# Integration of Formal Methods into System Safety and Reliability Analysis

O. Akerlund; Saab AB, Aerospace; Linkoping, Sweden

S. Nadjm-Tehrani; Dept. of Computer & Info. Science, Linköping University; Sweden

G. Stålmarck; Prover Technology AB and Dept. of Computing Science, Chalmers University; Sweden

Keywords: Formal verification, Safety analysis, Reliability analysis, NP-Tools

## Abstract

System verification and hazard analysis procedures on critical systems are traditionally carried out in separate stages of product development and by different teams of engineers. Safety and hazard analyses have for several decades been based on techniques such as fault tree analysis (FTA), whereas system verification is carried out by testing and simulation. Recent years have seen an increasing interest in application of formal methods for detecting design errors at early development stages. In this paper we propose a technique whereby both safety correctness proofs and reliability analysis, like FTA, can be performed on one design model: a model of the system in propositional logic and integer arithmetic. An obvious benefit is that the two parallel activities take place in the development process in a natural manner, and using a common model. The model is used for performing FTA-like analysis without building the fault-tree.

We describe the application with examples from the aerospace domain and show how the theorem prover NP-Tools can be used to combine the two types of analysis.

## Introduction

Safety-critical systems, especially those in aerospace, nuclear power industry and railway systems are confronted with stringent certification requirements. Recent standards in several of these areas emphasize both demands on functional correctness and safety and hazard analyses.

Detailed design and verification steps in the system development process are traditionally performed prior (or in parallel) to the safety and hazard analyses. These activities are often performed by separate teams of engineers and usually using different models and analysis environments. The resulting risk for incompatibility and incompleteness is amplified by the variety of engineering disciplines involved (mechanical, electrical, chemical, software, etc). Language barriers and methodological gaps are ample. The original fault-tree analysis technique, for example, was devised for systems consisting mostly of hardware, and attempts to apply and extend it to software are only recent.

Formal techniques for software correctness analysis are based on mathematical semantics for programming languages and design models. Hence, a methodical application of FTA to software requires a formal semantics for the standard FTA notations. Hansen et. al. provide a comprehensive overview of various semantics assigned to the FTA notation, and propose a unified underlying semantics in order to relate safety and correctness analyses to the same system model (ref. 1). They use a model whereby the system is described as a collection of state variables as function of time. Formal analysis in the Duration Calculus is then used to systematically derive requirement specifications from FTA. We have a common aspiration in that we propose the use of the same system model both for safety correctness and reliability analyses, though the notation and analysis methods we propose are based on propositional logic augmented with finite integer arithmetic (PROP+). Furthermore, in our approach the FTA is never built. Rather, the functional model of the system is extended with failure modes; reliability analysis in the sense of FTA is performed directly on the augmented model.

A recent breakthrough in application of formal methods within system verification builds on efficient analysis of systems with large state spaces. One such direction is the application of BDD-based model checking techniques, see e.g. reference 2. A BDD (Binary Decision

Diagram) is a data structure, which efficiently represents propositional formulas. That is, several boolean operations on formulas, if carried out on BDD representations, take time which is linear in the number of propositional variables used. In this paper we report on another direction whereby proofs about large systems are efficiently performed in a theorem prover for PROP+ based on Stålmarck's method (ref. 3). This method works well when the property of interest has an "easy" proof even though the system in question has a large state space. Hardness of a proof is related to the greatest number of simultaneously open assumptions in a proof. The proof tool NP-Tools is a verification environment based on Stålmarck's method. That is, it is a first order theorem prover which implements Stålmarck's algorithm.

The paper has the following structure. In section 2 we give a brief exposition to the proof technique and the application of the tool for verification of correctness in a system description. Section 3 describes reliability analysis in general, and in the context of the mentioned tool in particular. Section 4 elaborates on applications from the aerospace domain: a climatic chamber case study provided by Saab AB, as well as a report on application of the technique on the fuel subsystem of the JAS Gripen multi-role aircraft. The paper is concluded with a summary in section 5.

## Correctness Proofs using NP-Tools

In our approach systems and system requirements will be represented in PROP+. A dynamic system will be represented by a collection of state variables, input variables, output variables and a transition relation over these variables. Each state variable x in the system model is represented in the model used by NP-Tools via three variables: x(0) for initial value of x, x(t) and x(t+1) for the value of x before and after an arbitrary computation step t, respectively.

A typical requirement for a dynamic system is that "property P holds in all reachable states of the system". Such properties are referred to as safety properties in the formal verification literature; here in the context of safety and hazard analyses it would be confusing to use the same term. Therefore we refer to it as a correctness property. Such correctness properties will be proved by induction. To prove the inductive base and the inductive step for a given property, Stålmarck's algorithm as implemented in the verification tool NP-Tools (ref. 4) will be used.

In this proof environment the user-system interactions are of four types as shown in figure 1. The system model can be represented in a variety of ways, e.g. a system model can be represented as formulas in a textual editor or represented by block diagrams (resembling gate logic) in a graphical editor. The properties of the model that one is interested to prove are "Questions", also expressed as formulas. These correspond to formalized requirements on the system. The type of analysis is dependent on the question. If the objective of the question is to find out whether the property necessarily holds (for all variable assignments) the chosen mode is "Prove". In this case the outcome could be "Valid" meaning that the property holds for all variable assignments, or a counter example is produced. If the objective of the question is to check whether a particular property might hold, then the analysis mode is "Satisfy". The resulting answer could then be "Satisfiable" meaning that a satisfying assignment is found where the property is true, or "Contradictory" meaning that no variable assignments support this statement.

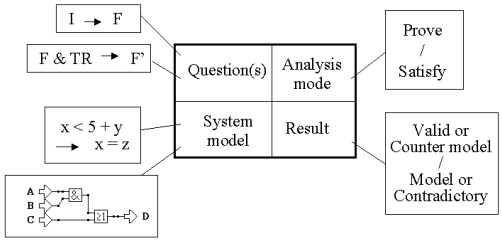


Figure 1 - User-system interactions in NP-Tools

System modeling: Systems modeling in NP-Tools is either performed via automated translators from design languages or by manual modeling in the editors of the tool. The manual modeling is often done in two steps, first by representing the system as a discrete synchronous system and secondly by a model in NP-Tools of the initial state and the transition function of the discrete synchronous system.

A discrete synchronous system is executed in discrete steps and has a given initial memory configuration, the initial state. At each step a combinatorial function f, the transition function, is computed. The function f is applied to input values and values of the internal memory cells and computes output values and updated system memory, see figure

2. We will restrict the data types of our systems to boolean and integers.

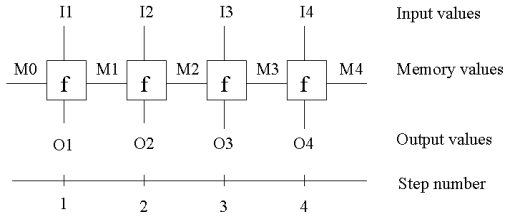


Figure 2 - The synchronous system model

Thus, we are working with a transition system model of the application, where due to synchrony and determinism the transition relation is a function f. The possible behavior of the system is uniquely determined by the initial state and the transition function.

The transformation of a system description into the discrete synchronous model involves a number of steps – for example, the identification of all memory cells (e.g. latches) and the modeling of communication between modules explicitly.

A system model will consist of two logical formulas, one formula I, characterizing the initial state, and another formula TR, representing the transition relation. In order to represent the transition function as a formula we need a notation for updated memory. We will use M(t+1) as a name for the updated value of a memory variable M(t).

k-step models: In order to prove more complex requirements the system models described above can be generalized to k-step models, i.e. models where k copies of the transition function are composed. In this way properties such as "given a state in which condition C holds, property P will hold within *n* steps from this state" can be proved. The k-step model can also be used for so called reachability analysis. A detailed description can be found in reference 5.

Translated models: NP-Tools can also read a few design languages such as Lustre (ref. 6) and Sternol (ref. 7) via translators. The automatically generated system model corresponds to the above representation, an initial state and a transition function expressed by a logical formula over state variables before and after a step.

Proving correctness properties by induction: Requirements will be expressed as logical constraints on inputs, outputs, memory and updated memory, i.e. logical formulas in these variables.

To prove that a property F holds in all states of a system represented as an initial state I and a transition function TR, we prove the two formulas:

$I \rightarrow F$ (F holds in the initial state)

$F \,\&\, TR \rightarrow F'$

where F' is the formula obtained as a result of simultaneously substituting M(t+1) for every state variable M(t) appearing in F. That is, if F holds before an arbitrary step then it will continue to hold after the step.

If the two induction formulas are valid, then the property F holds in all states of the system.

Counter models: If NP-Tools fails to prove a property, a counter model will be presented. This corresponds to an instantiation of the system variables for which the proposed formula evaluates to false. A counter model of the induction base shows how the property might fail in the initial state, and a counter model of the induction step shows how the property fails to be time invariant. This provides valuable help in finding design errors or misconceptions in requirement formulations.

Performance: Proving validity of formulas in propositional logic is a computationally hard problem. The general problem to show that a formula is true for some value assignments to its variables is NP-complete. (In practice this means that for some formulas, i.e. for some systems and/or questions, the answering time is too long to be practically useful.) This also relates to the name NP-Tools, which is intended to attack an NP-complete problem. Using techniques like BDD model checking and the Stålmarck's algorithm we get satisfactory results for many practical purposes. The advantage of Stålmarck's method for automated property proving, as compared to BDD based approaches, is the relatively low sensitivity to the size of system models (ref. 8).

## Reliability Analysis

Within the area of reliability analysis we explore two methods: Fault Tree Analysis (FTA) and Failure Mode and Effect Analysis (FMEA). Both are used to investigate the impact of hardware failures on system behavior. These methods were developed some 40 years ago when systems consisted mostly of hardware components and they were relatively uncomplicated. The much more complex systems of today, being highly integrated from hardware and software, can be difficult to analyze using these methods. We describe how to use the functional model, earlier used for correctness analysis, also for performing hardware failure analysis. This paper provides an overview and a more complete exposition is given in reference 9. We use simple examples to illustrate the ideas, but the method is equally applicable to more complex designs.

Modeling hardware failures: Usually many of the input variables to the functional model represent information originating from hardware. For example, an input variable may represent a signal coming from a button, which can be either on or off. Now, it is possible to express "why" the button is on or off in terms of "configuration-variables" and "failure mode-variables", see figure 3 for an illustration.

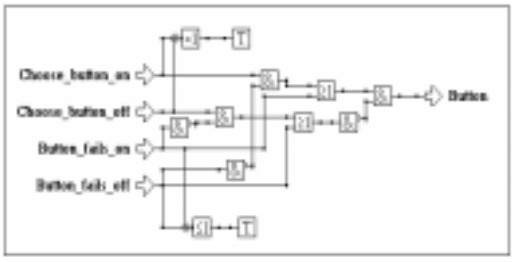


Figure 3 - Failure mode circuit for a button

The failure mode circuit is such that the "configuration-variables" express the wanted behavior of the system and the "failure mode-variables" can invalidate this behavior since the failures have priority over the choice. Note that the logic forces exactly one of the "configuration-variables" to be true and at most one of the "failure mode-variables" to be true at a time.

In the button-example only boolean variables were used. Having access also to integer arithmetic extends the possibilities for failure modes. For example, assume a system is powered by electricity, which is supplied by two batteries connected in series. Each battery gives 0 – 8 V, and the system requires 10 V to function, obtained as the sum of the two batteries. Also, none of the batteries must have a voltage of 0 V. A PROP+ model of this power supply is shown in figure 4.

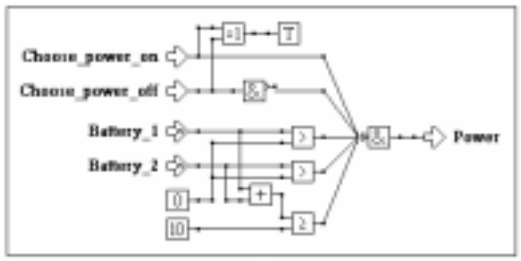


Figure 4 - Failure mode circuit of power supply from two batteries

Since Battery_1 and Battery_2 are integer variables with given domains the failure modes must be expressed accordingly. The failure modes related to this circuit are: "Battery_1 = 0", "Battery_2 = 0" and "Battery_1 + Battery_2 < 10". Also in this case the failure modes have priority over the choice being made.

Figure 5 generalizes the idea of extending the functional model with failure modes for hardware variables.

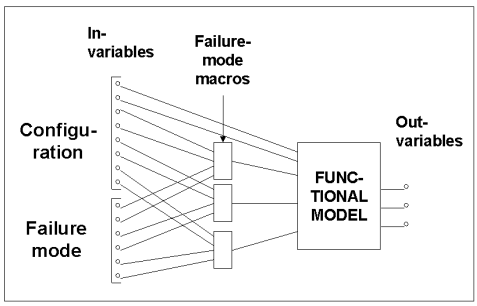


Figure 5 - Extension of functional model to include failure modes

Having access to a list of hardware failure modes we now have the necessary ingredients for performing analyses like FTA and FMEA.

Analysis for finding cut sets: The qualitative result of a FTA is the cut sets, i.e. single or combinations of failures resulting in some hazardous event, often called the "Top Event". Using the automatic verification capability in NP-Tools it is possible to find the cut sets without explicitly constructing the fault tree. The technique for doing this is to express the

top event in conjunction with the list of failure modes and some predefined system configuration in a Question (see figure 1).

It is most interesting to find minimal cut sets, i.e. sets consisting of as few failures as possible and which necessarily lead to the top event. The technique presented here makes it possible to decide how many of the failure modes may occur simultaneously. This is done using a predefined function *EQ(k,[$F_1$,..., $F_n$])*, which forces exactly *k* out of the *n* failure modes $F_1$,..., $F_n$ to be true at the same time. For example, when we are interested in finding single failures *k* is set to 1. The idea is illustrated in figure 6.

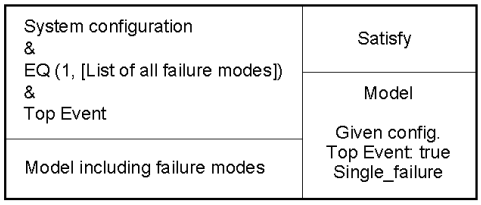


Figure 6 - Analysis set-up for finding single failures

Note that we are using the analysis mode Satisfy, which helps us to find all possible model instantiations when the specified System configuration, the EQ-function and the Top Event are true. If the result of the analysis in figure 6 is a model, we have found a single failure resulting in the Top Event. Also note that if there are no single failures causing the Top Event to occur the answer will be Contradictory, i.e. it is not possible to find any instantiation of the model having exactly one failure mode being true at the same time as the Top Event is true.

Depending on the category of the system failure function (SFF) – including both System and Question – being analyzed we can decide whether the single failure is a minimal cut set or not. If the SFF is monotonic we know that the single failure found is a minimal cut set. If it is not known whether the SFF is monotonic or not, which is mostly the case, we can perform a complementary analysis to decide if the single failure found is a prime implicant or not. This distinction between monotonic and non-monotonic SFF will be explained shortly. For non-monotonic SFF we use the word prime implicant instead of minimal cut set. The complementary analysis needed is to investigate if the single failure found in the figure 6 analysis will always, also in conjunction with other failure modes, lead to the Top Event. Figure 7 shows this set-up.

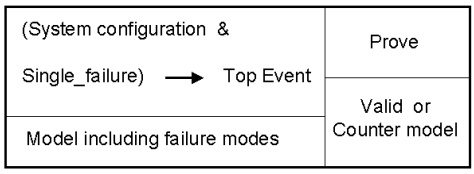


Figure 7 - Complementary analysis to investigate if a single failure is a prime implicant

Comparing figures 6 and 7 note that System configuration is the same but the EQ-function, including the list of failure modes, is not included in the second analysis. In the Question-area we claim, expressed by the implication, that the given configuration in conjunction with the single failure will always lead to the Top Event being true. To investigate if this claim is true the analysis mode is changed to Prove. The result of this analysis can be: (1) Valid, which means that the single failure found earlier is a prime implicant or (2) Counter model exists, meaning that it is possible for the Top Event not to happen even though the single failure has occurred. The reason for this contradiction must be that yet another failure (in the list of failure modes) may occur – a failure, which in conjunction with the single failure found earlier, prevents the Top Event from happening.

The next step is to examine whether there exist more than one single failure leading to the Top Event. This is done while specifying that single failures already found must not be true.

The interested reader is referred to the literature for a deeper study. The theory for safety analysis of monotonic and non-monotonic SFF can be found in reference 10. Prime implicants have for example been treated by references 11 and 12. The extension to find combinations of two or more failures and a procedure for finding prime implicants using NP-Tools is described in reference 9.

Using this style of reasoning we perform both FTA and FMEA types of analysis on the same system model. Actually, determining whether single or multiple failures lead to a hazardous event corresponds to FMEA-like analysis.

## Applications

For illustrating the methods described above we will use two examples. First a simplified climatic chamber is used where the model was obtained directly from a specification in

natural language, to a model in PROP+. Next, analysis of a part of the fuel transferring system in the Gripen aircraft is described.

**Climatic chamber**

This is a fictitious system which nevertheless exhibits some problems appearing in realistic applications.

Modeling: The climatic chamber specification prescribes the functionality of the controller for regulating the chamber temperature by means of a heater and a fan. Figure 8 illustrates the system and its environment.

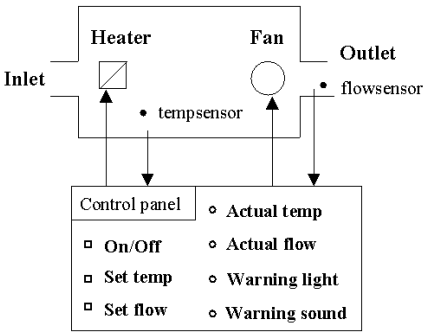


Figure 8 - An illustration of the climatic chamber system

The system consists of a chamber with a given volume. Outside air flows through the system via the inlet and leaves the chamber from the outlet. The aim is to ventilate the chamber using the heater and the fan. The actual temperature and actual flow are to be kept close to operator-selected values for temperature and flow. A condensed description of the system specification is:

1. Primarily the functionality is specified around five system states: off, wait, solution (sol), work and block. These correspond to different operating modes: wait for initialisation, solution for stabilization, work for normal operation, and block for emergency stop.

2. Initially the system starts from the off-state in which all variables are given pre-defined values.

3. Some variables are treated as constants; either because they are fixed in a particular design (e.g. thresholds), or because they are not controlled by the system (being determined by the environment at any given instance).

4. Transitions from one state to another are governed by the difference between the selected (set) temperature and the actual temperature. Depending on the size and/or the duration of a temperature discrepancy different state transitions are specified.

5. All state transitions are accompanied with different types of actions, for example resetting time to zero or switching on some warning indication.

6. The specification includes boundaries for rate of temperature change and rate of flow.

Since the system description in natural language is in a style of system-state-transitions, the modeling in PROP+ was done via a first translation into state-transition diagrams. Next this state-transition was modeled using the NP-Tools' graphical interface gate logic and integer arithmetic (PROP+). Thus, the system is mathematically represented by its state transition relation and its initial states. Figures 9 and 10 illustrate one such translation.

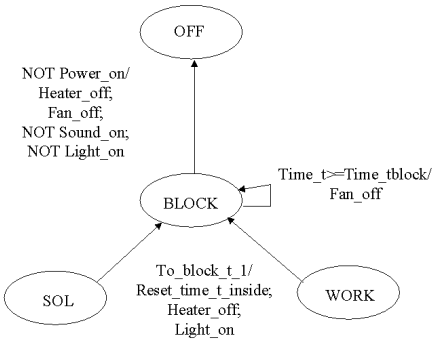


Figure 9 - Illustration of the first translation step, the state-transition format

Notice that in figures 9 and 10 the variables x(t) and x(t+1) are denoted by x_t and x_t_1 respectively.

The state transition fragment in figure 9 captures the functionality associated with the block-state. Each transition is associated with a condition for the transition to take place and one or more actions. In NP-Tools, for each state in the transition diagram there is a boolean variable representing the state, and a PROP+ model describing transitions in and out of the state. The idea is that the transition diagram can be developed in a system development language, such as statecharts (ref. 13) and the PROP+ model can be obtained by automatic translation.

Figure 10 shows the PROP+ model associated to the state transitions in figure 9.

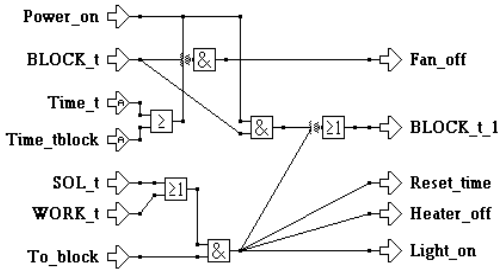


Figure 10 - The PROP+ format for how to reach the block-state

The PROP+ model also includes functionality not explicitly shown in the state transition model. For example, being in block-state it is obvious that we should remain there as long as "Power_on" is true and the fan is kept on until "Time_tblock" has elapsed. If there is more than one reason to leave a state the conditions leading away must be mutually exclusive to get a deterministic system.

Correctness Analysis: After the model is completed we start by doing "sanity" checks and correctness analysis.

One example of a sanity check is to investigate determinism. In a controller this analysis is done by claiming: given that we are in exactly one state at step t, are we also in exactly one state at the next step t+1? The analysis set up is shown in figure 11.

| | |
|---|---|
| EQ( 1, [off_t, wait_t, sol_t, work_t, block_t]) ⟶ | Prove |
| EQ( 1, [off_t_1, wait_t_1, sol_t_1, work_t_1, block_t_1]) | Valid or Counter model |
| Climatic chamber model | |

Figure 11 - Analysis to investigate determinism

The pre-defined function "EQ" is used here to force the system to be in exactly one mode at each time point. The claim about determinism is expressed by the implication, and at the same time choosing analysis-mode Prove. Getting the answer Valid indicates determinism in this sense. If the answer is Contradictory there is a possibility for the system not to end up in exactly one of the modes and this counter-model is shown by the tool – a typical case where the tool supports debugging a design model.

While building the climatic chamber model we first had non-determinism, mainly due to problems that conditions leading away from a state were not mutually exclusive and the events were not prioritized properly.

Next, the functional correctness is investigated in the absence of hardware failures. One safety related claim for the climatic chamber is that its temperature should never become too warm provided that the set temperature is within pre-defined limits. The analysis set up for this safety requirement in the three normal modes is shown in figure 12.

| | |
|---|---|
| ( wait_t_1 # sol_t_1 # work_t_1 ) & Temp_t_1 > 320 | Satisfy |
| Climatic chamber model | Possible instantiation or Contradictory |

Figure 12 - Safety correctness analysis that temperature is never to high

Getting the answer Contradictory in figure 12 means that it is impossible to end up in either of the states wait, sol or work and at the same time having a chamber temperature exceeding $320^0 C$. In this case we have chosen the hazardous temperature "too warm" to be $320^0 C$.

For this analysis we did get some counter models, indicating that our model of the rate of change of chamber temperature was too weak. After complementing the model the answer was Contradictory which proves that the "not too warm"-requirement is fulfilled in the given models.

Reliability Analysis: Before performing the reliability analysis we must extend the functional model with failure mode macros as described earlier. In this case we illustrate the idea with two failure circuits, one for the "temperature" signal which is an input to the controller and one for the warning-by-sound signal which is an output from the controller.

One of the input variables to the controller is the current temperature, "Temp(t)". It is possible to have a failure mode where the temperature sensor is wrongly calibrated by ten degrees, which will influence the temperature received by the controller. This could be modeled as follows.

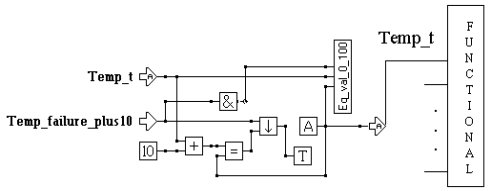


Figure 13 - Failure mode circuit for temperature sensor showing ten degrees too much

As shown in figure 13 the input variable Temp(t), on the functional macro, is expanded with the failure mode circuit. The logic of the failure circuit is such that the input variable Temp(t), to the failure mode circuit, passes unchanged to equivalent to Temp(t), to the functional model, if the boolean variable Temp_failure_plus10 is false – that is, if the failure mode has not occurred. On the other hand, if Temp_failure_plus10 is true, Temp(t) is increased by 10 before it is attached to the functional model.

An outgoing variable from the functional macro can also be affected by hardware failures. Here we show how such a failure affects the warning sound for the emergency mode. The sound failure may either be the result of an incorrect computation (logic), or there might be a hardware failure on the loud speaker itself, see figure 14.

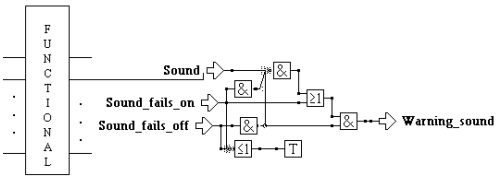


Figure 14 - Failure mode circuit for warning sound

Observe, the failure modes in figure 14 can occur at most one at a time and when occurring they have priority over the signal coming from the controller.

Using this method all hardware variables attached to the controller can be expanded with failure mode macros. This will result in the extended model shown in figure 5. The next step is to find the result of FTA using this model, i.e. to find single or combinations of failures resulting in the top event. Using our approach we can easily investigate if the safety requirement, already proven in the correctness analysis, still holds when failure modes are allowed. For example, the requirement concerning high chamber temperature, shown in figure 12, can be reanalyzed to investigate whether any failure mode makes overheating possible. Having access to the list of failure modes the FTA-like analysis is shown in figure 15.

| | |
|---|---|
| EQ( 1, [Temp_failure_plus10, Sound_fails_off, . . . ]) & ( wait_t_1 # sol_t_1 # work_t_1 ) & Temp_t_1 > 320 | Satisfy |
| Climatic chamber with failure modes | Possible instantiation including 1 failure or Contradictory |

Figure 15 - Analysis set-up to find single failures in the climatic chamber

We know that overheating is not possible in the absence of failures. Getting a possible instantiation from the analysis of figure 15 we can conclude that there is a single failure causing this violation. By negating the failure mode found, i.e. forcing it not to happen, more single failures can be found. When we get the answer Contradictory, we know that all single failures are found. We can further investigate whether each one of the failures found is also a prime implicant.

Next, we want to find combinations of two failures. This is done in the function *EQ( k , [ List of failures])* by changing *k* to 2, i.e. by forcing exactly two failure modes to happen at the same time. At the same time the conjunction of all earlier found prime implicants is negated, see figure 16.

| | |
|---|---|
| EQ ( 2, [Temp_failure_plus10, Sound_fails_off, . . . ]) & NOT PI1 & NOT PI2 & … & ( wait_t_1 # sol_t_1 # work_t_1 ) & Temp_t_1 > 320 | Satisfy |
| Climatic chamber with failure modes | Possible instantiation including 2 failures or Contradictory |

Figure 16 - Analysis to find combinations of two failures. PI1 and PI2 represent prime implicants found at earlier steps

Note in figure 16 that by excluding the prime implicants, expressed by the negated conjunction of all prime implicants in the Question, they can not appear in any combination of two failures.

Using the methodology in this way we can find single and multiple failures without explicitly building any fault tree.

**Aircraft Fuel system**

Here we report on analysis of a part of the fuel system in the Gripen aircraft, which controls the transferring of fuel between tanks. Fuel transfer is regulated by a number of factors, for example valve-states, amount of fuel left, aircraft state, etc. This exercise is interesting from two points of view. First, the automatic generation of the NP-Tools model, and second the realistic size of the application.

In this case, the system specification in natural language was first formalized using the Statemate tool (ref. 14). At the next step the Statemate model was automatically translated into a NP-Tools macro using a prototype which is under development. This translator is based on a restricted subset of the language of statecharts as implemented in Statemate. Hence, first the Statemate model had to be modified according to the restrictions. For example variables of type real had to be converted into variables of type integers and the top level of the model had to consist only of one statechart (in turn consisting of a hierarchy of statecharts). Moreover, the prototype does not handle activity charts.

The size of the analyzed system can be illustrated by the number of input, state and output variables. In this case, the controller has about 700 in-pins and about 500 out-pins. Nevertheless, the analyses performed were completed within a minute at the most.

This can be compared with size of the climatic chamber model, in another version than in this paper, which was automatically translated from statecharts. In this version there were 96 in-pins and 88 out- pins, and analysis results were obtained within seconds.

## Summary

We have presented an approach whereby functional analysis and reliability analysis is based on the same system model. The analyses are performed in three stages: first functional correctness of the design is checked in the absence of failures. Next, the functional model is augmented with failure modes. Single (as well as multiple) failures leading to a Top Event in the sense of FTA analysis can then be identified. In this manner traditional FTA analysis is extended to integrated HW/SW systems but no fault tree needs to be built.

For a particular application area, failure mode macros should be available as standard components. Hence, the extension of the model with failure modes can be partially automated. Also finding the prime implicants according to the method presented here can be automated.

Future expansions of this work are to include the quantitative part of the FTA by extending the model with probabilities for failure modes and treating other forms of FMEA analysis.

## References


1. Hansen K. M., Ravn A.P. and Stavridou V. From Safety Analysis to Software Requirements. IEEE Transactions on Software Engineering, Vol 24, No. 7, July 1998, Pp 573-584.
2. Chan W., Anderson R.J., Beame P., Burns S., Modugno F., Notkin D., and Reese J.D. Model Checking Large Software Specifications, IEEE Transactions on Software Engineering, 24: Pp 498-519, July 1998.
3. Sheeran M. and Stålmarck G. A tutorial on Stålmarck's proof procedure for propositional logic. In Proceedings of the International Conference on Formal Methods in Computer-Aided Design of Electronic Circuits (FMCAD'98), Springer-Verlag LNCS vol. 1522, 1998.
4. Prover Technology AB. Reference Manual, NP-Tools version 2.4, Stockholm, 1999.
5. Sheeran M. and Stålmarck G. Model Checking with Induction and Boolean Satisfiability. Technical report, U-99-003, Prover Technology.
6. Halbwachs N., Caspi P., Raymond P. and Pilaud D. The synchronous data flow programming language LUSTRE. In Proceedings of the IEEE. St Martin, 1991.
7. Borälv A. and Stålmarck G. Prover Technology in Railways. In Industrial-Strength Formal Methods, Academic Press, to appear.
8. Groote J., Koorn J. and van Vlijmen. The Safety Guaranteeing System at Station Hoorn-Kersenboogerd. Technical Report 121, Logic Group Preprint Series, Department of Philosophy, Utrecht University, 1994.

9. Åkerlund O. Safety Correctness and Reliability Analysis of Complex Systems using Formal Methods. Licentiate Thesis, Linköping University, LiU-Tek-Lic-1997:53.

10. Barlow R. and Prochan F. Statistical theory of reliability and life testing, probability models (2nd ed.). To Begin With, Silver Spring, 1981.

11. Quine W. (1959). A way to simplify truth functions. American Mathematical Monthly, volume 66, 1959.

12. Worrel R. Using the set of equation transformation system in fault tree analysis. In Reliability and Fault Tree Analysis, (ed. R. E. Barlow, J. B. Fussell and N. D. Singpurwalla), Philadelphia, 1975. Pp. 165-185.

13. Harel D. STATECHARTS: A Visual Formalism for Complex Systems, Science of Computer Programming, 8: Pp231-274, 1987.

14. i-Logix, Statemate Magnum Reference Manual, version 1.1.

## Biography

O. Åkerlund, Saab AB, 581 88 Linköping, Sweden, telephone - +46 13 185101, e-mail – ove.akerlund@saab.se.

Received his BA in Mathematics and Statistics in 1973 from Stockholm University. In 1998 he obtained a Licentiate degree from department of quality technology at Linköping University. He has been involved in the area of system safety and reliability both in the nuclear industry and the aerospace industry in Sweden for over 15 years. His current affiliation is with the Saab AB, Aerospace as a specialist in statistics and system safety analysis.

S. Nadjm-Tehrani, Department of Computer and Information Science at Linköping University, 581 83 Linköping, Sweden, telephone - +46 13 282411, e-mail – simin@ida.liu.se, http://www.ida.liu.se/~snt.

Is an associate professor in computer science at the Embedded Systems Laboratory. She received her PhD from Linköping University in 1994, and her current research interests are: formal specification and verification of embedded systems, hardware/software co-design, hybrid (continuous/discrete) models, and real-time system specification languages.

G. Stålmarck, Prover Technology AB, Alströmergatan 22, SE-112 47 Stockholm, Sweden, telephone - +46 8 6176800, e-mail – gunnar@prover.com, http://www.prover.com.

Received his BA degree in theoretical philosophy from Stockholm University 1982. He is the co-founder of the company Prover Technology AB who owns the registered patent for the propositional theorem prover incorporated in NP-Tools. He is also an adjunct professor within the Formal Methods research group at the Computing Science Department at Chalmers Technical University. His research interests include automatic theorem proving and formal verification of systems.