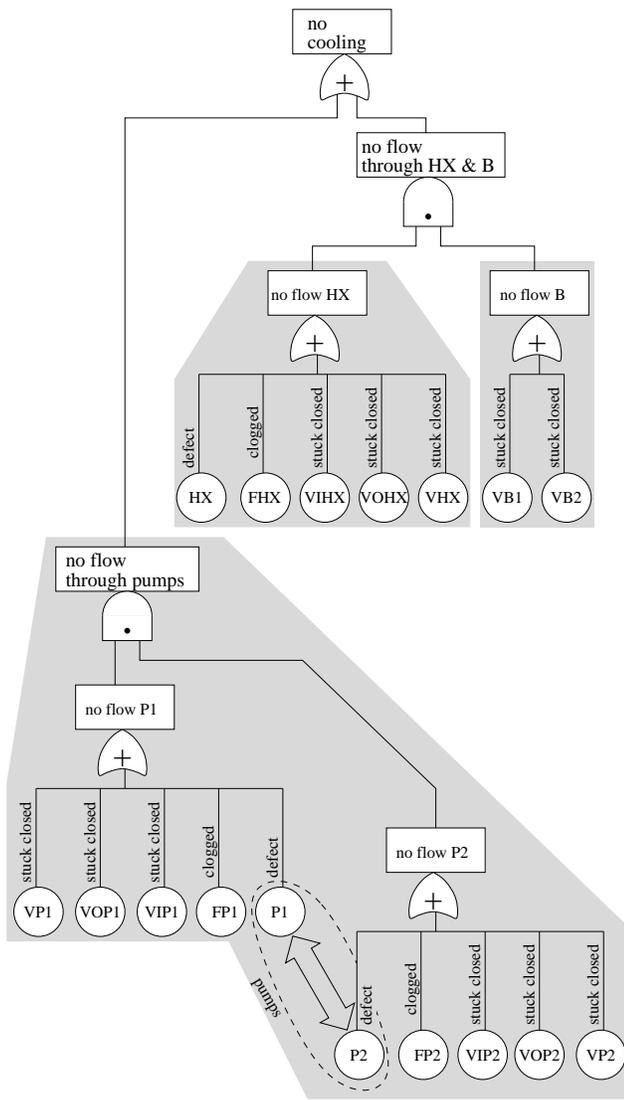


Figure 4. eFT for the system from Sec. 2

level, the name of the appropriate state is given next to the arc in the graphical representation (compare Fig. 4).

The event actually described by a leaf of the eFT depends on whether the component is degrading or not. For a non-degrading component, the event represented is "component A is in state B". For degrading components, the event is "component A is in state B or more degraded" if B is a fault state, otherwise the event is identical to the one for non-degrading components. This definition of events represented by a single leaf was chosen, as it corresponds to the behavior the modeler would intuitively expect.

A single multi-valued variable could also have been chosen to represent a multi-state component in the mathematical model. This approach was discarded, as the standard

boolean gates used here would have had to be replaced by user defined discrete functions, which would have placed too great a burden on the user.

As long as there are no additional stochastic interdependencies between the components than those caused by the fact, that more than a single failure event is associated with any one component, the eFT can still be evaluated qualitatively using the boolean methods for multi-state fault trees [5].

4.3. Modeling the Stochastic Dependencies

In real-world systems, a number of different stochastic dependencies can be found. Perhaps the best known is the *common cause failure*, the failure of several system components at the same point in time caused by the occurrence of a single event. This type of common cause failure can also be modeled by the functional-dependency-gates of dynamic FT. A common cause often is the failure of a component or subsystem. Other dependencies include that — due to the failure of some other part — a component suffers additional stress which may increase its failure rate. When modeling highly reliable systems, many of which can be repaired without needing to be taken offline, repair should be included in the model for more accurate results. Mapped to the real world, the FTA assumption of the stochastic independence of basic events means a dedicated repair team for every repairable component, a condition not usually found in real-world-systems. Repair dependencies therefore arise through the sharing of repair resources between several components.

In the eFT modeling paradigm, failure dependencies are defined by their cause and effects. The cause may be a component state change or an intermediate event modeled by another eFT. Two different classes of effects are defined. One is a state change occurring in one or more components. As an example, imagine both pumps in the cooling system example could fail due to power system outage. In the eFT modeling language [4] this could be described as in Fig. 5 (top). The other possibility occurs in the cooling system example: When one of the pumps fails completely, the other's failure rate increases due to the additional stress. This is shown in Fig. 5 (bottom).

Repair dependencies are defined by the number of available repair resources and the group of components depending on this pool of resources. Repair resources may be technicians, special tools or spare parts. Resources may be classified as reusable (e.g. technicians, tools) or non-reusable (e.g. spare parts). Reusable resources are replaced into the resource pool after repair completion, non-reusable resources are not. Non-reusable resources may be replaced in storage. This is modeled with a refill-rate. Additionally the rates, with which a component failure is detected and the

```

DEFINE FAILDEP poweroutage :
  CAUSE = power.out ;
  EFFECT = STATECHANGES P1 -> stopped,
          P2 -> stopped;
END

DEFINE FAILDEP Pump1 :
  CAUSE = P1.stopped ;
  EFFECT = RATECHANGES P2:*2 ;
END
DEFINE FAILDEP Pump2 :
  CAUSE = P2.stopped ;
  EFFECT = RATECHANGES P1:*2 ;
END

```

Figure 5. Failure Dependencies: State-change (top), Rate-change (bottom)

repair is begun, and the rate of repair completion must be given. Since the eFT is transformed into a SPN for analysis, it is also possible to model different repair strategies (as suggested in [23]). The repair dependency of the two pumps in the cooling system example is shown in Fig. 6.

```

DEFINE REPAIR GROUP Pumps :
  MEMBERS = P1, P2 ;
  REUSABLE RESOURCES = 1;
  DETECTION RATE = GROUP: nul;
  COMPLETION RATE = GROUP: rho1;
  SDISCIPLINE = RANDOM ;
END

```

Figure 6. Repair Dependency

5. Solving Extended Fault Tree Models

Section 5.1 first gives an overview of the algorithm. Sections 5.2 to 5.4 examine approaches useful for solving large eFT models.

5.1. Overview of the Algorithm

The quantitative analysis of an eFT-model, i.e. calculating the probability of the TE, is done in several steps. The flow of data (boxes) between the two different levels of abstraction (separated by a dashed line) and parts of the tool (ovals) is shown in Fig. 7. For small models the shaded path through Fig. 7 is taken. The complete model is mapped onto an equivalent SPN representation in a first step using the algorithm detailed in [5]. The SPN generated by the conversion algorithm is then solved with the tool PANDA [2] to produce a SPN solution. A reverse mapping algorithm, the details of which may also be found in [5], translates the

SPN results calculated by PANDA back to the level of abstraction of the eFT representation of the model. The mapping of the eFT model onto SPN [4] is done automatically by the tool and does not require any interaction or knowledge about SPN from the user. The system intact state (all components are in their respective intact state) is chosen as the default initial marking of the generated SPN, but any other valid system state is possible. The reverse mapping algorithm takes the results from the SPN analysis — which are basically probabilities of occurrence for the SPN markings — and sums up the probabilities of all those markings that are relevant to the TE of the original eFT model.

When using modularization the flow of data is a little less direct. For a more detailed explanation see Sec. 5.4.

When modeling large real-world systems, the number of markings of the SPN generated from the eFT model can grow so large, that the effort needed to evaluate the SPN becomes intolerable. The following sections introduce approaches which reduce the number of SPN markings.

5.2. Truncation

Truncation is widely used in traditional FTA to calculate approximated solutions for large FT. In traditional FTA, truncation criteria are usually based on the size or probability of component cutsets. These criteria cannot be used, when applying truncation to eFT, since cutsets are not calculated. The user must be able to specify truncation criteria from within the eFT level of abstraction. Three criteria were defined. The user may specify the maximum number of component failures to be taken into account for the calculation. In this case, care must be taken to treat failure dependencies and degrading components correctly. Another option is to allow no more component failures as soon as the TE has occurred. If the user has specified subsystems in the model it is also possible to disallow component failures within subsystems once the subsystem has failed. The latter two criteria will not usually reduce the state space of models of fault-tolerant redundant systems very much, as the TE or a subsystem failure will only occur, when a large number of components in the (sub)system has already failed. When applying the first criterium to the model of a fault-tolerant system and choosing a small number for the maximum component failures taken into account for the calculation, it is possible, that the TE does not occur at all. This behavior follows directly from the fault-tolerant nature of the system, which is designed not to fail when only a few of its components are defect. The truncation criterium chosen on the eFT level is included in the mapping onto the SPN. Solution using truncation also follows the shaded path through Fig. 7, with the mapping and reverse mapping algorithms adapted for truncation. Such a truncated SPN effectively has a smaller state-space than the non-truncated SPN, since

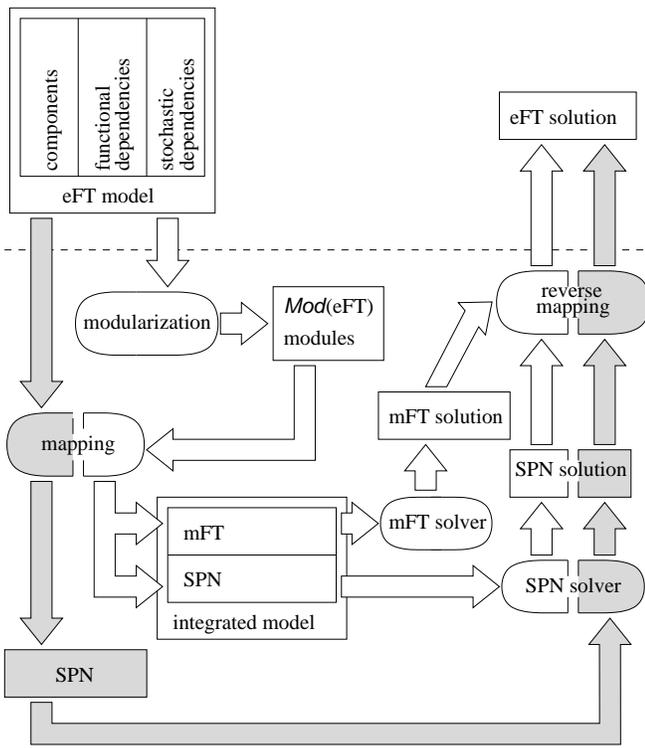


Figure 7. Flow of data

a number of the markings are missing. The number and probabilities of the markings missing from the truncated SPN relate directly to the error incurred by using truncation. From this obviously follows, that the exact error cannot be calculated without the knowledge of all markings of the non-truncated SPN. Comparative experiments with automatically generated scalable examples showed, that the error incurred was well with tolerable limits [5]. Research on the truncation of SPN and Markov chains has been done by [18, 19, 33].

For non-repairable systems, the error made by truncation only affects markings, from which no other marking can be reached in the truncated SPN when from the equivalent marking in the non-truncated SPN further markings may be reached. This behavior is explained in [9] and is used in the calculation of dynamic FT.

5.3. Folding

Folding is a term used in the Petri-net world. It describes certain simplifications of Petri-nets entailing a loss of information and a reduction of possible markings. The term folding is derived from the picture, that the Petri-net is folded (much like a piece of cloth) such that several e.g. places come to rest stacked on top of each other. These stacked

places are then melted into a single place.

Folding can be used for eFT solution, when the eFT-model contains a number of identical redundant components, for which it is not important to be able to distinguish between the components. A good example would be the disks in a RAID. Groups of components which may be folded in the SPN can be marked in the eFT model, without needing to know details of the folding process. Research in this area is done by [3].

5.4. Modularization

Modularization was originally developed for the classical FTA [7, 14, 21]. It follows the well known "divide and conquer" approach and partitions the eFT into *modules*. The time to find *all* modules of a FT increases exponentially with the number of components in the FT. Since this is contrary to the goal of modularization — speeding up the computation — we concentrate on modules which are also subtrees. A subtree is a module, when all gates and leaves within the subtree (except the root of the subtree) have no connection to gates or leaves external to this subtree. Often, modules correspond to subsystems in the real-world. The probability of the root event of a module can be calculated independently from the rest of the tree. For the calculation of the TE-probability, the module can now be treated as though it were a simple component. A linear algorithm to find subtree-modules was proposed in [14]. The algorithm uses the graphical representation of a FT and thus, depending on the representation chosen for the underlying boolean function, does not always find all possible modules of this type. The algorithm in [14] was extended to work with eFT. A few additions addressing the problem mentioned above were also included.

An eFT model can be partitioned into modules using a very similar approach. Additionally to the condition mentioned above, no component within a module may be stochastically dependent on a component not in the module. After modularizing the eFT the modules to be used for further calculations are chosen and the corresponding $Mod(eFT)$ is created, by replacing the modules with leaves. Modules of the eFT in Fig. 4 are shaded. Note that the event "no flow through HX & B" is also the top of a module. Thus there are several possibilities of creating $Mod(eFT)$. One is shown in Fig. 8 (the shaded modules from Fig. 4 are replaced by leaves). The integrated model would in this case consist of the FT in Fig. 8, the mFT for the modules "no flow HX" and "no flow B" and the SPN generated from the module "no flow through pumps". Those modules containing stochastic dependencies are mapped onto SPN, thus creating the integrated model consisting of one or more mFT and SPN. When the modules in the original eFT have been replaced by leaves and the resulting eFT does not contain

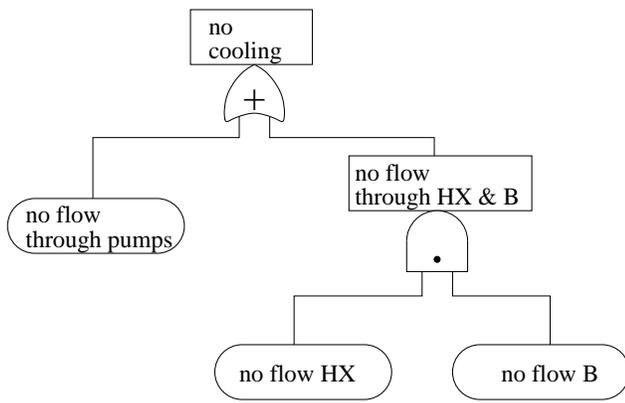


Figure 8. Mod(eFT) from Fig. 4

any further stochastic dependencies between its leaves, it is in fact a mFT. The TE-probability can then be calculated directly using methods applicable to mFT. The overhead for finding the modules and computing the TE solution from the module solutions is small, so the savings in both memory and time are significant, when modularization is used.

Contrary to truncation and folding, modularization produces an exact solution with no loss of information.

6. Practical Experience

The eFT approach was implemented in a prototype tool and applied to a number of examples. Two classes of examples were used: Firstly, eFT models were generated automatically to facilitate testing the approach on similarly structured models of different sizes. Secondly several examples from literature [11, 31, 35] were solved. The tests showed that modularization is the method of choice when it comes to solving large models. Although modularization did not work well on all of the automatically generated examples, the systems taken from practical literature were well suited for modularization. The automatically generated models contained stochastic dependencies and repeated events distributed regularly throughout the model, which makes modularization difficult or impossible. It seems, that in real-world systems, dependencies are most often contained within subsystems. The solution of the cooling system from Sec. 2 is shown in Fig. 9. As is to be expected from a fully repairable system a steady state is reached after a certain time.

7. Conclusions

This paper introduces eFT as a modeling paradigm combining the intuitive structure of FT with the modeling power

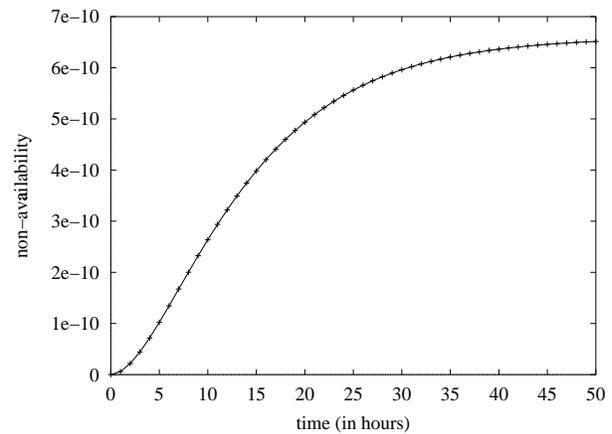


Figure 9. Non-availability of the cooling system

of SPN. eFT allow modeling systems with multi-state components and stochastic interactions between components. Several approaches used to handle large models are discussed. The eFT modeling approach has been used successfully on a number of examples.

References

- [1] M. Ajmone Marsan, G. Balbo, G. Conte, S. Donatelli, and G. Franceschinis. *Modelling with Generalized Stochastic Petri Nets*. John Wiley & Sons, Chichester, 1995.
- [2] S. Allmaier and S. Dalibor. PANDA — petri net analysis and design assistant. In *Tools Descriptions, 9th International Conference on Modeling Techniques and Tools for Computer Performance Evaluation*, pages 58–60, 1997.
- [3] A. Bobbio, G. Franceschinis, R. Gaeta, and L. Portinale. Exploiting petri nets to support fault tree based dependability analysis. In *International Workshop on Petri Nets and Performance Models*, 1999.
- [4] K. Buchacker. Combining fault trees and petri-nets to model safety-critical systems. In A. Tentner, editor, *High Performance Computing 1999*, pages 439–444. The Society for Computer Simulation International, 1999.
- [5] K. Buchacker. Definition und Auswertung erweiterter Fehlerbäume für die Zuverlässigkeitsanalyse technischer Systeme. Technical Report 33/3, Friedrich-Alexander-Universität Erlangen-Nürnberg, Institut für Informatik, July 2000.
- [6] L. Caldarola. Grundlagen der Booleschen Algebra mit beschränkten Variablen. Technical Report KfK 2915 / EUR 6405d, Kernforschungszentrum Karlsruhe, Institut für Reaktorentwicklung, Projekt Nukleare Sicherheit, 1979.
- [7] P. Chatterjee. Modularization of fault trees: A method to reduce the cost of analysis. In *Reliability and Fault Tree Analysis*, pages 101–126. SIAM, Philadelphia, 1975.

- [8] O. Coudert and J. C. Madre. Metaprime: An interactive fault-tree analyzer. *IEEE Transactions on Reliability*, 43(1):121–127, Mar. 1994.
- [9] J. B. Dugan. Fault trees and imperfect coverage. *IEEE Transactions on Reliability*, 38(2):177–185, June 1989.
- [10] J. B. Dugan, S. J. Bavuso, and M. Boyd. Fault trees and sequence dependencies. In *Annual Reliability and Maintainability Symposium*, pages 286–293, 1990.
- [11] J. B. Dugan, S. J. Bavuso, and M. A. Boyd. Dynamic fault-tree models for fault-tolerant computer systems. *IEEE Transactions on Reliability*, 41(3):363–377, Sept. 1992.
- [12] J. B. Dugan and M. R. Lyu. Dependability modeling for fault-tolerant software and systems. In M. Lyu, editor, *Software Fault Tolerance*, pages 109–138. John Wiley & Sons, Chichester, 1995.
- [13] J. B. Dugan, R. Manian, K. J. Sullivan, and D. Coppit. Combining various solution techniques for dynamic fault tree analysis of computer systems. In *Proceedings Third IEEE International High-Assurance Systems Engineering Symposium*, pages 21–28, Nov. 1998.
- [14] Y. Dutuit and A. Rauzy. A linear-time algorithm to find modules of fault trees. *IEEE Transactions on Reliability*, 45(3):422–425, Sept. 1996.
- [15] K. Gopal, K. K. Aggarwal, and J. S. Gupta. Reliability analysis of multistate device networks. *IEEE Transactions on Reliability*, R-27(3):233–236, Aug. 1978.
- [16] B. Grams. Entwurf und Implementierung von anwendungs-basierten Methoden zur Dependability-Analyse basierend auf stochastischen Petri-Netzen. Master's thesis, Institut für Mathematische Maschinen und Datenverarbeitung der Friedrich-Alexander-Universität Erlangen-Nürnberg, Lehrstuhl für Informatik III (Rechnerstrukturen), Apr. 1995.
- [17] R. Gulati and J. B. Dugan. A modular approach for analyzing static and dynamic fault trees. In *Annual Reliability and Maintainability Symposium*, pages 57–63, 1997.
- [18] B. R. Haverkort. Approximate performability and dependability analysis using generalized stochastic Petri nets. *Performance Evaluation*, (18):61–78, 1993.
- [19] B. R. Haverkort. In search of probability mass: Probabilistic evaluation of high-level specified markov models. *The Computer Journal*, 38(7):521–529, 1995.
- [20] Y. Kai. Multistate fault-tree analysis. *Journal of Reliability Engineering and System Safety*, 28(1):1–7, 1990.
- [21] T. Kohda, E. J. Henley, and K. Inoue. Finding modules in fault trees. *IEEE Transactions on Reliability*, 38(2):165–176, June 1989.
- [22] W. S. Lee, D. L. Grosh, F. A. Tillman, and C. H. Lie. Fault tree analysis, methods, and applications — a review. *IEEE Transactions on Reliability*, R-34(3):194–203, Aug. 1985.
- [23] M. Malhotra and K. S. Trivedi. Dependability modeling using Petri nets. *IEEE Transactions on Reliability*, 44:428–440, 1995.
- [24] R. Manian, D. W. Coppit, K. J. Sullivan, and J. B. Dugan. Bridging the gap between systems and dynamic fault tree models. In *Annual Reliability and Maintainability Symposium*, 1999.
- [25] T. Murata. Petri nets: Properties, analysis and applications. *Proceedings of the IEEE*, 77(4):541–580, Apr. 1989.
- [26] L. B. Page and J. E. Perry. Reliability of networks and three state devices. *Microelectronics and Reliability*, 27(1):175–178, 1987.
- [27] N. H. Roberts, W. E. Vesely, D. F. Haasl, and F. F. Goldberg. Fault tree handbook. Technical Report NUREG-0492, 1981.
- [28] N. Satoh, M. Sasaki, T. Yuge, and S. Yanagi. Reliability of 3-state device systems with simultaneous failures. *IEEE Transactions on Reliability*, 42(3):470–477, Sept. 1993.
- [29] B. Singh and C. L. Proctor. Reliability analysis of multistate device networks. In *Annual Reliability and Maintainability Symposium*, pages 31–35, 1976.
- [30] B. Specker. Evaluation of fault-tolerant systems using stochastic petri nets and fault trees. In *European Simulation Symposium*, pages 83–87, 1992.
- [31] K. Stecher. Program ZUSIM — reliability assessment of complex technical systems. in Fehlertolerante Rechensysteme: Innovative Architekturen, Modellierung und Bewertung, Interner Bericht IMMD3, 1993.
- [32] A. T. Tai, H. Hecht, K. S. Trivedi, and B. Zhang. Toward accessibility enhancement of dependability modeling techniques and tools. In *Proceedings of the 27th Annual International Symposium on Fault Tolerant Computing, Seattle, WA*, pages 37 – 41, June 1997.
- [33] A. P. A. van Moorsel and B. R. Haverkort. Probabilistic evaluation for the analytical solution of large markov models: Algorithms and tool support. *Microelectronics and Reliability*, 36(6):733–755, 1996.
- [34] A. P. Wood. Multistate block diagrams and fault trees. *IEEE Transactions on Reliability*, R-34(3):236–240, Aug. 1985.
- [35] L.-M. Xing, K. N. Fleming, and W.-T. Loh. Comparison of markov model and fault tree approach in determining initiating event frequency for systems with two train configurations. *Journal of Reliability Engineering and System Safety*, 53(1):17–29, 1996.
- [36] Xue Janan. On multistate system analysis. *IEEE Transactions on Reliability*, R-34(4):329–337, Oct. 1985.
- [37] S. K. Yang and T. S. Liu. Failure analysis for an airbag inflator by Petri nets. *Quality and Reliability Engineering International*, 13:139–151, 1997.