

Formal Safety Analysis in Transportation Control

A. Thums, G. Schellhorn

Universität Augsburg

Lehrstuhl für Softwaretechnik und Programmiersprachen

Universitätsstr. 14, D-86135 Augsburg

Tel.: +49 (0)821 598 2176

Fax: +49 (0)821 598 2175

E-Mail: {thums,schellhorn}@informatik.uni-augsburg.de

Abstract: Transportation control systems are safety critical systems. While a couple of years ago control systems mainly used to be built up from (electro-) mechanical devices, nowadays more and more functionality is software controlled. To sustain the high level safety standards for these embedded systems, we propose to use fault tree analysis integrated with formal methods for analyzing system safety. This approach combines typical safety analysis techniques from engineering resp. software engineering. Fault tree analysis mainly focuses on system safety and considers defective components whereas formal methods mainly focuses on functional correctness.

This paper presents the methodical aspects of the combination. To benefit from both, it is important to retain the different points of view of fault tree analysis and formal methods and nevertheless end up with an integrated formal model, where safety properties can formally be proven.

Key words: safety analysis, fault tree analysis, formal methods, model checking, methodology

1 Introduction

In engineering, a number of very useful techniques have been developed to analyze safety-critical systems. Examples are fault tree analysis (FTA) [VGRH81], failure mode and effects analysis (FMEA) [Rei79], or hazard and operability analysis (HAZOP) [FMNP95]. These techniques assist in the detection of design errors, safety flaws, and weaknesses of technical systems. Classical safety analysis mainly operates on informal grounds. It is based on an informal description of the underlying system. Therefore it is quite hard to mechanize the safety analysis or to check the description for adequacy or consistency. This is an issue where formal specification techniques might help. These use formally specified system models to proof consistency and other properties of the model. We will use formal models as a basis for verifying the safety properties derived from the fault tree analysis.

Originally, safety analysis was applied on hardware components and formal methods mainly on software. For software-based systems like transportation control systems it is necessary to integrate both. We will use the example of a decentralized radio-based crossing control [JS99] to explain our approach.

This paper shortly introduces our integrated approach but mainly focuses on the methodology. To benefit from the intuitive approach of fault tree analysis, we propose to start with an informal model. Based on this model, we develop a formal model and the fault trees for all important hazards separately, first. Then we take over the safety properties derived with fault tree analysis and proof them within the formal model. This step can be done with two different levels of formalization. Either safety properties derived with FTA are formalized and proven. Or we take a more formal way and formalize, in addition to the safety properties, the events of the fault tree, so that they can be expressed in the formal model. Proof obligations derived from the fault tree are verified, which guarantee that the FTA is correct and complete with respect to the formal model. For this validation of the fault tree, the formal semantics of FTA presented in [STR02] will be used. We will call these two levels of combination loose resp. tight coupled formal FTA.

The paper is organized as follows. The ‘radio-based crossing control’ example will be described in Sect. 2. Sect. 3 sketches the fault tree analysis technique in general and Sect. 4 the formalization of FTA. The paper mainly focuses on the methodology of the application of formal fault tree analysis, presented in Sect. 5. Finally, Sect. 6 concludes.

2 The Radio-Based Crossing Control

The German railway organization, Deutsche Bahn, prepares a novel technique to control level crossings: the so called ‘FunkFahrBetrieb’ [Bet]. This technique aims at medium speed routes, i.e. routes with maximum speed of 160 km/h. The essential parts of this technique is reflected in the case study of the decentralized, radio-based crossing control [JS99]. An overview is given in Fig. 1. The main difference between this technology and the traditional control of level crossings is, that signals and

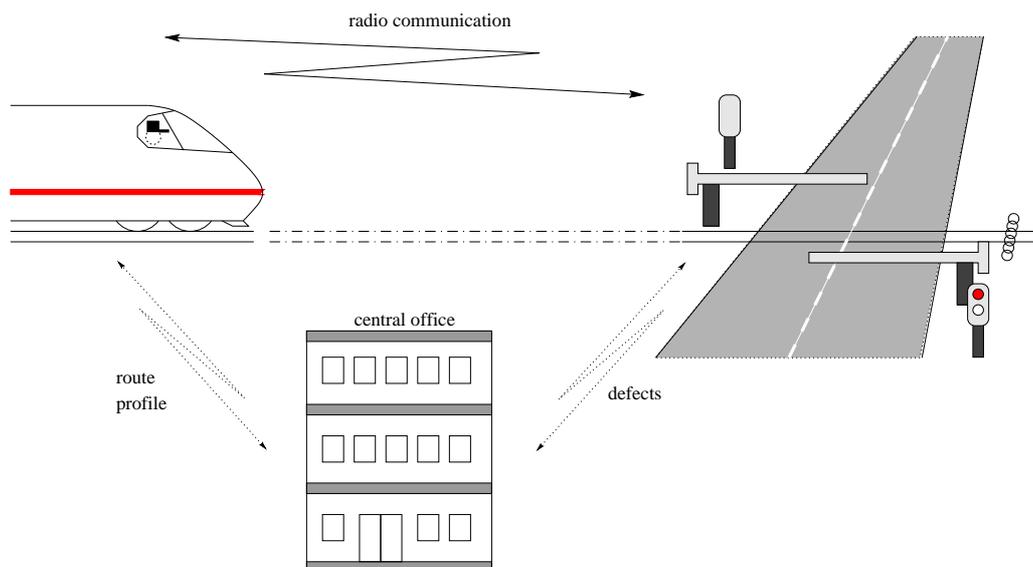


Figure 1: Radio-Based Crossing Control System

sensors on the route are replaced by radio communication and software computations in the train and level crossing. This offers cheaper and more flexible solutions, but also shifts safety critical functionality from hardware to software.

Instead of detecting an approaching train by a sensor, the train computes the position where it has to send a signal to secure the level crossing. Therefore, the train has to know the position of the level crossing, the time needed to secure the level crossing, and its current speed and position. The first two items are memorized in a data store and the last two items are measured by an odometer.

When the level crossing receives the command to 'secure', it switches on the traffic lights, first the 'yellow' light, then the 'red' light, and finally closes the barriers. When they are closed, the level crossing is 'safe' for a certain period of time. The 'stop' signal on the train route indicating an insecure crossing is also substituted by computation and communication. Shortly before the train arrives the 'latest braking point' (latest point, where it is still possible for the train to stop before the crossing), it requests the status of the level crossing. When the crossing is safe, it responds with a 'release' signal which indicates, that the train may pass the crossing. Otherwise the train has to brake and stop before the crossing.

The level crossing periodically performs self-diagnosis and automatically informs the central office about defects and problems. The central office is responsible for repair and provides route descriptions for the train. These descriptions indicate the positions of level crossings and maximum speed on the route.

This system will be used for exemplifying our approach in the following sections. For a formal fault tree analysis the example was modeled with the STATEMATE tool and we will refer to [KT02] for a detailed description of the model and to [RST00] for the safety analysis. STATEMATE supports statecharts for modeling the behavior of components. Statecharts are an expressive form of state transition diagrams and the semantics of STATEMATE statecharts is described by [PS91] and formalized in [DJHP98].

3 Fault Tree Analysis

Fault tree analysis (FTA) is a well known technique in engineering and was developed for technical systems to analyze, if they permit a hazard (top event). This event is noted at the root of the fault tree. Events which cause the hazard (intermediate events) are given in the child nodes and analyzed recursively, resulting in a tree of events. Each analyzed event (main event, either top or intermediate) 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

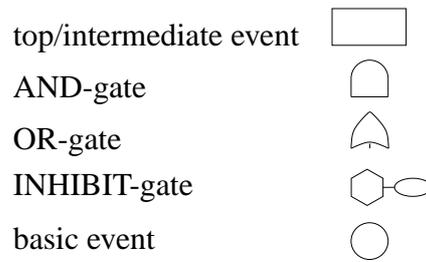


Figure 2: Fault Tree Symbols

AND-gate, where the condition needs not to be a fault.

The leaves of the tree are the low level causes (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 which leads to the hazard is called *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. Minimal cut sets can be computed from fault trees by combining the basic 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.

The cut sets help to identify failure events whose exclusion secures the system. E.g., if one event occurs in different minimal cut sets, the probability of the top level hazard will strongly decrease, if this event can be excluded. Beside this qualitative analysis, FTA also makes quantitative analysis possible. If the failure probabilities of the basic events are known (failure probabilities of basic components can frequently be derived from statistical data) the probability of the occurrence of the hazard can be computed bottom up. For statistically independent failure probabilities, the failure probability for a main event is the the product of the probabilities of its sub-events, if they are connected through an AND-Gate, resp. the sums (for small values), if they are connected trough an OR-Gate (see [VGRH81] for a more sophisticated computation). Therefore, FTA is a good starting point for the assessment of system safety.

As an example of a partial fault tree, let us consider the top level hazard *collision* for the radio-based level crossing (see Fig. 3). Collision was defined as *train on crossing, barriers not closed*. This event occurs, if either the train does not brake before the crossing despite it has no ‘release’ signal or the crossing does send a ‘release’ signal even though the barriers are not closed. The left sub-event has again two causes. Either the brakes are defective or they have not received a ‘brake’ signal from the train control. The rest of the tree can be developed by breaking down the properties to the components of the model (see [RST00] for the complete fault tree). Because of the fault tree consists of OR-gates only, the minimal cut sets are all singleton sets.

However, the informal approach of fault tree analysis offers some weaknesses. Causes for hazards can be overlooked (incompleteness), or events can be noted in the fault tree, which do not influence the

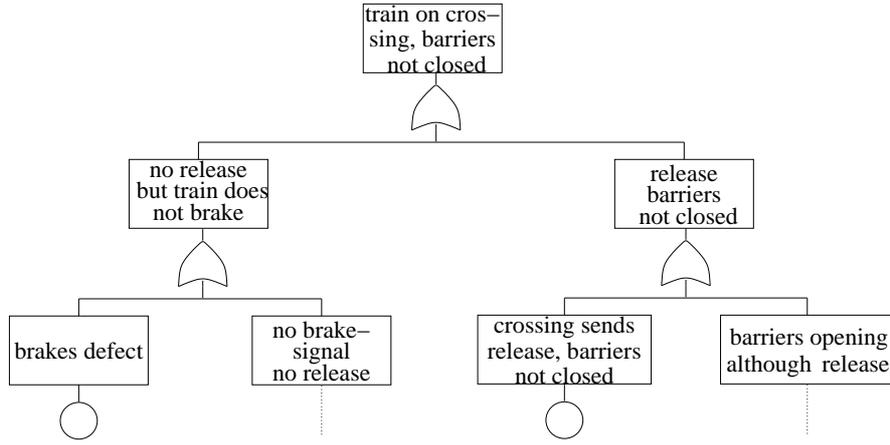


Figure 3: FTA for the hazard *collision*

safety at all (incorrectness). These errors in the application of FTA lead to incorrect safety statements for systems. To overcome these weaknesses, we propose the combination of FTA with formal methods to prove the correctness and completeness of FTA. This combination and the necessary semantics of FTA will be given in the next section.

4 Formalizing Fault Tree Analysis

The formalization of fault tree analysis allows to prove the correctness and completeness of a fault tree over a formal model. Therefore, the events of the fault tree have to be formalized, i.e. the informal events have to be translated into formulas of the formal model, and the conditions expressed through the gates within the fault tree have to be proven. These proofs guarantee correctness and completeness of the FTA. In this section, we want to give a formal interpretation for the gates, i.e. every gate of the fault tree gets a corresponding formula, which expresses the condition of the gate. Because we consider dynamic systems where the consequence of an event does often not occur immediately, new, so called cause-consequence gates, have to be introduced. To cope with dynamic systems, we formalize the fault tree conditions in Interval Temporal Logic (ITL), introduced in Sect. 4.1 and present the formalization of the fault tree in Sect. 4.2.

4.1 Logical Foundations

Our formal semantics for fault trees is based on Interval Temporal Logic (ITL, [Mos85]) with a continuous semantics similar to Duration Calculus [HZ92, CHR91]. Syntax and semantics are introduced informally in this section and we refer to [STR02] for a detailed definition.

The set of ITL-formulas is defined as an extension of the first-order formulas. Temporal formulas φ

are built from first-order formulas using propositional connectives and (amongst others) the following temporal operators:

- $\Box \varphi$: in all initial intervals φ
- $\Diamond \varphi$: in some initial interval φ
- $\boxtimes \varphi$: in all subintervals φ
- $\diamond \varphi$: in some subinterval φ
- $\varphi ; \psi$: interval can be split, s.t. φ holds in the first part and ψ in the second (read ‘ φ chop ψ ’)

The semantics of a temporal specification are all intervals \mathcal{I} starting at time point $a = 0$ which fulfill the axioms of the specification. When we prove the correctness and completeness of a fault tree gate, the corresponding condition has to be valid over the specification, i.e. the condition has to be true for every interval \mathcal{I} of the specification.

4.2 Fault Tree Semantics

The definition of gate conditions simply as the conjunction resp. disjunction of the sub-events is not always adequate for several reasons. The problems can be demonstrated with one of the nodes of Fig. 3 which is described (using suitable predicates) as:

$$\text{release} = \text{true} \wedge \neg \text{closed}(\text{barriers}) \quad (1)$$

If one defines two sub-events each containing one conjunct, then each conjunct alone does no longer describe a fault event. The fact that the barriers are not closed, does not indicate a fault *on its own*, otherwise we would have to keep the barriers closed all the time. Similarly, the fact that the train has received the ‘release’ signal and therefore assumes the crossing to be safe, is not faulty, otherwise we would have to prevent any train from passing the crossing. Therefore, our fault tree proposes two causes (2) \vee (3) for the event: Either the crossing has sent the ‘release’ signal, although the barriers were not closed, or the barriers started to open after the train received the ‘release’ signal.

$$\text{crossing_sends_release} = \text{true} \wedge \neg \text{closed}(\text{barriers}) \quad (2)$$

$$\text{release} = \text{true} \wedge \text{crossing_open_signal} = \text{true} \quad (3)$$

This decomposition shows another problem, in case we define the semantics simply as a disjunction: It does not take into account that the sub-events (causes) happen *strictly before* the main event (consequence). Therefore we will distinguish between decomposition gates (short: D-gates) with the classical boolean semantics and cause-consequence gates (short: C-gates) where the causes have to occur before the consequence.

gate g	correctness CORR(g)	completeness COMPL(g)
	$\boxtimes (\varphi_1 \wedge \varphi_2 \rightarrow \psi)$	$\boxtimes (\psi \rightarrow \varphi_1 \wedge \varphi_2)$
	$\boxtimes (\varphi_1 \vee \varphi_2 \rightarrow \psi)$	$\boxtimes (\psi \rightarrow \varphi_1 \vee \varphi_2)$
	$\diamond (\varphi_1 \wedge \varphi_2) \rightarrow \diamond \psi$	$\neg (\neg \diamond (\varphi_1 \wedge \varphi_2) ; \diamond \psi)$
	$\diamond \varphi_1 \wedge \diamond \varphi_2 \rightarrow \diamond \psi$	$\neg (\neg \diamond \varphi_1 ; \diamond \psi)$ $\wedge \neg (\neg \diamond \varphi_2 ; \diamond \psi)$
	$\diamond \varphi_1 \vee \diamond \varphi_2 \rightarrow \diamond \psi$	$\neg (\neg \diamond (\varphi_1 \vee \varphi_2) ; \diamond \psi)$

Figure 4: semantics of fault trees

The fact, that at a C-gate the cause must occur before the consequence, implies that we cannot formalize its semantics as a simple equivalence. Instead we need two conditions. The *correctness condition* guarantees that if the cause happens, the consequence must happen too. The *completeness condition* guarantees, that all causes have been listed: the consequence must not happen without the cause. For symmetry, we also split the equivalence condition for D-gates: the implication from sub-events to main event is the correctness condition, the reverse implication becomes the completeness condition. Finally, for C-AND-gates we distinguish two cases, depending on whether the two causes must happen simultaneously to result in the consequence or not. We call the first type synchronous, the other asynchronous.

Summarizing, we get 5 types of gates: D-OR- and D-AND-gates (\hat{D} , \underline{D}), C-OR-gates (\hat{C}), and synchronous and asynchronous C-AND-gates (\hat{C} , \underline{C}). The INHIBIT-gate is defined analogous to the and gate and omitted here. The correctness- and completeness conditions for each of these types of gates are listed in Fig. 4. The case of two causes is shown, the generalization to $n > 2$ causes should be obvious.

The conditions for D-gates gives the usual boolean semantics, which should hold in any subinterval of the run. The correctness conditions for C-OR-gates says, that if in a run one of the causes occurs, then the consequence must also happen during the run. The completeness condition says, that it must not be possible to partition a run at some point of time t , such that no cause happens before t , and the consequence happens after t .

The conditions for synchronous and asynchronous C-AND-gates are similar to the C-OR-gate. Instead of one of the two causes they require both causes (synchronously or asynchronously).

We think that this formalization is adequate, because it preserves the meaning of the minimal cut sets. I.e., if all completeness conditions (see Fig. 4) of a fault tree can be verified, and if for each minimal cut set it is possible to exclude at least one of its basic events from happening on each run, then the top-level event will never happen. This theorem is formally verified with the KIV system [BRS⁺00]. For a detailed description of the theorem, the semantics of FTA, and a discussion of related work we refer to [STR02].

5 Methodology

In this section we describe our approach of the combined application of fault tree analysis and formal methods for analyzing system safety. Leveson mentions: ‘Safety must be designed into a system’ [Lev95], because adding safety aspects after system development complicates the system and makes it more susceptible to failures. Therefore we propose to integrate formal safety analysis into the system development process.

We describe four phases of system development. The first phase is used to get a detailed understanding of the application domain. The second phase consists of the analysis of the system with formal methods and fault tree analysis. These two analysis methods are applied separately, to preserve the different points of view – the functional and the safety view – to the system. In the third phase the outcome of both methods are exchanged which results in improvements of the formal model and the fault tree analysis. This phase also contains the validation of the fault tree, which can be made on two levels of formalization, called loose resp. tight coupled formal FTA. Finally, the fourth phase sums up the results.

5.1 Requirements Analysis (I)

The aim of the first phase is to get a precise definition of the *system requirements* and the *system boundary*. This requires, that developers and customers get a common understanding of the system functionality. *Semi-formal* specifications (e.g. UML diagrams) and prototypes which can be simulated can help to achieve this goal. They can also serve as a starting point for the safety analysis, as depicted in Fig. 5. In [ORS⁺02] we presented a case study, where an executable model helped to get a common understanding of the system through simulating use cases.

Second, we propose a *preliminary hazard analysis* (PHA) [Sto96] to create a list of hazards as the base for the FTA in the second phase. This list should contain all critical hazards of the system.

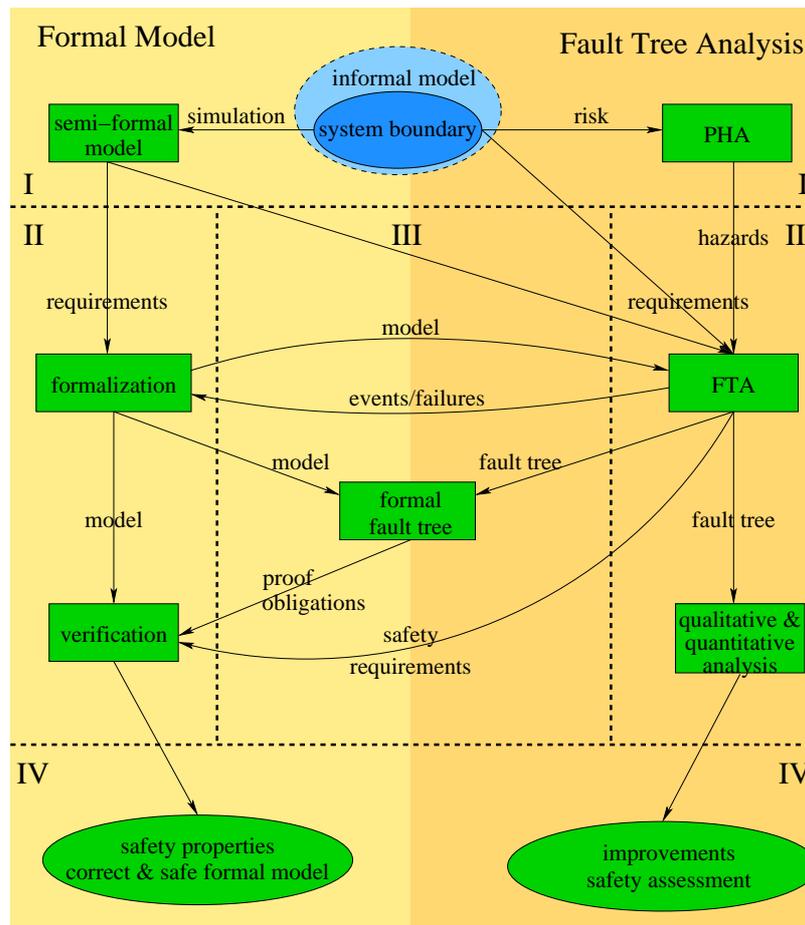


Figure 5: Process Model

5.2 Safety Analysis (II)

The second phase starts with the independent development of FTA and the formal model. Our experiences show, that it is very important to develop the formal model and the fault tree independently. Otherwise, the FTA is focused too much on the formal system description and loses the view on possible defects, failures of system components and other unspecified behavior. A precondition for this independent approach is the same understanding of the system and the system boundaries, which were defined within the first phase.

For the formal treatment of the system, first of all, a specification language for the system model has to be chosen. It has to be supported from the verification environment used for proving safety properties, as described in the next section. The *formalization* describes the informal requirements within this specification language, resulting in a formal system model. Validation against the informal description, the semi-formal specification, and *verification* of required system properties improves the formal model.

For the *fault tree analysis*, all hazards of the PHA have to be analyzed. This results in a set of fault

trees. The minimal cut sets of the fault trees are the basis of the *qualitative analysis* of the system. If the failure probabilities of the basic events are known, also *quantitative* safety assessment is possible.

5.3 Integration (III)

The third phase brings together the results of the formal and the safety analysis. The fault tree has to be adapted, to reflect the constraints of the formal model. We observed, that we often find cause-consequence relations in the fault tree which are not valid over the formal model and, therefore, have to be changed. The other way round, events occurring in and component failures detected with FTA influence the formal model. All events occurring in the FTA must be expressible within the formal model. If this is not the case, either the formal model has to be enhanced or the fault tree has to be modified, such that this event does no longer occur. In addition, the safety engineer decides, which failures have to be considered within the formal model and which can be neglected. We distinguish between hardware and software failures.

Hardware vs. Software Failures FTA of software-based systems discovers hardware failures as well as software faults. Because hardware failures can not be eliminated (defects can not be ‘ruled out’) only the probability of their occurrence can be decreased by using more reliable components or adding redundancy. To get an adequate system model, these hardware failures have to be respected within the formal model.

For software faults we propose another approach. Like Leveson we propose to fix the faults: ‘if design errors are found in the tree through this process, they should be fixed rather than left in the code and assigned a probability’ [Lev95]. This should be done by validation of the specification through verifying (safety) properties and proving software correctness against this specification.

Loose vs. Tight Coupled Formal FTA After the adaption step, two levels of formalizing the FTA exist. If the FTA and the formal model are coupled via exchanging results, we will call this a *loose coupled formal FTA*. The FTA is mainly used to derive safety properties, which will be proven over the formal system model. Although the exchange between the FTA and the formal model is not formalized, different case studies [ORS⁺02, RST00] have shown mutual benefits. The formal model gets systematically derived safety properties and the FTA a formal basis.

In addition to the safety properties, the *tight coupled formal FTA* formalizes the events of the fault tree with formulas from the formal model. The verification of the proof conditions for the fault tree gates (as presented in Sect. 4) guarantee, that the fault tree is correct and complete. If, in addition, the formal model can rule out the occurrence of (at least) one event of every minimal cut set, the analyzed hazard cannot occur at all. This fact is guaranteed by the minimal cut set theorem [STR02].

Model-Checking vs. Interactive Verification Both, the loose and tight coupled formal FTA require verification. If the system can be represented as a finite, discrete model, proof conditions for restricted fault trees can be verified by model checking. In [TSR02], these restrictions and the corresponding proof conditions for CTL model-checkers are described.

If model-checking is not applicable, interactive verification is required. The interactive verification environment KIV [BRS⁺00] supports verification of ITL formulas, necessary for proving the proof conditions for fault tree gates. Currently, specification support for statecharts [HN96] is added, as described in [BT02], to model dynamic systems.

5.4 Results (IV)

Finally, the fourth phase summarizes the results of both techniques. The overall process results in a correct and safe formal model with verified safety properties. In addition to the safety statement for an operational system, the safety analysis assesses the systems safety with respect to component failures and assists in detecting system improvements through qualitative and quantitative analysis. E.g., the integrated technique found different improvements for the industrial case study presented in [ORS⁺02], which are likely to be implemented in the real project.

Summarizing, we think that the presented approach is very useful for developing high assurance system specification.

6 Conclusion

This paper described the combination of two typical safety analysis techniques: fault tree analysis from engineering and formal methods from software engineering. This combination was developed to cope with software-based systems, like the radio-based crossing control which we used for explanation, where both disciplines get together.

For the presentation of the formal fault tree analysis we introduced the conventional FTA, necessary formal foundations, and the formalization of FTA. The formalization resulted in proof obligations guaranteeing complete and correct fault trees.

The focus of this paper is the presentation of the methodology for the application of the formal FTA. An essential point is, that FTA and formal methods are applied independently, at first, and stepwise adjusted within the analysis process. This retains the different points of view of both methods and nevertheless ends up with an integrated model. The integration of the FTA and the formal model can be done loosely or tightly coupled, resulting in different levels of formalization. The first level

exchanges safety properties which are proven over the formal model. The second level also formalizes the fault tree and proves it correct and complete.

We propose to apply both levels for high assurance systems, as transportation control systems are. For finite systems, the proof conditions can be verified by model checking. For infinite systems, we are developing proof support for the interactive verification of fault tree conditions in the KIV system. Statechart specifications [HN96] and the corresponding proof calculus is integrated [BT02, BDRS02] to specify and verify dynamic systems.

References

- [BDRS02] M. Balsler, C. Duelli, W. Reif, and G. Schellhorn. *Verifying concurrent systems with symbolic execution*. Journal of Logic and Computation (Special Issue), 2002. (to appear).
- [Bet] *Betriebliches Lastenheft für FunkFahrBetrieb*. Stand 1.10.1996.
- [BRS⁺00] M. Balsler, 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. Springer, 2000.
- [BT02] M. Balsler and A. Thums. *Interactive verification of statecharts*. In Integration of Software Specification Techniques (INT'02), 2002.
- [CHR91] Zhou Chaochen, C. A. R. Hoare, and Anders P. Ravn. *A calculus of durations*. Information Processing Letters, 40(5):269–276, December 1991.
- [DJHP98] W. Damm, B. Josko, H. Hungar, and A. Pnueli. *A compositional real-time semantics of STATEMATE designs*. In W.-P. de Roever, H. Langmaack, and A. Pnueli, editors, COMPOS' 97, volume 1536 of LNCS, pages 186–238. Springer-Verlag Berlin Heidelberg, 1998.
- [FMNP95] P. Fenelon, J. McDermid, A. Nicholson, and D. Pumfrey. *Experience with the application of HAZOP to computer-based systems*. In Proceedings of the 10th Annual Conference on Computer Assurance, Gaithersburg, MD, 1995. IEEE.
- [HN96] D. Harel and A. Naamad. *The statechart semantics of statecharts*. ACM Transactions on Software Engineering and Methodology, 5(4):293–333, October 1996.
- [HZ92] M. R. Hansen and Zhou Chaochen. *Semantics and completeness of the Duration Calculus*. In J. W. de Bakker, K. Huizing, W.-P. de Roever, and G. Rozenberg, editors, Real-Time: Theory in Practice, volume 600 of LNCS, pages 209–225. Springer-Verlag, 1992.

- [JS99] L. Jansen and E. Schnieder. *Referenzfallstudie Bahnübergang – Referenzfallstudie im Bereich Verkehrsleittechnik des DFG-SPP Softwarespezifikation*. Technical report, Institut für Regelungs- und Automatisierungstechnik, <http://www.ifra.ing.tu-bs.de/m33/spezi/>, 1999. in German.
- [KT02] J. Klose and A. Thums. *The STATEMATE reference model of the reference case study ‘Verkehrsleittechnik’*. Technical Report 2002-01, Universität Augsburg, 2002.
- [Lev95] N. Leveson. *Safeware: System Safety and Computers*. Addison Wesley, 1995.
- [Mos85] B. Moszkowski. *A temporal logic for multilevel reasoning about hardware*. IEEE Computer, 18(2):10–19, 1985.
- [ORS⁺02] F. Ortmeier, W. Reif, G. Schellhorn, A. Thums, B. Hering, and H. Trappschuh. *Safety analysis of the height control system for the Elbtunnel*. In Proceedings SAFECOMP 2002, pages 296 – 308, Catania, Italy, 2002. Springer LNCS 2434.
- [PS91] A. Pnueli and M. Shalev. *What is in a step: On the semantics of statecharts*. In Symposium on Theoretical Aspects of Computer Software, pages 244–264, 1991.
- [Rei79] D. Reifer. *Software failure modes and effects analysis*. IEEE Transactions on Reliability, 28(3):147 – 249, August 1979.
- [RST00] W. Reif, G. Schellhorn, and A. Thums. *Safety analysis of a radio-based crossing control system using formal methods*. In E. Schnieder and U. Becker, editors, *9th IFAC Symposium Control in Transportation Systems 2000*, June 2000.
- [Sto96] N. Storey. *Safety-Critical Computer Systems*. Addison-Wesley, 1996.
- [STR02] 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.
- [TSR02] 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.
- [VGRH81] W. E. Vesely, F. F. Goldberg, N. H. Roberts, and D. F. Haasl. *Fault Tree Handbook*. Washington, D.C., 1981. NUREG-0492.