Formal Safety Analysis of Transportation Control Systems

Frank Ortmeier, Wolfgang Reif

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

Abstract. From a safety point of view different transportation systems share many properties and requirements. They are all supposed to function correctly, to be failure tolerant and to be associated with only minimal risk. To ensure these requirements formal methods are a great help.
The ForMoSA approach provides an integrated methodology for formally analyzing safety-critical transportation systems. One important and difficult part of all formal analysis methods is the building of precise and correct models of the system. In the domain of safety analysis this task is even more complex, because not only intended behavior must be modeled but unintended (faulty) behavior as well.
In this paper we show how functionally correct models of systems can be extended such that they not only model the system's functional behavior but possible failures as well. We give construction rules which assure, that the extended model is a conservative extension of the intended behavior. We demonstrate the method with a railroad example: the autonomous control of a radio-based level crossing.

Key words: safety analysis, formal methods, failure modes, failure models

1 Introduction

New technologies are emerging in every aspect of railways. Let it be electronic scheduling and speed control, new propulsion techniques like in the german ICE 3, where the engines are distributed among all cars of the whole train or autonomous, intelligent equipment on the track. This all results in great benefits like faster travel, less energy consumption and improved maintainability. But there is also a price to pay for it. More and more software has to be integrated into the control units, control is increasingly decentralized and the complexity of track items like track switches, signals or level crossings rises dramatically. This results in an increased risk of failure. For example the U.S. Department of Transportation lists in its 2003 report on transportation [7] more fatalities involving railroads than any other form of transportation besides cars. This rises the call for new, safer railway systems.

The most advanced techniques with respect to safety guarantees are formal methods. In [11] and [12] an integrated approach for formal safety analysis of embedded systems is presented. Systems are described as mathematical models and safety predictions can be analyzed and rigorously verified. This methodology comprises both formal methods for qualitative and quantitative assumptions. One well-known way to ensure quality of a system is to prove that it is functionally correct. This means if the systems works as intended then nothing will go wrong. Formal safety analysis on the other hand rigorously determines what must (at least) go wrong such that the system fails. Both methods rely on a formal description of the system. However, in general these two system descriptions are not identical, because for safety analysis failures must be part of the model while this is not the case for functional correctness. On the other hand models which include failures may rarely be proven to be functionally correct (because if only enough components fail the system will fail as well). So an open question is: How do the two system models depend upon each other and how can a functionally correct model be used as basis for modeling failures? This problem will be the central part of this paper.

In the following we will briefly describe the ForMoSA approach to safety analysis of critical, embedded system (Sect. 2), outline a possible methodology for modeling failure modes (Sect. 3) and demonstrate this methodology on a real world example from railroad operations (Sect. 4). Section 5 concludes the paper with some summarizing remarks.

2 ForMoSA approach

For safety analysis there exists broad variety of informal techniques like fault tree analysis (FTA) [14], failure modes and effects analysis (FMEA) [8], hazard analysis (HazAn) [5] and many others. In recent years several formal theories for the analysis of safety were developed. The oldest one is of course verification of functional correctness. Newer ones are formal FTA (FFTA) [13, 4, 1], failure-sensitive specification [9], deductive cause-consequence analysis (DCCA) [3] and others.

The ForMoSA approach [11] is a methodology of how the different formal and informal techniques may be combined best. It takes into account time, money and criticality of the system. It is clear that some methods may be combined easier and more efficient while others only benefit marginally from each other. Figure 1 shows the interaction of some techniques as a graph. Arrows represent dependencies, and the time line is from left to right.

A path in this graphs represents a safety analysis process. For example: the top trace - informal specification, formal functional model, functional correctness - represents the standard approach of formal verification of software in computer science. The bottom trace - informal specification, traditional FTA, quantitative FTA - is the methodology for traditional safety analysis with FTA. According to the system under investigation specific paths yield best results [12]. An example for this process may be found in [10], where the methodology for tightly versus loosely coupled FTA is presented. We can not present all the details here, but will focus on one part of it: the building of a system

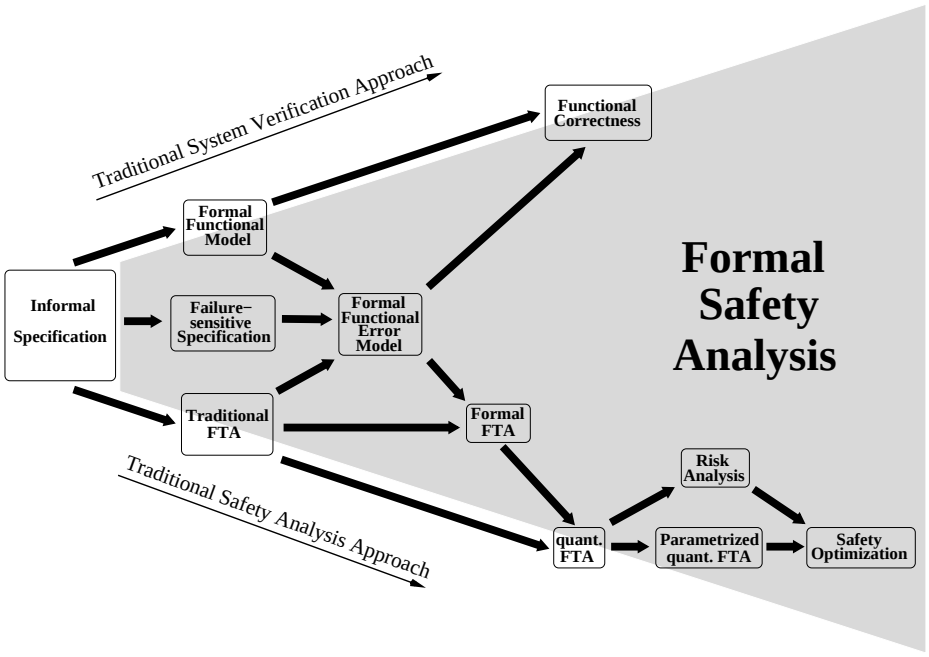

Fig. 1. Interaction of techniques

model which includes intended as well as failure behavior. This is called the functional error model.

3 Modeling failures

How system models of intended behavior are build is widely known. We assume, that the (intended) system under investigation has already been modeled in state chart notation [2]. For formal safety analysis failure modes must be explicitly modeled (and integrated into the model).

The ForMoSA approach suggests to divide the modeling of failures into two steps. Firstly model the occurrence pattern of the failure mode and – secondly – model the failure mode itself. By "occurrence pattern" we understand how and when the failure mode occurs. For example whether the failure mode occurs indeterministically (like packet loss in IP traffic), occurs once and forever (like a broken switch) or only occurs during certain time intervals (like until the next maintenance). To model this failure charts are used. Figure 2 shows two such failure charts.

State *yes* means the component is (at the moment) faulty, while state *no* means the component works correctly. The left chart models a transient failure which can indeterministically occur and disappear (e.g. loss of some radio-communication messages in the later example). The right one models a persistent failure, which happens once and stays forever (e.g. a broken relay). Maintenance etc. may be modeled analogously.

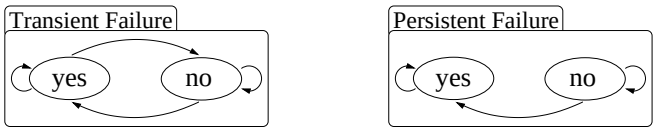

Fig. 2. Failure charts for transient and persistent failures

These charts are integrated into the formal model by parallel composition of the state charts (see Fig. 3).

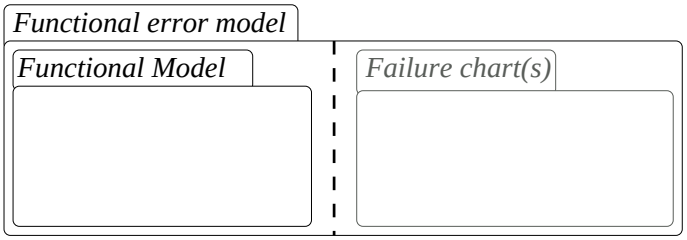

Fig. 3. Integrating failure charts

The second step is to model the direct effects of failure modes. There are three allowed possibilities for integrating the direct effect of failures in the ForMoSA approach:

1. New states and from these states outgoing transitions may be introduced. Additional states may become necessary during this process (e.g. if the bars of a railroad crossing get stuck), but new transitions leading to such states must be of the form $\varphi \wedge Failure_{Chart} = Yes$.
2. New transitions may be introduced only, if their condition is of the form $\varphi \wedge Failure_{Chart} = Yes$. This means these additional transitions – which reflect erroneous behavior – may only be taken, when a failure chart is in state *yes* i.e. when a failure occurs. The idea behind this is, that the behavior of the system only changes while some failures occur.
3. For existing transition with the activation condition φ this condition may be strengthened to $\varphi \wedge \neg(Failure_{Chart} = Yes)$. This is in particular useful, if some functional, intended behavior should deterministically be prevented by the failure. An example is that a broken motor for the bars of a railroad crossing should deterministically prevent the closing/opening of the bars.

The time model of the failure chart and the system must fit together. So if the system runs in macro steps (see the semantics of Harel for a description) then the failure charts must only change state at macro steps. All others properties of the functional model remain unchanged (in particular initial states and state hierarchy) remain unchanged. If these simple construction rules are followed, then the following theorem may be shown:

Theorem 1. *Conservative failure integration*
If a functional model SYS *of the intended behavior is extended according to the rules above, then the traces of the original model will be a subset of the possible traces of the extended model*[1].

This theorem basically says, that the original, intended behavior is a subset of the behavior the new model can show. Or in other words: if no components fail then the functional error model and the (original) functional model are equivalent. This is an important requirement. In practice unsystematic integration of failures modes into a formal model often changes the underlying intended behavior. In the following we will give an example of how this methodology works for a railway application.

4 An Example: radio-based railroad crossing

The example is taken from the German railway organization, Deutsche Bahn, which prepares a novel technique to control level crossings: the decentralized, radio-based level crossing control [[6]]. This technique aims at medium speed routes, i.e. routes with maximum speed of 160 km/h. The main difference between this technology and

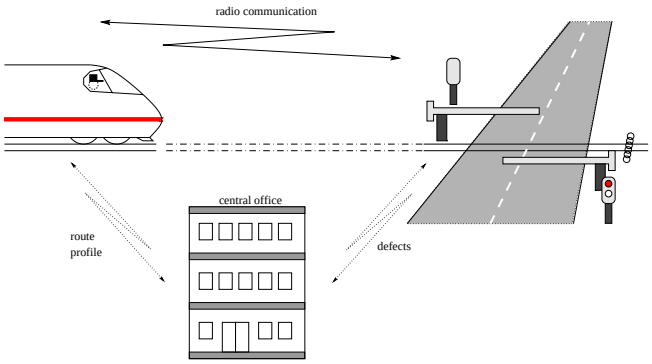

Fig. 4. Overview over the Radio-Based Crossing

the traditional control of level crossings is, that there is no central control unit. Signals and sensors on the route are connected by radio communication. Software computations in the train and in the level crossing decide if the crossing can be passed safely or if an emergency stop must be triggered.

To achieve this, 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,

[1] In detail there are two more prerequisites, which we will not go into detail here: Firstly all charts must be only in micro-step semantics and secondly the set inclusion of the traces only holds for the variables declared within the state chart. For details on this topic we refer to [12]

the time needed to secure the level crossing, and its current speed and position, which is measured by an odometer. When the level crossing receives this command 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. After this time the crossing opens the barriers again automatically.

The train requests the status of the level crossing, before reaching the latest point for a safety stop. Depending on the answer the train will brake or pass the crossing. The level crossing periodically performs self-diagnosis and automatically informs the central office about defects and problems. This solution is cheaper and more flexible than a centralized control, but also shifts safety critical functionality towards all involved components.

4.1 Functional Model

The radio-based level crossing is modeled in the formalism of state charts (of Statemate). This notation has a formal semantics [2] and is very similar to common engineering notations. The system is decomposed into the four parts train, crossing, communication and environment. These parts are modeled using state charts and activity charts. We will only discuss the crossing in detail here. A complete presentation of this case study (including formal safety analysis and quantitative approximations) may be found in [12]. Figure 5 shows the (functionally correct) model of the crossing.

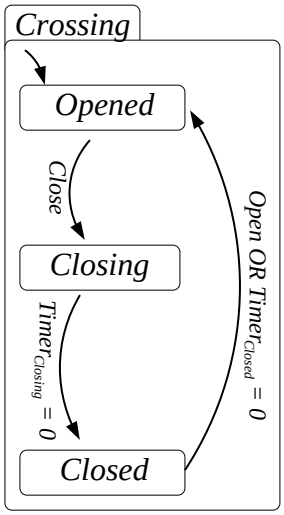

Fig. 5. Model of the crossing

Initially the crossing is (*Opened*). If it receives a request for closing (*Close*) then it starts securing the crossing. This process takes $Max_{Closing}$ time units. This is modeled by a timer ($Timer_{Closing}$). The timer is started when state *Closing* is entered and

counts down from $Max_{Closing}$ to 0. In reality this process comprises switching on yellow and red lights and slowly lowering the bars. After that the bars are closed and the crossing is safe (state *Closed*). The crossing remains in this state until it receives a request to open (*Open*) but for at most Max_{Closed} time units (this is modeled by timer $Timer_{Closed}$). This design is chosen because empirical data shows, that if the bars are closed to long then drivers of the cars tend to drive around the closed bars. A quantitative safety analysis of this scenario has shown, that this is more dangerous then opening the bars again.

4.2 Failure Modes

For the crossing three failure modes are to be examined. Firstly a request to close the crossing may get lost. This failure mode is called $Fails_{Close}$. The second type of failure is that the bars may get stuck ($Error_{Stuck}$). The last failure mode which is to be examined here is the unwanted opening of the bars: $Unasked_{Open}$. This is not a complete list of failures modes, but the relevant failure modes for this example. A complete list may be found with failure-sensitive specification and is presented in [12].

4.3 Functional error model

For the integration failure charts for each failure mode are defined (according to Sect. 3). In the example we assume, that the loss of communication is transient. The same holds for an unwanted opening of the crossing (this is normally triggered by a faulty sensor reading). However, if the bars get stuck, then this will be permanent. This is modeled in the failure chart for $Error_{Stuck}$. Once the chart is in state *Yes* it will stay there forever.It is easy to model maintenance or repair. As described in the previous section, these three charts are then put in parallel composition with the existing functional model. Figure 6 shows this composition (for the modification of chart "Crossing" see below).

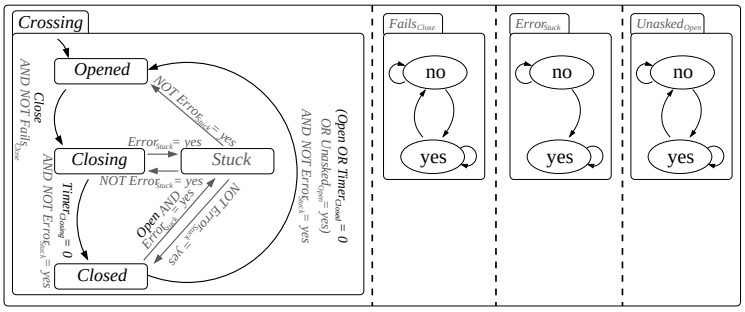

Fig. 6. Model of the crossing with failures

The next step is to model the direct effects of failures. $Fails_{Close}$ is fairly easy to model. It does not require any new states or transitions. But it does require the third rule

– i.e. some transitions are to be prevented while the failure occurs. Therefore the transition from *Opened* to *Closing* is changed to $Close \wedge \neg Fails_{Close}$. Unwanted opening of the crossing is also easy to model. This is modeled with a new transition from state *Closed* to *Opened* with condition $Unasked_{Open}$. Note that this transition has been – for readability – collapsed together with the existing transition from *Closed* to *Opened* into one transition (with condition $(Open \vee Timer_{Closed} = 0) \vee Unasked_{Open}$). The third failure mode – $Error_{Stuck}$ – is the most difficult to model. It firstly requires a new state (rule two), which is called *Stuck*; secondly it requires several new transition according to rule number one. All states in which the bars are to move, can lead to this new state. Therefore transitions from *Closing* and *Closed* to *Stuck* are introduced. In this example transitions from *Opened* to *Stuck* are not wanted, because the bars do not start moving in this state (closing always starts with the traffic lights etc.)[2]. Thirdly if this failure mode occurs then some intended behavior should be prevented deterministically. This is modeled by adding "$\wedge \neg Error_{Stuck}$" to each transition which must not be taken when this failure occurs. In the example transitions from *Closing* to *Closed* and from *Closed* to *Opened*. The functional model of the crossing is shown in Fig. 6. This model can now be used for any sort of formal safety analysis like formal FTA or DCCA. These methods allow to formally prove if certain combinations of failures may cause system failure or not. Note that the system is still functionally correct for all traces, where no failure occurs. This concludes the example.

5 Conclusion

The small example above shows how difficult it can be to derive correct formal models of failures. Comparing the fault-free model of the intended behavior (see Fig. 5) and the functional error model (see Fig. 6) it is obvious, that models including errors are by far more complex than those describing only intended behavior. The here presented methodology assures, that intended behavior won't be lost when integrating failure modes (while traces inclusion between the model of Fig. 5 and 6 is not obvious). Modeling the occurrence patterns of failures with failure charts allows a great freedom. Different types of faults, repair or maintenance sequences can be modeled in a generic and intuitive way.

For the domain of railways this approach can even be extended. Railways comprise of a certain number of standard components (e.g. crossings, tracks, cars, etc.). For such standard components standardized models either exist or can be developed. Using the presented methodology standardized models including failures could be derived. These models can then be used as the basis for formal safety analysis of most train systems.

References

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

[2] However this is a design decision which is application specific. Other views may be modeled analogously.

[2] 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, 1998.

[3] Gerhard Schellhorn F. Ortmeier, Wolfgang Reif. Deductive cause-consequence analysis (dcca). In *Proceedings of IFAC World Congress*, 2005.

[4] J. Górski and A. Wardziński. Formalising fault trees. In F. Redmill and T. Anderson, editors, *Achievement and Assurance of Safety*. Springer-Verlag Berlin Heidelberg, 1995.

[5] T. A. Kletz. Hazop and HAZAN notes on the identification and assessment of hazards. Technical report, The Institution of Chemical Engineers, Rugby, England, 1986.

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

[7] editor Marsha Fenn. Transportation statistics annual report. Technical report, US Department of Transportation Statistics, October 2003.

[8] Robin E. McDermott, Raymond J. Mikulak, and Michael R. Beauregard. *The Basics of FMEA*. Quality Resources, 1996.

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

[10] 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.

[11] F. Ortmeier, A. Thums, G. Schellhorn, and W.Reif. Combining formal methods and safety analysis – the ForMoSA approach. In *Integration of Software Specification Techniques for Applications in Engineering*. Springer LNCS 3147, 2004.

[12] Frank Ortmeier. *Formale Sicherheitsanalyse*. PhD thesis, Universität Augsburg, 2005.

[13] A. Thums. *Formale Fehlerbaumanalyse*. PhD thesis, Universität Augsburg, Augsburg, Germany, 2004. (in German).

[14] Dr. W. Vesley, Dr. Joanne Dugan, J. Fragole, J. Minarik II, and J. Railsback. *Fault Tree Handbook with Aerospace Applications*. NASA Office of Safety and Mission Assurance, NASA Headquarters, Washington DC 20546, August 2002.