Combining Formal Methods and Safety Analysis - The ForMoSA Approach [⋆]

Frank Ortmeier, Andreas Thums, Gerhard Schellhorn, and Wolfgang Reif

Lehrstuhl für Softwaretechnik und Programmiersprachen,
Universität Augsburg, D-86135 Augsburg
{ortmeier, thums, schellhorn, reif}@informatik.uni-augsburg.de

Abstract. In the ForMoSA project [17] an integrated approach for safety analysis of critical, embedded systems has been developed. The approach brings together the best of engineering practice, formal methods and mathematics: traditional safety analysis, temporal logics and verification, and statistics and optimization.
These three orthogonal techniques cover three different aspects of safety: fault tolerance, functional correctness and quantitative analysis. The ForMoSA approach combines these techniques to answer these safety relevant question in a structured and formal way. Furthermore, the tight combination of methods from different analysis domains yields results which can not be produced by any single technique.
The methodology was applied in case studies to different industrial domains. One of them is the height control of the Elbtunnel in Hamburg [16] from the domain of electronic traffic control, which we present as an illustrating example.

Key words: fault tree analysis, dependability, optimization, safety analysis, embedded systems

1 Introduction

Safety is an important property of embedded systems. Their complexity and the role of software used in them constantly increases. This makes new techniques necessary, that can deal with complex hardware-software systems. Safety covers many different aspects like functional correctness, fault tolerance or risk minimization. Therefore, different techniques are necessary to deal with different aspects.

⋆ This work is partly sponsored by the German Research Foundation's priority program "Integrating software specification techniques for engineering applications".

To account for this we combine methods from three different domains: software engineering, engineering and mathematics. To assure a high level of confidence formal methods from software engineering have been used. Safety relevant properties may be analyzed by safety analysis techniques from engineering. Mathematical methods from statistics and optimization can yield good quantitative approximations for risk.

In section 2 we develop a new technique to find failure modes. Failure modes are an important input for many safety analysis techniques [13, 20] like failure modes and effects analysis (FMEA) or fault tree analysis (FTA). The combination of FTA and formal methods results in formal FTA. Formal FTA, described in Sect. 3 integrates the rigorous proof concepts of formal methods into the reasoning process of FTA. Bringing statistical distributions into fault trees yields parameterized risk approximations. This allows to quickly evaluate the effects of different component configurations or changing working environments. This technique is demonstrated in Sect. 4. Mathematical safety optimization is presented in Sect. 5. It produces an optimal configuration for the free parameters of a system. This helps to reduce the time required for testing and fine tuning a system.

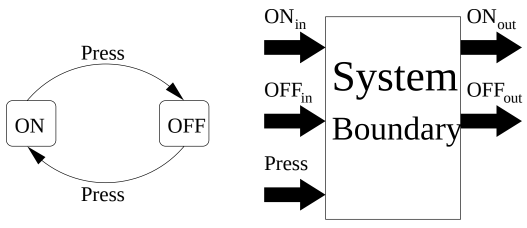
Our experience is, that no single technique can produce all the results, which a combined approach can produce. A suitable methodology, which tries to maximize the mutual benefits is presented in Sect. 6. An application of this methodology on a real-world case study - the electronic height control of the Elbtunnel - is shown in section 7. Section 8 concludes the paper.

2 Failure-sensitive Specification

Failure modes are important for many safety analysis techniques. A failure mode is a specific way in which a component may fail. The leaves of all fault trees are failure modes. The starting column of FMEA also contains the failure mode to be examined. Normally failure modes are specified in an informal and manual way. Failure-sensitive specification [15] allows to find and characterize failure modes in a structured formal way.

For better understanding we will give an example of a simple switch in parallel with the formal definitions of failure-sensitive specification. Every system component reacts on a certain set of (boolean) input signals and produces some (boolean) output signals as answers to these inputs. Let Ω be the set of all possible combinations of signals - called the chaotic model.

Ω does not contain any functional properties. Figure 1 shows this on the example of a simple switch. On the left a formal model of the switch is shown. The box in the middle shows the relevant input and output signals. For the example: The behavior of a switch depends on whether it was ON (=signal ON_{in}) or OFF (=signal OFF_{in}) and if it is pressed (=signal Press) or not. As reactions one may observe whether the switch is ON (=signal ON_{out}) or OFF (=signal OFF_{out}).



ON_{in}	OFF_{in}	Press	ON_{out}	OFF_{out}
				X
			X	
			X	X
...	...	...	...	...
X	X	X	X	
X	X	X	X	X

Fig. 1. Chaotic model of a switch

The right column shows the chaotic model. It contains all possible combinations of inputs and outputs. Each row of the tabular is to be read as a scenario, where ON_{in}, OFF_{in} and Press are the input signals and ON_{out} and OFF_{out} are output signals, which are produced by the component in response to the inputs. An "X" means the signal is present and a blank means the signal is not present. So each IO-scenario $\Phi \in \Omega$ is represented by a pair (s,t) of sets of input and output signals. This (finite) set of scenarios is the starting point for a failure-sensitive specification. Formally, the chaotic model is defined as Cartesian product of input and output signals.

Definition 1 (Chaotic model). *Let the system boundaries $SIG := IN \cup OUT$ be described by a set of input signals IN (called input set) and a disjoint set of output signals OUT (called output set). The chaotic model of the system is the Cartesian product of the power sets of the set of input and output signals. $\Omega := \mathcal{P}(IN) \times \mathcal{P}(OUT)$. The elements of Ω are called scenarios.*

Specification information is now subsequently added by rules and the set Ω is restricted according to these requirements. Formally, the specification process is defined as follows:

Definition 2 (Specification Rules). *A specification rule SpecRule is a boolean expression over SIG. Each rule can be identified with a corresponding relation, $SPECRULE_i = \{\Phi \in \Omega | SpecRule_i(\Phi)\}$.*

Specification rules are often formulated as properties, which should hold for the component like "if the switch was off and pressed, then it must be on"(RULE1).

In the example, RULE1 removes all rows in which an "X" is present in the "OFF_{in}" and in the "press", but not in the "ON_{out}" column. Note, that the complement of a rule - against Ω - is then a failure mode. This is because a definition of failure is "faulty behavior" or "behavior against the specification". For example the failure mode "fails to close" is the complement of RULE1. Finding a complete list of failure modes is thus equivalent to finding a complete list of SpecRules.

Specification is done by set intersection of the specification rules. The failure-sensitive specification FS for n SpecRules is

$$FS := \bigcap_{j=1}^{n} SPECRULE_j$$

.

In real problems boolean expression are not enough. Therefor, we define projection operators $\pi : SIG \rightarrow PL(SC)$. PL(SC) denotes the predicate logics formulae over the signature of the statechart SC. These operators associate the elements of the FS with a predicate logic formula over the statechart.

In the example these operators are simple identities. E.g. ON_{in} would be associated with the (boolean) state variable *ON*. In other applications these projections are not simple identities. For example in the radio-based level crossing [12] case study the signal "before critical point" is associate with "pos > CP".

Assume, that a statechart model of the intended behavior of the component is available. We can then define completeness of failure modes formally. The completeness criterion will be that, "the statechart shows exactly the behavioral patterns of the failure-sensitive model and vice versa". Or in other words: the IO-relation of the state-chart model is equal to the failure-sensitive model. To prove this equivalence we identify each scenario $\Phi = (s, t) \in FS$ with an ITL-formula $F(\Phi)$ over the statechart model:

$$F(s,t) = \bigwedge_{x \in s} \pi(x) \wedge \bigwedge_{x \in IN \setminus s} \neg\, \pi(x) \wedge \circ\, (\bigwedge_{x \in t} \pi(x) \wedge \bigwedge_{x \in OUT \setminus s} \neg\, \pi(x))$$

If in any trace σ of the statechart the formula $F(\Phi)$ is valid at a certain time t, then this means that the transition (σ_t, σ_{t+1}) shows the IO-behavior Φ. Now, behavioral equivalence of failure-sensitive specification FS and statechart SC is defined as follows:

Definition 3 (Behavioral equivalence). *A statechart SC and a failure-sensitive specification FS are behaviorally equivalent ($SC \cong FS$) , if*

1. *on all runs of SC, always $\bigvee_{\Phi \in FS} F(\Phi)$ holds.* ("$FM \leq FS$")
2. *for every $\Phi \in FS$ exists a run of SC, for which at some time $F(\Phi)$ holds.* ("$FS \leq FM$")

So if SC is a model of the intended system and if $SC \cong FS$ then the list of specification rules is complete - and so is the list of failure modes. Note, that this list is not unique. It is, for example, always possible to combine two rules into one with logical conjunction. But the resulting failure mode is then also the combination (disjunction) of the two original failure modes. This allows us to find a complete list of failure-modes, which can then be used for formal safety analysis.

It is interesting, that behavioral inclusion ($FS \leq SC$) is enough to ensure a complete list of failure modes. The reason is, that if the failure-sensitive model can show less behavior, then it is has more or stricter specification rules than necessary. This means, the model has more or more general failure modes than necessary. A safety analysis done with this list of failure modes is already safe.

3 Formal FTA

A well-known safety analysis technique is fault tree analysis (FTA, [23]). FTA was developed for technical systems to analyze if they permit a hazard (top event). The top event is noted at the root of the fault tree. Events which cause the hazard are given in the child nodes and analyzed recursively, resulting in a tree of events. Each analyzed event (main event) is connected to its causes (sub-events) by a gate in the fault tree (see Fig. 2). An AND-gate indicates that all sub-events are necessary to trigger the main event, for an OR-gate only one sub-event is necessary. An INHIBIT-gate states that in addition to the cause stated in the sub-event the condition (noted in the oval) has to be true to trigger the main event. The inhibit gate is more or less an AND-gate, where the condition does not have to be a fault. The leaves of the tree are the failure modes (basic events) for the top event, which have to occur in combination (corresponding to the gates in the tree) to trigger the top event. A combination of basic events

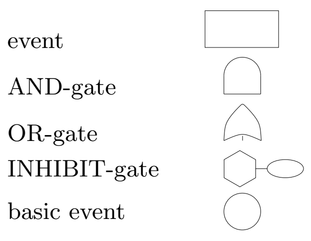


Fig. 2. Fault Tree Symbols

which leads to the hazard is called a *cut set*. A *minimal cut set* is a cut set which can not lead to the top level hazard, if only one event of the set is prevented. This information helps to identify failure events whose exclusion secures the system. If for example one event occurs in different minimal cut sets, the probability of the top level hazard will strongly decrease, if this event can be excluded.

Minimal cut sets can be computed from fault trees by combining the primary events with boolean operators as indicated by the gates. A minimal cut set then consists of the elements of one conjunction in the disjunctive normal form of the resulting formula.

Given a formal system model, we defined a formal semantics of this informal technique. Each gate is represented by a interval temporal logic (ITL) formula. Temporal formulas in ITL are built from first-order formulas using propositional connectives and the following temporal operators[1]: $\boxed{\scriptstyle i}\ \varphi$ ("in all initial intervals φ"), $\boxed{\scriptstyle x}\ \varphi$ ("in all subintervals φ"), $\Diamond_{i}\ \varphi$ ("in some initial interval φ"), $\Diamond_{x}\ \varphi$ ("in some subinterval φ"), and $\varphi\ \mathbf{;}\ \psi$ (read φ chop ψ: "the interval can be split, such that φ holds in the first part and ψ in the second").

[1] ITL also defines quantification and many other derived operators not needed here. More information may be found in [2][6]

The formalization of FTA showed, that defining the semantics of an OR-gate simply as a disjunction is insufficient, since it does not take into account that the sub-events (causes) usually happen *before* the main event (consequence), and that events may have duration. Therefore, we distinguish between decomposition gates (D-gates) and cause-consequence gates (C-gates). We get 7 types of gates. In figure 3 the FTA formulae for D-gates (left column) and C-gates (right column) are shown. D-OR- and D-AND-gates (D, D) can be defined canonically:

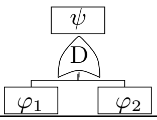
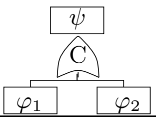
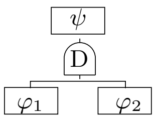
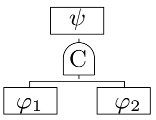
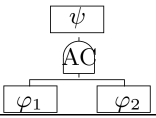
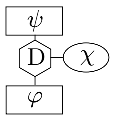
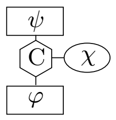
Gate	ITL-formula	Gate	ITL-formula
	$\boxtimes\ (\psi \rightarrow \varphi_1 \vee \varphi_2)$		$\neg\ (\neg\ \Diamond_{\mathrm{x}}\ (\varphi_1 \vee \varphi_2)\ ;\ \Diamond_{\mathrm{i}}\ \psi)$
	$\boxtimes\ (\psi \rightarrow \varphi_1 \wedge \varphi_2)$		$\neg\ (\neg\ \Diamond_{\mathrm{x}}\ (\varphi_1 \wedge \varphi_2)\ ;\ \Diamond_{\mathrm{i}}\ \psi)$
			$\neg\ (\neg\ \Diamond_{\mathrm{x}}\ \varphi_1\ ;\ \Diamond_{\mathrm{i}}\ \psi)$ $\wedge\ \neg\ (\neg\ \Diamond_{\mathrm{x}}\ \varphi_2\ ;\ \Diamond_{\mathrm{i}}\ \psi)$
	$\boxtimes\ (\psi \rightarrow \varphi \wedge \chi)$		$\neg\ (\neg\ \Diamond_{\mathrm{x}}\ (\varphi)\ ;\ \Diamond_{\mathrm{i}}\ \psi)$ $\wedge\ \neg\ (\neg\ \Diamond_{\mathrm{x}}\ (\chi)\ ;\ \Diamond_{\mathrm{i}}\ \psi)$

Fig. 3. Formal semantics of fault trees

for example the D-AND-gate (D) states, that whenever the effect ψ happens, both causes φ_1 and φ_2 must happen as well. A C-OR-gates (C) states, that it must not be possible to split a run, such that none of the causes φ_1 and φ_2 ever happens in the first half, but the consequence ψ happens at the beginning of the second half. In other words: if the consequence happens, one of the causes must have happened before (*completely*, if it has duration, therefore the chop is necessary). Causes and consequences must not overlap. The asynchronous and synchronous C-AND-gates (C, AC) are similar, they require that both causes must have happened (at the same time) before the consequence. The conditions for D-INHIBIT- and C-INHIBIT-gates (D, C) are the same as for the D-AND-gate and AC-gate.

Hansen et al. [10] defines cause/consequence gates in Duration Calculus (DC, [24]), but their definition does not meet the requirement, that causes are completed before the consequence. A subsequent publication [9] is restricted to decomposition gates. Bruns and Andersen [5] also define a fault tree semantics using μ-calculus. They also distinguish between cause/consequence and decomposition gates. Only events without duration are considered. For this special case, our semantics is equivalent (see [22] for details).

For the semantics in Fig. 3, the following theorem was proven:

Theorem 1 (completeness theorem (minimal cut set theorem)). *If all conditions of a fault tree can be verified, and if for each minimal cut set at least one of its basic events is prevented from happening on each run, then the top-level event will never happen.*

For a complete fault tree it is sufficient to prevent one primary event of each minimal cut set, to avoid the fault under consideration altogether. A complete fault tree is therefore a partial proof for the safety of the system. The completeness theorem gives a formal justification for the use of minimal cut sets in safety analysis, even for cases where timing conditions are relevant [18].

The theorem is proved using structural induction over the size of the fault tree. The basic fact underlying the proof is transitivity of the cause-consequence relation. The proof was formally verified in KIV ([3]), using an algebraic specification of the syntax and semantics of continuous Interval Temporal Logic.

4 Parameterized, Quantitative FTA

In reality it is not only interesting which reason might cause system failure, but also how probable this failure is. In safety analysis, there exists a variety of techniques for assessing failure probabilities. One of them is quantitative FTA.

The idea of quantitative FTA is to take the fault tree and calculate the minimal cut sets. To approximated the risk, probabilities are assigned to each primary failure PF. Assuming statistical independence[2], the probability of a minimal cut set MCS can then be approximated by:

$$P(MCS) := \prod_{PF \in MCS} P(PF)$$

If second and higher order terms are neglected the probability of the hazard H may be approximated by the sum of all minimal cut sets of the fault tree:

$$P(H) := \sum_{MCS \in MCSS_H} P(MCS)$$

MCSS denotes the set of all minimal cut sets of the fault tree for hazard H. Given a fault tree and the probabilities of all primary failures this allows to calculate the hazard probability. This technique is widely used in practice and more details may be found in [23].

[2] If statistical dependence is to be examined, then FTA is often not the method of choice, because calculating dependent probabilities can be very hard. In this case probabilistic model checking ([1]) can help, here are the failure probabilities already included into the system model, which is in general by far easier.

The drawback is, that this is a static analysis, which has to be redone every time some parameter changes. This is unnecessary: if a system has free parameters like safety gaps, timeouts or maintenance intervals, then it is much more convenient to replace the static probabilities by probability distributions. Many random variables may be approximated by well-known standard distributions. For safety analysis the most important ones are exponential and normal distributions. More information on probability distribution may be found in [19].

To obtain a parameterized probability for the hazard, we use the same approach as standard quantified FTA but replace the static probabilities of the primary failures with parameterized probability distributions. So P(H) no longer is a fixed value but rather a function of the free parameters $X_1, \ldots, X_n$. Now it is very easy to examine the system in dependence of the free parameters.

$$P(H)(X_1, .., X_n) := \sum_{MCS \in MCSS_H} \prod_{PF \in MCS} P(PF)(X_1, .., X_n)$$

However, there are still possibilities to improve this approach for quantitative approximation. Often the events of a cut set lead to the hazard only, if some constraints (which are no faults) are satisfied, e.g. only if the system is in a certain operation mode. Constraints are typically derived from INHIBIT-gates, since the side-condition χ of an INHIBIT-gate is a a formal description of a necessary constraint. Constraint probabilities are multiplied with a cut sets probability. So the final formula to calculate a hazards probability is:

$$P(H)(X_1, .., X_n) := \sum_{MCS \in MCSS_H} P(MCS) \prod_{PF \in MCS} P(PF)(X_1, .., X_n)$$

Constraint probabilities result in a much more precise quantitative approximations for the hazards occurrence.

5 Safety Optimization

Safety optimization is the next step towards formal safety analysis. It deals with the problem of choosing free parameters. Many industrial projects start in a testing phase to find out the optimal working parameters. For safety analysis this is not a feasible approach. As good system design should make the hazards very improbable i.e. only one hazard in several years, it would take an extraordinary long testing time to verify which parameters are the best choice for safe operations.

The problem becomes more interesting and complicated if two or more competing hazards are investigated. Consider e.g. the pre-flight safety check of an airplane. The most critical hazard is, that an unsafe airplane passes the test. A competing hazard is, that a technically okay airplane fails the test. It is intuitively clear, that it is not possible to minimize both hazards at the same time. A fair balance must be found.

To find it, we set up a mathematical optimization problem, which combines the results of (formal) FTA with parameterized probabilities. A cost function is defined, which measures risk in estimated costs (costs are also used into calculations whether operation of the system is profitable). For many domains a good cost function is the weighted sum of all hazards probabilities:

$$f_{cost}(P(H_1), .., P(H_m)) := \Sigma_{i=1}^{m} Cost_{H_i} P(H_i)$$

The weight for each hazard is the costs it usually causes. This type of cost function describes the mean expected costs that all hazards together will cause. The hazards probabilities $P(H_i)$ are substituted with the parameterized probabilities of the previous section.

$$f_{cost}(X_1, .., X_n) := \Sigma_{i=1}^{m} Cost_{H_i} \sum_{MCS \in MCSS_{H_i}} P(MCS) \prod_{PF \in MCS} P(PF)(X_1, .., X_n)$$

The cost function now no longer depends on the hazards probabilities, but on the probabilities of the primary failures and constraints. More specifically, it depends on the choice of the free parameters of the system. In other words: Minimization of the cost function will result in an optimal set of free parameters. The minimization may be solved with any mathematical tool or optimization algorithm ([8], [14]). Newtons Methods is often a good choice, as usually the function is twice continuously differentiable and an appropriate initial guess easy to find.

6 Methodology

The previous section outlined formal safety analysis techniques. In this chapter we will describe how these techniques may be combined in an efficient way. Figure 4 shows the interaction of our techniques as a graph. Arrows represent dependencies, and the time line is from left to right.

The top trace - informal specification, formal functional model, functional correctness - represents the standard approach of formal software development in computer science. The bottom trace - informal specification, traditional FTA, quantitative FTA - is the methodology for traditional safety analysis with FTA.

The ForMoSA techniques are in between these two boundaries. They combine the tools of formal verification techniques with the matters of safety analysis. This yields a mutual benefit: the results of safety analysis techniques are now built on a solid formal foundation, and the formal model is validated by checking the proof obligations of FTA.

Typically, the connection between the different techniques is an interdependence. Method A yields new results for method B, which in turn results in a feedback to method A. An example for this process may be found in [16]. There the methodology for tightly versus loosely couple FTA is presented. We can not present all the details here, but try to describe the whole approach for formal safety analysis.

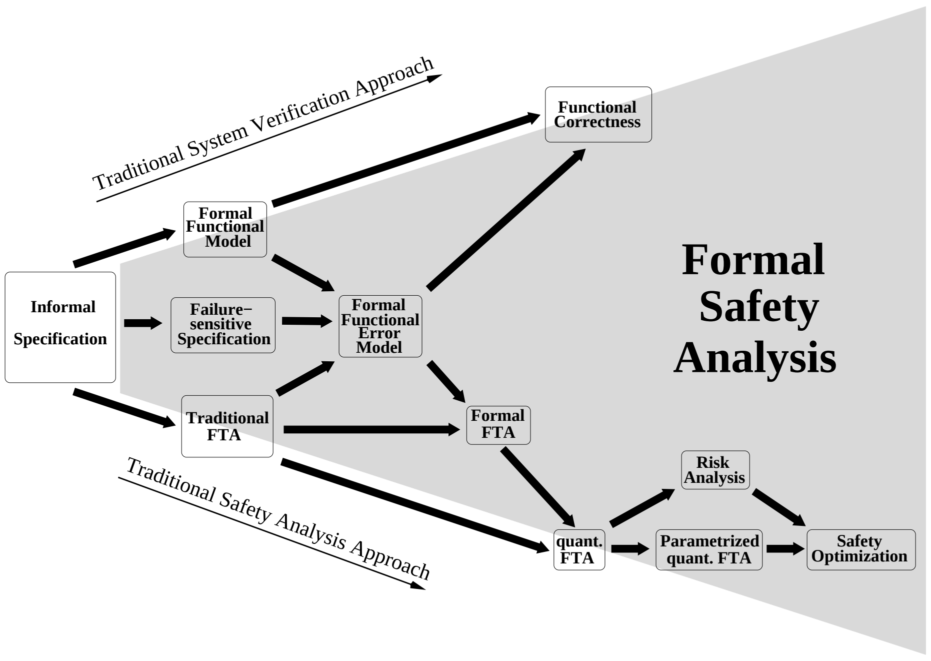


Fig. 4. Interaction of techniques

6.1 The ForMoSA Approach

The ForMoSA approach may be split into three different sections. The first deals with building a formal specification, which also models possible failures. We call this the functional error model. The second part deals with qualitative analysis. The goal in this part is to find the causal dependencies between component failures and system failure. The last step deals with quantitative approaches. Namely, the approximation of hazard probabilities and the optimization of the system. The parts are not completely disjoint. But it is also possible to skip various work packages within one block, if they seem not adequate or - more probable - if time is too short to do the full analysis process.

6.2 Building the Formal Model

This block contains the methods building the functional model, failure-sensitive specification, traditional FTA and building the functional error model. We suggest to stick to the methodology sketched in Fig. 5.

As a first step, we set up an informal (requirements) specification of the system to build, and describe the potential hazards informally. The description fixes import domain specific notions.

Based on the informal description a formal model is specified, that describes the intended functional behavior, called the functional model. This model usually does not describe any component failures. We typically use statecharts (or more restricted forms of automata) to describe it.

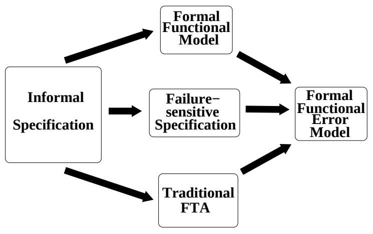

Formal Functional Model
Informal Specification
Failure–sensitive Specification
Formal Functional Error Model
Traditional FTA

Fig. 5. Part 1

1. Build a functional system model.
2. Do an informal FTA of the system.
3. Do failure-sensitive specification for all critical components.
4. Integrate the relevant failure modes into the functional model.
5. Check for correct integration of the failure modes into functional model.

The functional model is of course not sufficient for formal safety analysis, since it will not contain component failures. To derive critical components and their failure modes, we do an informal FTA for the hazards of the system, based on the informal specification. We suggest to do FTA *independent* of the formal model. This has two advantages: first, an informal FTA can be done by an expert of the domain, without knowledge of formal specifications. Second, it avoids the danger that FTA finds hazards only, that are already suggested by the formalization. This methodology is different from the one proposed in [4], which suggests to generate fault trees from formal models.

If an informal FTA finds, that there are components with a large number of failure modes, we set up failure-sensitive specifications for these components. If these are proven to be equivalent to the functional model of the component, then a complete list of failure modes has been obtained. This comprehensive list gives a good idea of what can go wrong in each component.

Finally, we combine the results of all three techniques into the functional error model that adds component failures to the functional model. Not all component failures that were found by failure-sensitive specification have to be added, but at least those which appear as leaves in the fault trees are necessary. For others, a domain expert has to check whether they are relevant in practice.

Integration of failure is done in two steps. First for each failure mode, an additional chart describing the occurrence of the failure is added. This chart describes when and how the failure mode takes place. In the second step the effects of the failure mode have to be integrated. In many cases this may be done by changing the guard condition of a transition. However, sometimes additional states or even changes in the design may be necessary.

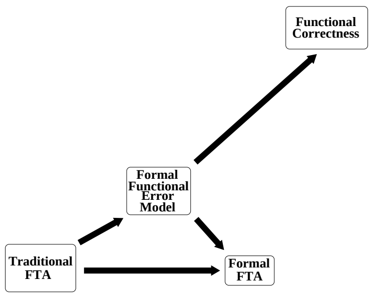
In the last step the correct integration of the failure modes into the functional model can be verified. This is done by proving behavioral equivalence of the new functional error model and a failure-sensitive specification model, where the integrated failure modes have been dropped, see Sect. 2.

The functional error model, that is the final result of this work, does not only model the intended functionality but also possible component failures and errors.

6.3 Qualitative Analysis

This part integrates traditional FTA and formal methods as shown in Fig. 6. Both methods can be used to analyze possible causes for hazards. While the traditional verification task tries to show the absence of the hazard (if all components of the system working properly) and tries to find design flaws, traditional FTA accepts the hazard, but tries to find out the reasons. Formal verification is based on a solid formal foundation. Traditional FTA is not formal at all and its results largely depend on the skill of the engineer.

Formal FTA fills the gap between those to bounds. It has both a solid formal semantics ([18]) and allows to reason about failures as causes for hazards. Formal FTA is a significant improvement over verification or FTA, but it requires the effort to build a good functional error model, which is used as its foundation.



1. Verify functional correctness of the functional error model.
2. Do traditional FTA (if not already done before).
3. Do formal FTA.

Fig. 6. Part 2

The starting point for formal FTA is the fault tree of traditional FTA. As a first step, each event of the fault tree has to be defined as a formula over the system model. Events, for which this is difficult or impossible, usually indicate, that the functional error model is still not adequate and has to be changed. Such cases should be rare, since the informal fault tree has already been taken into account when the failure modes were chosen and integrated.

Formal FTA results in a set of temporal logics formulae which have to be proven correct for the functional error model. If all proof obligations are proven valid, then the fault tree is complete. This means no branches have been forgotten. More specifically it has been proven, that for a complete fault tree the prevention of one element of every minimal cut set results in the prevention of the hazard. This is the formal equivalent to the intuitive understanding of a fault tree in engineering.

In addition to formal FTA, formal verification may also be applied to ensure functional properties of the system. Typically, functional properties are proved under the condition that all components work properly.

Note that another use of formal methods — to prove the absence of hazards provided all components work properly — becomes conceptually obsolete using formal FTA. That hazards are not possible because of design flaws is part of the result of formal FTA. Nevertheless we suggest to verify the absence of design flaws for the functional model before doing a safety analysis. First, proofs over the (smaller) functional model are often easier than proofs over the (more complex) functional error model. Second, they can be done before a (costly) safety analysis is attempted, that may turn out to be obsolete since the design was (irreparably) flawed.

The main result of this part is a precise description of the qualitative relationship between component failures and hazards. This relationship is essentially captured by the minimal cut sets.

6.4 Quantitative Approaches

The third part of our approach deals with quantitative analysis. It includes quantitative FTA, parameterized FTA and safety optimization. The idea is to use the results of part two for calculating probabilities. In a second step the free parameters are taken into account.

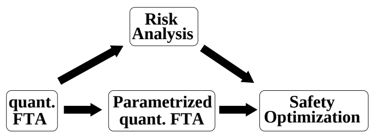


Fig. 7. Part 3

1. Do quantitative FTA.
2. Determine free parameters.
3. Do (quantitative) parameterized FTA.
4. Do a risk analysis to determine concurring hazards costs.
5. Do mathematical optimization.

We start by doing a quantitative FTA. This helps in two aspects. First, it gives a rough upper bound estimation of the hazards probabilities. This allows to decide at an early stage, whether the system has a high probability of working correctly or not. Second, quantitative FTA requires to approximate all primary failures probabilities. In this process it often becomes clear, that some of these approximations depend on parameters. These probabilities are good candidates for parameterized probabilities.

The next step is to determine the free parameters of the system (e.g. the runtime of a timer), specific to the system at hand, and to parameterize FTA.

For every primary failure it must be examined, in which way it depends on the free parameters. The probabilities are replaced by probability distributions. The correct choice of a matching distribution is a problem of statistics. For many problems - in particular for failure rates - exponential distribution is a good choice. If it is not clear, which distribution should be used, then it is necessary to do a standard statistical analysis (see e.g. [19]).

To make the hazards comparable, risk analysis ([7]) is the method of choice. For the final step of the analysis it is necessary to have a function describing the importance of the different hazards. A good choice is to use a cost function. This function is the sum of all hazards mean costs. The mean cost of a hazard are given by the product of the hazards probability and its costs. However, our approach does not depend on the exact type of function. The most adequate cost function may only be determined by domains experts.

The last step is optimization. In this step an optimal configuration for the free parameters may be found. If the probability distribution are twice continuously differentiable and the cost function as well, then Newtons Method is a good choice as optimization algorithm. For other problems there exists a variety of different optimization algorithm, but in practice most problems may be modeled with the standard set of probability distributions for which Newton is applicable.

The result of this step is an optimal configuration of the system with respect to the input data. Inputs are the used distributions and the hazards relationship of importance.

7 Applications

In this section we present an application of our approach on a real world case study. The examined problem is the new Elbtunnel in Hamburg. This is a road tunnel beneath the river Elbe in Hamburg. It connects the city with the harbor. Up to now the tunnel consists of 3 tubes with 2 lanes each. Now a new fourth tube has been built. The new tube is higher to allow over-high - more than 4 meters height - vehicles (OHVs) to pass. This requires a new height control system necessary, which significantly extends the old one.

The old system consisted of light barriers across all lanes, that triggered an emergency stop for vehicles higher than 4 meters,

For the new tunnel, the existing height control had to be enhanced, such that it allows over-high vehicles to drive through the new, higher tube, but not through the old ones.

In the following, we will distinguish between *high vehicles* (HVs), which may drive through all tubes and *over-high vehicles* (OHVs), which can only drive through the new, fourth tube. Figure 8 sketches the layout of the tunnel. Only the northern entrance is shown, as we will only discuss the crossing from north to south.

The system uses two different types of sensors. Light barriers (LB) are scanning all lanes of one direction to detect, if an OHV passes. For technical reasons they cannot be installed in such a way, that they only supervise one lane. Therefore overhead detectors (OD) are necessary to detect, on which lane a HV passes. The ODs can distinguish vehicles (e.g. cars) from high vehicles (e.g. buses, trucks), but not HVs from OHVs (but light barriers can!). If the height

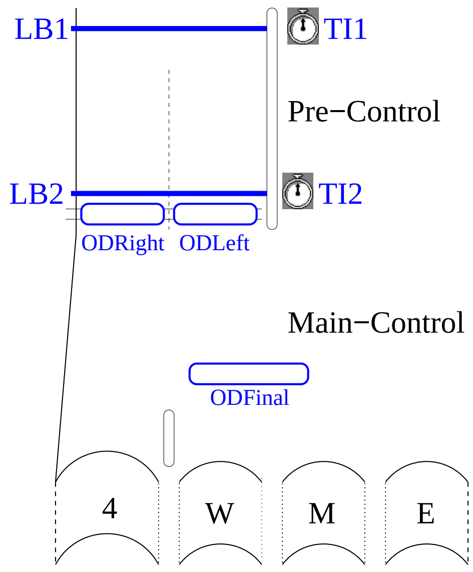


Fig. 8. Layout of the northern tunnel entrance

control detects an OHV heading towards a different than the fourth tube, then an emergency stop is signaled, locking the tunnel entrance.

The idea of the height control is, that the detection starts, if an OHV drives through the light barrier $\mathrm{LB}_{\mathrm{pre}}$. To prevent unnecessary alarms through faulty triggering of $\mathrm{LB}_{\mathrm{pre}}$, the detection will be switched off after expiration of a timer (30 minutes). Road traffic regulations require, that after $\mathrm{LB}_{\mathrm{pre}}$ both HVs and OHVs have to drive on the right lane through tunnel 4. If nevertheless an OHV drives on the left lane towards the west-tube, detected trough the combination of $\mathrm{LB}_{\mathrm{post}}$ and $\mathrm{OD}_{\mathrm{left}}$, an emergency stop is triggered. If the OHV drives on the right lane through $\mathrm{LB}_{\mathrm{post}}$, it is still possible for the driver to switch to the left lanes and drive to the west- or mid-tube. To detect this situation, the height control uses the $\mathrm{OD}_{\mathrm{final}}$ detector. To minimize undesired alarms (remember, that normal HVs may also trigger the ODs), a second timer will switch off detection at $\mathrm{OD}_{\mathrm{final}}$ after 30 minutes. For safe operation it is necessary, that after the location of $\mathrm{OD}_{\mathrm{final}}$ it is impossible to switch lanes. Infrequently, more than one OHV drive on the route. Therefore the height control keeps track of several but at the most three OHVs.

In the following we will present the safety analysis of this system according to the presented approach. It is not possible to show the whole analysis in this paper. We will rather take some parts of the analysis as explanation of all steps. The example can be modeled as a finite state system, and model checking can be applied to verify properties. The ILL formalizations of failure-sensitive specification and formal FTA may be transfered into an equivalent CTL semantics, if the basic events have no duration (see [21]). The verification has been done with the SMV-tool.

7.1 Building the Formal Model

The formal model of the Elbtunnel consist of 10 parallel automata (three for modeling OHVs, three for modeling HVs, two timers, the pre-control and the post-control). Figure 10 shows the automaton for an OHV.

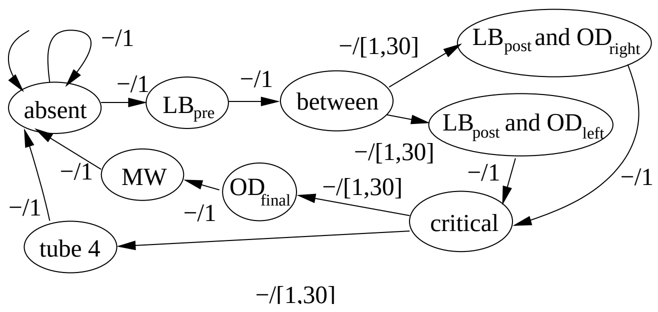


Fig. 9. Automaton modeling an OHV

For the sake of readability we extend the normal notation of finite automata with timing conditions $[t_1, t_2]$. This means the transition may be taken indeterministically between time t_1 and t_2. The translation into a standard automaton is fairly easy, by introducing intermediate states.

To obtain the failure modes we did a failure-sensitive specification. Note, that in this case component failure not necessarily means failure of a component but can also mean misbehavior of the driver. The failure-sensitive specification for this component consists of 2^{21} different input-output scenarios. It yields 14 different failure modes. Some of them can be ruled out by simple comparison with reality. For example it is not possible, that an OHV suddenly starts existing in one section without traveling through the others. Another failure mode "vehicle does not stop despite stop signals" has also been ignored, as system failure is obvious. But many other failure modes are important like "vehicle needs longer than runtime of timer 1 for section 1" or "vehicle passes through OD_{Left} AND OD_{Right}".

Altogether failure-sensitive specification resulted in 30 proof obligations that were proven with SMV. In practice this helped us a lot find specification errors. When proofs failed we discovered in some cases, that our formalization of the failure mode was not precise enough, in others we found that our system model was not consistent with our intuitive understanding.

7.2 Qualitative Analysis

The central safety property of the Elbtunnel is, that given no failures is it always true, that whenever an OHV tries to enter another but the fourth tube, an alarm has been signaled. In CTL this formula has the following form:

$$\mathrm{AG}((\mathrm{OHV}_1 = \mathrm{MW} \vee \mathrm{OHV}_2 = \mathrm{MW} \vee \mathrm{OHV}_3 = \mathrm{MW}) \rightarrow \mathrm{CO}_{\mathrm{post}} = \mathrm{stop})$$

This theorem[3] does not hold for our system. Why? The reason lies in the design of the control system. If two OHVs pass LB1 simultaneously, then the system counts them only as one (the light barrier is only interrupted once). If the two OHVs drive at very different velocities, then it is possible that the final overhead detector has already been deactivated before the second OHV reaches it. It is very improbable, that this scenario would have been discovered by simple testing of the system. This problem can be solved by installing additional overhead detectors at LB1 or by changing the control system, such that main-control is always active for the whole runtime of timer 1.

Traditional FTA for collision yields a fault tree, which has leaves only, where a sensor failed in such a way, that they missed the detection of an vehicle (misdetection). It seemed not appropriate, that it would have an impact on safety, if the sensors detected to often (false detection). However, it turned out, that a false detection of LB_{post} is even a primary failure. This means if everything works correctly, but the second light barrier has a false detection (e.g. a passing bird), then a collision is possible. The scenario is easy to construct. An OHV passes LB_{pre}, immediately after this a bird triggers LB_{post}, when the OHV reaches LB_{post} it is deactivated, and when the OHV finally reaches OD_{final} this sensor has been deactivated again. The corresponding branch would have been forgotten with traditional FTA.

7.3 Quantitative Analysis

For quantitative FTA there are a lot of probability approximations for failures and traffic conditions necessary. We will not go into detail, but only give the results. The overall probability for the hazard evaluates to 3^{-8} per minute or one hazard every 64 years. The false alarm rate is around 3^{-4} or one false alarm every two days. This is a very good result for the system and gives confidence.

We will now parameterize the probabilities. Obviously free parameters of the system are the runtimes of the timers. The probability for an OHV getting into a traffic jam for more than the runtime of a timer of course depends on this parameter. We call this failure mode overtime $OT_{1/2}$.A good model for the driving time of OHVs from LB_{pre} to LB_{post} and from LB_{post} to the tunnel is normal distribution [4]. $P_{OHV_{1/2}}(Time \leq T)$ denotes the probability for a driving

[3] The CTL-operator AG means "on all paths it is always true, that ...".

[4] truncated at zero; mean time $\mu = 4$ minutes; standard deviation $\sigma = 2$ minutes

time $\leq$ T. We can then calculate P(OT_1) in dependence of the runtime T1 of timer 1:

$$P(OT_1)(T1) = 1 - P_{OHV_1}(Time \leq T1)$$
$$where\ P_{OHV_1}(Time \leq T) := \frac{1}{\int_0^\infty exp(-\frac{(x-\mu)^2}{2\sigma^2})dx} \int_0^T exp(-\frac{(x-\mu)^2}{2\sigma^2})dx$$

Timer 2 is treated analogously. The hazards probability is now a function of the runtimes of the timers. The runtimes of the timers have not only an influence on probabilities in the fault tree of the collision hazard, but also on the fault tree for false alarms. So both hazards probabilities are now formulated in terms of timer runtimes. The last thing we now need is a cost function. A rough approximation for the costs is the a collision costs 100000 times the money of a false alarm The following figure shows this cost function:

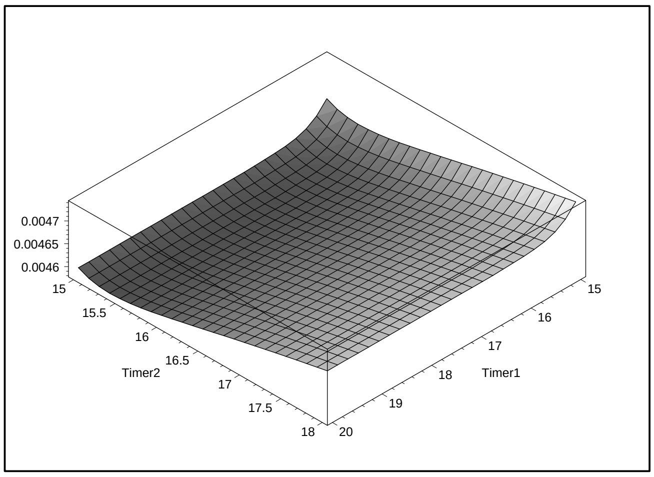


Fig. 10. The cost function around its minimum

It turns out, that the optimal configuration is 19 minutes for time 1 and 15.6 minutes for timer 2. This is a big improvement compared to the initial values proposed - 30 minutes for both timers. However, the resulting failure probabilities do not improve as much for collision it stays at 3^{-8} and the false alarm rate drops by 10% to 2.7^{-4}. This is due to another design problem, which can also be found with parameterized probabilities. If the frequency of correctly driving OHVsis introduced as free parameter, then it turns out that over 80% of these vehicles trigger a false alarm. The reason lies in detector OD_{Final}. This detector always stays activated for the runtime of timer 2 (and whenever an OHVs passes the tunnel, it will be activated). During the runtime of timer 2 it is almost sure (more than 80%), that a bus, minivan or normal truck will pass the sensor, but every detection at the sensor immediately triggers an alarm. If one takes into account, that this means the frequency of false alarm is roughly proportional to the frequency of OHV, then this could be called a design flaw. The problem can be solved easily by introducing a third light barrier at OD_{Final}. This reduces the false alarm rate by more than one order of magnitude and removes the dependence of the false alarm rate on the OHV rate.

8 Conclusion

Our experience is that it is important to combine different techniques for safety analysis. This is because different methods not only examine different aspects of the system, but also give contrary views [11]. Therefore an integrated approach is needed. The growing complexity of embedded systems makes a formal process necessary. Formal reasoning helps to better understand how unwanted and unintended events are propagated through the system and which effects they have.

The ForMoSA approach is an integrated formal methodology. It includes different techniques to cover different aspects on safety as well as a solid formal foundation for qualitative reasoning. Together this addresses many common problems of safety critical systems and assures a very high level of safety.

We illustrated the different techniques of our approach on a real world case study - the Elbtunnel in Hamburg. This case study shows, that no single technique can give all results the combined approach can give.

References

[1] Christel Baier, Edmund M. Clarke, Vassili Hartonas-Garmhausen, Marta Z. Kwiatkowska, and Mark Ryan. Symbolic model checking for probabilistic processes. In *Automata, Languages and Programming*, pages 430–440, 1997.

[2] M. Balser. *Verifying Concurrent System with Symbolic Execution – Temporal Reasoning is Symbolic Execution with a Little Induction.* PhD thesis, University of Augsburg, Augsburg, Gemany, 2004. (to appear).

[3] M. Balser, W. Reif, G. Schellhorn, K. Stenzel, and A. Thums. Formal system development with KIV. In T. Maibaum, editor, *Fundamental Approaches to Software Engineering*, number 1783 in LNCS, pages 363–366. Springer-Verlag, 2000.

[4] M. Bozzano, A. Cavallo, M. Cifaldi, L. Valacca, and A. Villafiorit. Improving safety assessment of complex systems: An industrial case study. In K. Araki, S. Gnesi, and D. Mandrioli, editors, *FM 2003: Formal Methods*, number 2805 in LNCS, pages 208–222. Springer-Verlag, 2003.

[5] G. Bruns and S. Anderson. Validating safety models with fault trees. In J. Górski, editor, *SafeComp'93: 12th International Conference on Computer Safety, Reliability, and Security*, pages 21 – 30. Springer-Verlag, 1993.

[6] A. Cau, B. Moszkowski, and H. Zedan. *ITL – Interval Temporal Logic.* Software Technology Research Laboratory, SERCentre, De Montfort University, The Gateway, Leicester LE1 9BH, UK, 2002. www.cms.dmu.ac.uk/~cau/itlhomepage.

[7] ECSS. Dependability. In European Cooperation for Space Standardization, editor, *Space Product Assurance*. ESA Publications, 2001.

[8] A. H. G. Rinnooy Kan G. L. Nemhauser, editor. *Optimization*, volume Vol 1. Elsevier Science Publishers B.V, 1989.

[9] K. Hansen, A. Ravn, and V. Stavridou. From safety analysis to software requirements. *IEEE Transactions on Software Engineering*, 24(7):573 – 584, July 1998.

[10] K. M. Hansen, A. P. Ravn, and V. Stavridou. From safety analysis to formal specification. ProCoS II document [ID/DTH KMH 1/1], Technical University of Denmark, 1994.

[11] E.G. van den Blieck J.L. Rouvroye. Comparing safety analysis techniques. *Reliability Engineering & System Safety*, 2002.

[12] J. Klose and A. Thums. The STATEMATE reference model of the reference case study 'Verkehrsleittechnik'. Technical Report 2002-01, Universität Augsburg, 2002.

[13] N. Leveson. *Safeware: System Safety and Computers*. Addison Wesley, 1995.

[14] David G. Luenberger. *Linear and nonlinear programming*. Addison-Wesley Publishing Company, 1989.

[15] F. Ortmeier and W. Reif. Failure-sensitive specification: A formal method for finding failure modes. Technical Report 3, Institut fr Informatik, Universitt Augsburg, 2004.

[16] F. Ortmeier, W. Reif, G. Schellhorn, A. Thums, B. Hering, and H. Trappschuh. Safety analysis of the height control system for the Elbtunnel. *Reliability Engineering and System Safety*, 81(3):259–268, 2003.

[17] F. Ortmeier and A. Thums. Formale Methoden und Sicherheitsanalyse. Technical Report 15, Universitt Augsburg, 2002. (in German).

[18] G. Schellhorn, A. Thums, and W. Reif. Formal fault tree semantics. In *Proceedings of The Sixth World Conference on Integrated Design & Process Technology*, Pasadena, CA, 2002.

[19] Klaus Schürger. *Wahrscheinlichkeitstheorie*. R. Oldenbourg Verlag, 1998.

[20] N. Storey. *Safety-Critical Computer Systems*. Addison-Wesley, 1996.

[21] A. Thums and G. Schellhorn. Model checking FTA. In K. Araki, S. Gnesi, and D. Mandrioli, editors, *FME 2003: Formal Methods*, LNCS 2805, pages 739–757. Springer-Verlag, 2003.

[22] A. Thums, G. Schellhorn, and W. Reif. Comparing fault tree semantics. In D. Haneberg, G. Schellhorn, and W. Reif, editors, *FM-TOOLS 2002*, Technical Report 2002-11, pages 25 – 32. Universität Augsburg, 2002.

[23] W. E. Vesely, F. F. Goldberg, N. H. Roberts, and D. F. Haasl. *Fault Tree Handbook*. Washington, D.C., 1981. NUREG-0492.

[24] Zhou Chaochen and M. R. Hansen. Duration calculus: Logical foundations. In *Formal Aspects of Computing*, pages 283–330, 1997.