Integrated Design and Process Technology, **IDPT-2002**
Printed in the United States of America, June, 2002
©2002 Society for Desing and Process Science

Formal Fault Tree Semantics

Gerhard Schellhorn, Andreas Thums, Wolfgang Reif
Lehrstuhl für Softwaretechnik und Programmiersprachen
Universität Augsburg
D-86135 Augsburg, Germany
email:*{*schellhorn,thums,reif*}*@informatik.uni-augsburg.de

ABSTRACT: In train control systems, more and more (electro-)mechanical devices are substituted by software based devices. To sustain the high level safety standards for these embedded systems, we propose the integration of fault tree analysis and formal methods. This combines two important safety analysis methods from the involved domains of engineering and software development.

Our approach proposes to build a formal model of the system together with fault trees, which investigate the safety critical aspects by breaking them down to software and hardware requirements. The events of fault trees are formalized with respect to the model. Formal completeness and correctness conditions are given, using Interval Temporal Logic with continuous semantics. They define a formal semantics of fault trees, which allows cause-consequence relations between events in addition to boolean decomposition. The semantics is therefore suitable for dynamic systems. We will prove, that the conditions guarantee, that the fault tree is a correct and complete analysis of the causes of the considered fault.

I. INTRODUCTION

In engineering, a number of very useful techniques have been developed to analyze safety-critical systems. Examples are fault tree analysis (FTA) [21], failure mode and effects analysis (FMEA) [20], or hazard and operability analysis (HAZOP) [8]. 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. The idea is to formally specify the system model, and to check consistency and other properties derived from the safety analysis by formal verification techniques.

Originally, safety analysis was applied on hardware components and formal methods mainly on software. For systems like the train control example it is necessary to integrate both.

Work supported by the DFG (Deutsche Forschungsgemeinschaft), priority programme 'Integrating Software Specification Techniques for Engineering Applications'

In this paper we present a formalization of fault tree analysis. An example of a decentralized radio-based crossing control serves to explain our approach [14]. Fault tree analysis breaks down a system hazard to component failures step by step by linking failure events with their causes. Because fault tree analysis is used for qualitative and quantitative analysis of the system, it is essential that for a hazard every cause is considered in the fault tree and, the other way round, that every mentioned cause is actually needed to trigger the hazard. If some causes do not lead to the hazard, the analysis considers the system less safe than it is and if some causes are ignored, the system seems safer than it actually is.

Therefore we developed verification conditions which link fault events and their causes. These conditions express that every cause is considered and that every mentioned cause actually is one. The conditions have to be verified over the formal system model to get a complete and correct fault tree. Thereby we get a highly reliable safety analysis of the system which guarantees that no additional failure cause for the system hazard has been overlooked.

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. By stepwise adjusting the notions used in both models, the events of the fault tree are formalized, so that they can be expressed in the formal model and the corresponding correctness and completeness conditions can be verified.

In this paper we will focus on the formal semantics for fault trees and refer to [19] for the methodological aspects and to [15] for the formal description of the radio-based crossing control example, which will informally be described in Sect. II. Sect. III sketches the fault tree analysis technique in general. The logical foundations for the formalization of FTA will be given in Sect. IV and the formalization itself in Sect. V. Some related work is discussed in Sect. VI. Finally, Sect. VII describes our current work on the development of a formal safety analysis tool, and Sect. VIII concludes.

II. THE RADIO-BASED CROSSING CONTROL

The German railway organization, Deutsche Bahn, prepares a novel technique to control level crossings: the de-

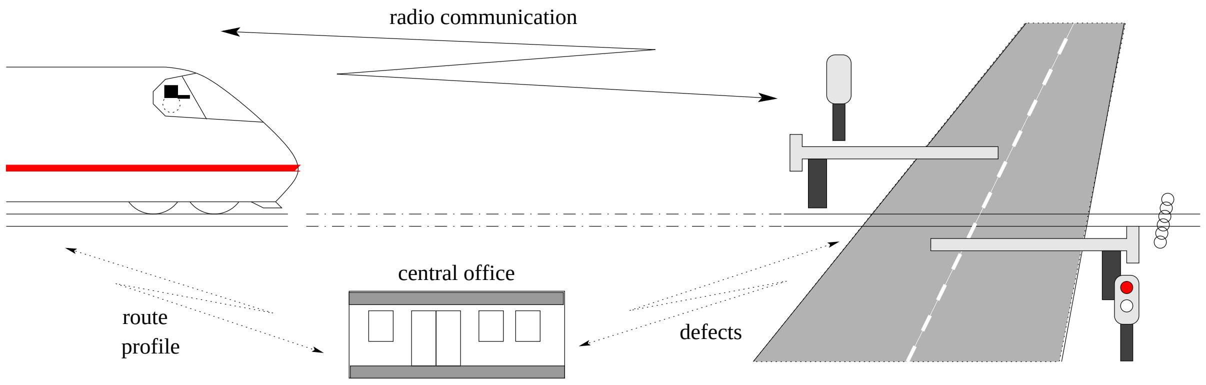


Fig. 1. Radio-Based Crossing Control System

centralized, radio-based level crossing control [4]. This technique aims at medium speed routes, i.e. routes with maximum speed of 160 km/h. An overview is given in Fig. 1. The main difference between this technology and the traditional control of level crossings is, that signals and 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 'secure', it switches on the traffic lights, and then 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 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 trains. These descriptions indicate the positions of level crossings and maximum speed on the route.

III. Fault Tree Analysis (FTA)

FTA 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 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 has 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

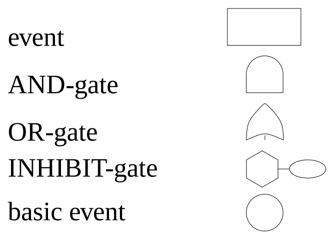


Fig. 2. Fault Tree Symbols

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. This information helps 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.

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.

Our aim is to formalize this analysis technique and to get verification conditions which guarantee that the fault tree is correct and complete, corresponding to a given model. Nevertheless the meaning of the (minimal) cut sets should be preserved, because they give hints for improving the system.

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 reasons. 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 [19] for the complete fault tree).

IV. Logical Foundations

Our formal semantics for fault trees is based on Interval Temporal Logic (ITL, [17]) with a continuous semantics similar to Duration Calculus [13], [6]. Syntax and semantics are defined briefly in this section.

The set of ITL-formulas is defined as an extension of the first-order formulas φ over a many-sorted signature SIG and a set of variables X. We assume the reader is familiar with the semantics of such formulas using SIG-algebras $\mathcal{A}$ and valuations (or states) σ, which map the variables of X to appropriate elements of the carrier sets of $\mathcal{A}$. The set of states is denoted S. An interpretation $\mathcal{I}$: I $\longrightarrow$ S is a valuation of the variables for all points of time within an interval I. The interval I may be finite, then I = [a,b] with two real numbers $0 \leq$ a $\leq$ b. Or it may be infinite, i.e. I = [a,∞) with $0 \leq$ a. ITL-formulas are evaluated over an algebra $\mathcal{A}$ and an interpretation $\mathcal{I}$. A first-order formula φ is true over $\mathcal{A}$ and $\mathcal{I}$ (i.e. $\mathcal{A}, \mathcal{I} \models \varphi$ holds), if φ evaluates to true over $\mathcal{A}$ and the state $\mathcal{I}$(a) using the usual first-order definition (where a is the left border of the interval on which $\mathcal{I}$ is defined).

Temporal formulas are built from first-order formulas using propositional connectives and the following temporal operators[1]: $\boxed{\text{i}}\ \varphi$ (“in all initial intervals φ”), $\boxed{\text{x}}\ \varphi$ (“in all subintervals φ”), and φ **;** ψ (read φ chop ψ: “the interval can be split, such that φ holds in the first part and ψ in the second”). The semantics of the temporal operators is defined as follows (assuming $\mathcal{I}$ is defined on the interval [a,b], and $\mathcal{I}|_{[c,d]}$ is the restriction of $\mathcal{I}$ to [c,d]):

- $\mathcal{A}, \mathcal{I} \models \boxed{\text{i}}\ \varphi$ iff
  for all c $\neq \infty$ with a $\leq$ c $\leq$ b: $\mathcal{A}, \mathcal{I}|_{[a,c]} \models \varphi$
- $\mathcal{A}, \mathcal{I} \models \boxed{\text{x}}\ \varphi$ iff
  for all c $\neq \infty$,d with a $\leq$ c $\leq$ d $\leq$ b: $\mathcal{A}, \mathcal{I}|_{[c,d]} \models \varphi$
- $\mathcal{A}, \mathcal{I} \models \varphi$ **;** ψ iff
  there is c $\neq \infty$ with a $\leq$ c $\leq$ b such that
  $\mathcal{A}, \mathcal{I}|_{[a,c]} \models \varphi$ and $\mathcal{A}, \mathcal{I}|_{[c,b]} \models \psi$

$\Diamond_{\text{i}}\ \varphi$ (“in some initial interval φ”) and $\Diamond_{\text{x}}\ \varphi$ (“in some subinterval φ”) abbreviate $\neg\ \boxed{\text{i}}\ \neg\ \varphi$ resp. $\neg\ \boxed{\text{x}}\ \neg\ \varphi$.

A temporal logic specification SPEC = (SIG,Ax) consists of a signature SIG and a number of temporal logic formulas Ax. Its semantics SEM(SPEC) are all pairs ($\mathcal{A}$, $\mathcal{I}$), such that $\mathcal{I}$ (called “run” or “trace”) starts at time a = 0 and $\mathcal{A}$, $\mathcal{I} \models \psi$ holds for all axioms $\psi \in$ Ax. We call a run finite (or terminating) if b $< \infty$, infinite otherwise. A formula φ is valid over a specification SPEC (in short SPEC $\models \varphi$), iff $\mathcal{A}, \mathcal{I} \models \varphi$ holds for all ($\mathcal{A}$, $\mathcal{I}$) $\in$ SEM(SPEC).

V. Fault Tree Semantics

In this section we will define a formal semantics for fault trees. The basis for such a formalization is a formal model for the system under consideration, in our case a formal specification SPEC of the level crossing of Sect. II. The second step consists in replacing the informal descriptions of fault events described by the nodes of the fault tree by formal descriptions using the signature of this specification. We have done this for the fault tree given in Fig. 3.

For the formal description of fault events we use the continuous Interval Temporal Logic described in Sect. IV. In contrast to simpler temporal logics as LTL or CTL* this allows to formalize events which describe continuous changes and have a duration like “speed increasing linearly by 1 m/sec^2 for 10 seconds”. Such events are necessary in the example, where train speed is needed to compute the latest braking point before the crossing.

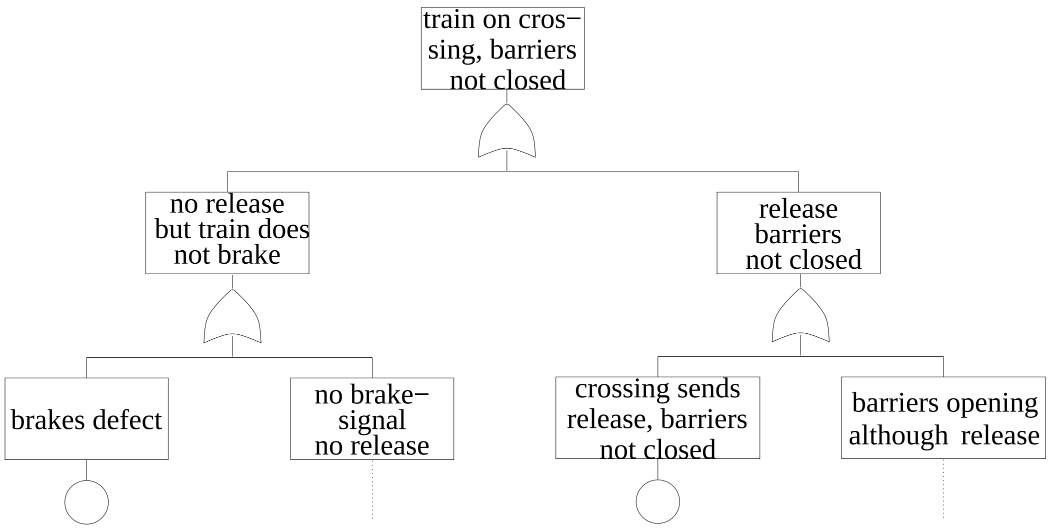
Based on SPEC and the formulas in the nodes of the fault tree the result of our semantics will be a set of *conditions* (also formulas of ITL) for each gate of the fault tree. Verification of all these conditions over the formal specification of the system guarantees, that the top event of the fault tree is indeed adequately partitioned into the basic events at the leaves. We will prove theorems to this effect at the end of the section.

A first naive approach to the definition of such conditions, which is suggested by most informal presentations of FTA such as [21] or [16], is to define the semantics of AND- and OR-gates simply as the conjunction resp. disjunction of the sub-events. The condition derived from this semantics then is the equivalence between the conjunction (disjunction) of the formulas describing the sub-events and the formula describing the main event of the gate. This approach seems attractive since it suggests the automatic construction of gates by splitting along the formula structure: if the top event is described as a formula of the form $\varphi_1 \wedge \varphi_2$ then it seems natural to define an AND-gate with sub-nodes φ_1 and φ_2.

But this simple approach is wrong for several reasons. The problems can be demonstrated with one of the nodes of Fig. 3 which is described (using suitable predicates) as:

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

[1]ITL also defines quantification and many other derived operators not needed here



train on crossing, barriers not closed
no release but train does not brake
release barriers not closed
brakes defect
no brake-signal no release
crossing sends release, barriers not closed
barriers opening although release

Fig. 3. FTA for the hazard *collision*

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 it's 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 over the crossing.

Therefore we must reject splitting the conjunction with an AND-gate in this case. Note that the Fault Tree Handbook [21], p. IV-3 explicitly requires an intermediate event to be "A *fault* event that occurs because ...".

In fact, 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.

$$\text{crossing\_sends\_release} = \text{true} \wedge \neg\, \text{closed(barriers)} \quad (2)$$

$$\text{release} = \text{true} \wedge \text{crossing\_open\_signal} = \text{true} \quad (3)$$

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).

That this problem is usually neglected in the literature, is mainly due to the fact, that only simple hardware systems are analyzed, where time delays between causes and consequences are unimportant. Then all gates are decomposition gates (short: D-gates), which simply partition the set of faulty states into subsets. Since we consider dynamic systems, where timing conditions are essential, we will distinguish such D-gates from cause-consequence gates (short: C-gates). Such a distinction is proposed informally in [9], too.

The fact, that in 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 must distinguish two cases, depending on whether the two causes must happen simultaneously to result in the consequence. We call the first type synchronous, the other asynchronous.

Summarizing, we get 5 types of gates: D-OR- and D-AND-gates (D, D), C-OR-gates (C), and synchronous and asynchronous C-AND-gates (C, AC). Two additional gates, C-INHIBIT- and D-INHIBIT-gates (C, D), will be explained below. 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. Note that it is not suitable to require, that the consequence must happen *after* the cause for two reasons. First, there may be several occurrences of given causes and the consequence then is required to occur only after the first occurrence (if "brake failure" leads to a train crash, then a subsequent occurrence of "barriers do not close" will not result in a second train crash). Second, there may be other causes for the consequence, since the causes have not been listed completely. The latter case is excluded

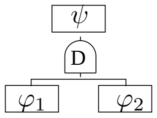
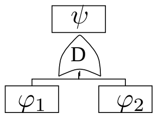
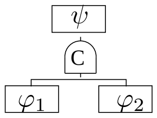
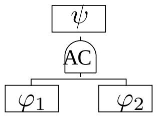
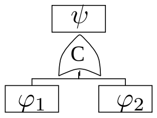
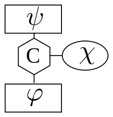
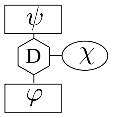
gate g	correctness CORR(g)	completeness COMPL(g)
	$\boxed{x}\,(\varphi_1 \wedge \varphi_2 \rightarrow \psi)$	$\boxed{x}\,(\psi \rightarrow \varphi_1 \wedge \varphi_2)$
	$\boxed{x}\,(\varphi_1 \vee \varphi_2 \rightarrow \psi)$	$\boxed{x}\,(\psi \rightarrow \varphi_1 \vee \varphi_2)$
	$\Diamond_x\,(\varphi_1 \wedge \varphi_2) \rightarrow \Diamond_x\,\psi$	$\neg\,(\neg\,\Diamond_x\,(\varphi_1 \wedge \varphi_2)\,;\,\Diamond_i\,\psi)$
	$\Diamond_x\,\varphi_1 \wedge \Diamond_x\,\varphi_2 \rightarrow \Diamond_x\,\psi$	$\neg\,(\neg\,\Diamond_x\,\varphi_1\,;\,\Diamond_i\,\psi) \wedge \neg\,(\neg\,\Diamond_x\,\varphi_2\,;\,\Diamond_i\,\psi)$
	$\Diamond_x\,\varphi_1 \vee \Diamond_x\,\varphi_2 \rightarrow \Diamond_x\,\psi$	$\neg\,(\neg\,\Diamond_x\,(\varphi_1 \vee \varphi_2)\,;\,\Diamond_i\,\psi)$
	$\Diamond_x\,(\varphi_1 \wedge \chi) \rightarrow \Diamond_x\,\psi$	$\neg\,(\neg\,\Diamond_x\,(\varphi_1)\,;\,\Diamond_i\,\psi)$
	$\boxed{x}\,(\varphi_1 \wedge \chi \rightarrow \psi)$	$\boxed{x}\,(\psi \rightarrow \varphi_1)$

Fig. 4. semantics of fault trees

by the completeness condition, which 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. Our completeness condition is stronger than the condition

$$\boxed{i}\,(\Diamond_x\,(\psi) \rightarrow \Diamond_x\,(\varphi_1 \vee \varphi_2)) \tag{4}$$

proposed in [12], since we require that the cause must have been *completed*, before the consequence can happen. The formalization (4) just requires, that the cause must have *started* before the consequence finishes, which is not adequate for events with a duration. Consider e.g. the event (consequence) “barriers not closing for 60 seconds although signal to close them was sent”. A cause for this may be be “power switch breaks”. But if we immediately see the consequence and the power switch breaks only after 30 seconds, there must have been another cause. Nevertheless (4) can be shown, which falsely asserts that the analysis is already complete.

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

When we tried to verify the conditions of the example fault tree, we found that sometimes the correctness condition is violated, since the number of states considered faulty by the sub-events is larger than the ones of the main event of the gate. We found that such gates typically represent design decisions. An example is the topmost gate of our fault tree. The formal specification of the top-event is:

$$\text{pos(train)} = \text{pos(crossing)} \wedge \neg\, \text{closed(barriers)} \tag{5}$$

This event is decomposed into two cases, depending on whether the train has received a ‘release’ signal. In the positive case (right sub-event) we get

$$\begin{aligned} &\text{release} = \text{true} \wedge \text{pos(train)} = \text{pos(crossing)} \\ &\wedge \neg\, \text{closed(barriers)} \end{aligned} \tag{6}$$

Then we drop the condition, that the train is on the crossing, yielding formula (1). This reflects the design decision, that we consider all states described by (1) already faulty, regardless whether the train is still able to brake.

The considered gate is a D-gate, but a design decision to enlarge the set of faulty states can also be made in C-gates. The fact, that correctness conditions are sometimes unprovable, has led previous formalization attempts to either require the completeness condition only [12] or to define two different semantics for correctness and completeness [5]. We prefer to make those points in the fault tree explicit, where the set of faulty states is enlarged. To this purpose we use an INHIBIT-gate between (5) and (6) with side condition χ := *pos(train) = pos(crossing)*. The correctness condition for an INHIBIT-gate requires only that the cause *together with the condition* must lead to the consequence. Again we have two versions: A D-gate which just enlarges the state set and a C-gate which allows time to pass between cause and consequence. In our case, the side condition specifies exactly, which additional states are considered faulty. By giving the trivial side condition *true* it is also possible to specify that the faulty states described by the main event are enlarged to all states of the sub-event. The

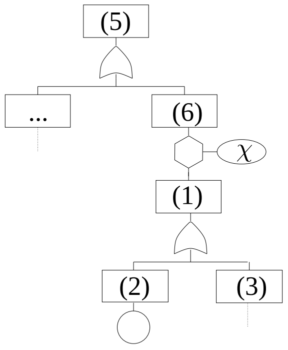


Fig. 5. formal FTA

completeness condition does not mention the side condition and just requires that without the cause the consequence must not happen. Note that our semantics of INHIBIT-gates is different from the semantics of AND-gates: the side condition of an INHIBIT-gate is allowed to describe *any* set of states, it is not necessary (and even not desirable, otherwise we would use an AND-gate) that it describes a fault. Also, side conditions will not occur in minimal cut sets described below. The formalized fault tree of Fig. 3 is shown in Fig. 5.

This completes the definition of the conditions, which have to be verified to show that the fault tree describes a meaningful decomposition of the the top-event into basic events. The advantage of formalization is, that we can now give a precise definition of the informal term “meaningful decomposition” in the form of theorems. For correctness we get:

correctness theorem

If a fault tree contains no INHIBIT-gates, no synchronous C-AND-gates and no D-AND-gates, and if all correctness conditions from Fig. 4 can be proved, then the following holds: When during a system run all primary events of some minimal cut set occur, then the top-level event must also occur.

Formally: If for all gates SPEC $\models$ *CORR(g) and SPEC* $\models \Diamond_x$ *b for all basic events b of a minimal cut set, then SPEC* $\models \Diamond_x$ *e for the top-event e.*

Minimal cut sets are computed as usual (see Sect. III). The theorem does hold neither for synchronous C-AND-gates nor for D-AND-gates with causes φ_1 and φ_2, since even the simultaneous occurrence of all primary events which are causes for φ_1 or φ_2 can not enforce, that φ_1 and φ_2 will happen at the same time. For completeness we have the following stronger theorem:

completeness theorem (minimal cut set theorem)

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.

Formally: If for all gates SPEC $\models$ *COMPL(g) and if for each cut set s SPEC* $\models \boxed{x} \neg$ *b for some element* $b \in s$*, then SPEC* $\models \boxed{x} \neg$ *e for the top-event e.*

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.

Both theorems are proved using structural induction over the size of the fault tree. The basic fact underlying both proofs is transitivity of the cause-consequence relation (in both directions), which is proved using the semantics of the involved basic temporal operators. The proofs were verified formally using an algebraic specification of the syntax and semantics of continuous Interval Temporal Logic as described in Sect. IV in KIV [2][2].

VI. Related Work

Three different semantics for fault trees are proposed in [5], where the temporal logic CTL is used to define cause-consequence gates. The first approach is the standard boolean approach, which corresponds to using D-gates only. The other two semantics agree with our correctness and completeness conditions reduced to the special case of discrete events without duration. No global completeness or correctness theorem is proven. Note also that [5] suggests another methodology for safety analysis, by proposing to use the conditions derived from the fault tree (or from several fault trees) as a first specification of the intended system (i.e. using them for synthesis of the system), while we propose to develop a system model separately and to analyze the system to satisfy the fault tree conditions.

Hansen’s work is the only other work we know of that

[2]note that the formalization is purely algebraic. It does not use the temporal logic we currently implement in KIV (see Sect. VII).

proposes to use a logic for a continuous setting (Duration Calculus). [12] defines a completeness condition that is close to our condition for C-gates (see Sect. V). Later work [10], [11] allows D-gates only, which makes all formulas in intermediate nodes definable as boolean combinations of primary events. With this semantics it is impossible to define causes which lead to an effect only after some time. Therefore our fault tree for the level crossing is impossible to define in this setting.

[9] defines a predicate logic approach to safety analysis. Although we found the idea of distinguishing between D- and C-gates useful, we do not know how the approach should be used in practice, since it requires a predefined causal relationship between events, not offered by most specification methods (e.g. statecharts or algebraic specification).

Finally, it should be noted, that all of the cited approaches transform INHIBIT-gates to AND-gates. We give them a different semantics, exploiting the fact, that side conditions do not have to be fault events.

VII. A Formal Safety Analysis Tool

The formal semantics of fault tree analysis defined in Sect. V is currently used to realize a formal fault tree analysis tool, which will allow the integrated development of formal system descriptions and formal fault trees. It will also allow to formalize the events and to prove the completeness and correctness conditions, presented in Sect. V. As an implementation platform, we use the specification and verification environment KIV [2] which already offers strong proof support for algebraic specifications with higher order logic and for the verification of sequential programs. Verification of temporal properties with interval temporal logic with a discrete semantics is already possible [1], and more deduction support is added currently. Support for fault trees still has to be implemented.

To model systems we not only use temporal logic but also allow statecharts in the KIV system [3]. Since we use the STATEMATE[3] semantics of statecharts [18], which was formalized in [7], we are able to use the STATEMATE model for the level crossing we co-developed in [15] directly as our system model. First experiments with the verification of ITL properties show, that an overall verification of fault tree conditions should be possible.

VIII. Conclusion

In this paper we presented an approach to the safety analysis of systems, which integrates fault tree analysis from engineering and formal specification from software development.

Because software-based systems have a dynamic behavior, fault tree analysis was enhanced to capture this behavior within the fault tree. This resulted in a definition of the semantics of seven different kinds of fault tree gates with correctness and completeness conditions in ITL. The semantics allows to define cause-consequence relations, where time may pass between cause and consequence. It satisfies the minimal cut set theorem which states, that if for every minimal cut set one event can be prevented, the hazard may not occur. To our knowledge this is the first temporal semantics for fault trees for dynamic systems, which respects the meaning of minimal cut sets from classical fault tree analysis.

Currently we are working on an extension of the specification and verification environment KIV, which will allow to develop and formalize fault trees, and to generate and verify the correctness and completeness conditions.

[3]STATEMATE is a registered trademark of i-Logix Inc.

References

[1] M. Balser, C. Duelli, W. Reif, and G. Schellhorn. Verifying concurrent systems with symbolic execution. *Journal of Logic and Computation (Special Issue)*, 2002. (to appear).

[2] 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. Springer, 2000.

[3] M. Balser and A. Thums. Interactive veri£cation of statecharts. In *INT2000: Integration of Speci£cation Techniques with Application in Engineering*, 2002.

[4] Betriebliches Lastenheft für FunkFahrBetrieb. Stand 1.10.1996.

[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] Zhou Chaochen, C. A. R. Hoare, and Anders P. Ravn. A calculus of durations. *Information Processing Letters*, 40(5):269–276, December 1991.

[7] 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.

[8] 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.

[9] J. Górski. Extending safety analysis techniques with formal semantics. In F. J. Redmill and T. Anderson, editors, *Technology and Assessment of Safety Critical Systems*, pages 147 – 163, London, 1994. Springer Verlag.

[10] K. Hansen. *Linking Safety Analysis to Safety Requirements*. PhD thesis, Danmarks Tekniske Universitet, Lyngby, August 1996.

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

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

[13] 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 *Lecture Notes in Computer Science*, pages 209–225. Springer-Verlag, 1992.

[14] L. Jansen and E. Schnieder. Referenzfallstudie Bahnübergang – Referenzfallstudie im Bereich Verkehrsleittechnik des DFG-SPP Softwarespezi£kation. Technical report, Institut für Regelungs- und Automatisierungstechnik, http://www.ifra.ing.tu-bs.de/m33/spezi/, 1999. in German.

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

[16] R. D. Leitsch. *Reliability Analysis for Engineers: An Introduction*. Oxford Science Publications, 1995.

[17] B. Moszkowski. A temporal logic for multilevel reasoning about hardware. *IEEE Computer*, 18(2):10–19, 1985.

[18] 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.

[19] W. Reif, G. Schellhorn, and A. Thums. Safety analysis of a radio-based crossing control system using formal methods. In *9th IFAC Symposium on Control in Transportation Systems 2000*, 2000.

[20] D. Reifer. Software failure modes and effects analysis. *IEEE Transactions on Reliability*, 28(3):147 – 249, August 1979.

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