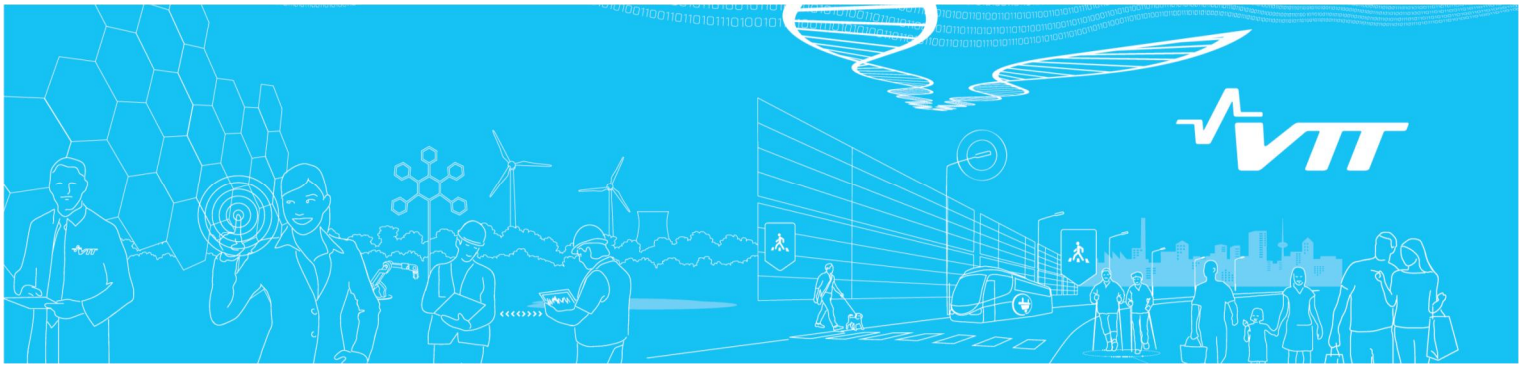
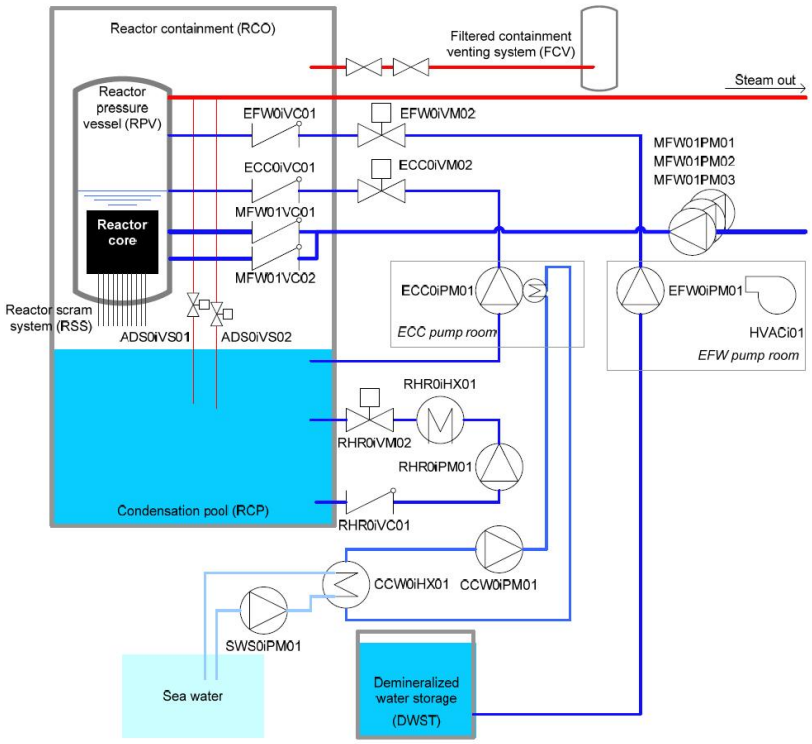


RESEARCH REPORT

VTT-R-04924-15



Feasibility study on the integration of PRA methods and model checking

Authors: Jussi Lahtinen, Kim Björkman

Confidentiality: Public

Report's title

Feasibility study on the integration of PRA methods and model checking

Customer, contact person, address	**Order reference**
Valtion ydinjätehuoltorahasto VYR	SAFIR 5/2015
Project name	**Project number/Short name**
Integrated safety assessment and justification of nuclear power plant automation	102392/SAUNA
Author(s)	**Pages**
Jussi Lahtinen, Kim Björkman	26/
Keywords	**Report identification code**
Probabilistic risk assessment, model checking, integration, nuclear, automation	VTT-R-04924-15

Summary

Digital instrumentation and control (I&C) systems play an important role in the operation of nuclear power plants (NPP). The safety and reliability analysis of such systems is challenging. We have focused on the use of two methods: model checking for verifying the correctness of I&C systems, and fault tree analysis (FTA) in the context of probabilistic risk assessment (PRA). Model checking is a formal method capable of exhaustively analysing system behaviour. Fault tree analysis is a top down approach used for failure analysis.

Neither approach can sufficiently analyse situations involving both software design errors and hardware failures. In this paper, we look for different ways to solve this issue and to integrate and couple these two methods to enable more extensive or practical safety analysis of digital I&C systems. We identify several potential integration approaches and analyse their feasibility. We also propose a concrete concept-level coupling approach, and experiment with it in practice using a small example model.

Confidentiality	Public

Espoo 7.1.2016

Written by	**Reviewed by**	**Accepted by**
Jussi Lahtinen, Research Scientist	Tero Tyrväinen, Research Scientist	Riikka Virkkunen, Head of Research Area

VTT's contact address

VTT Technical Research Centre of Finland Ltd., P.O. Box 1000, FI-02044 VTT, FINLAND

Distribution (customer and VTT)

SAFIR2018 RG1 members, electronic copy
VTT archive, 1 copy

Preface

This report has been prepared as part of the research project Integrated safety assessment and justification of nuclear power plant automation (SAUNA), which is part of the Finnish Research Programme on Nuclear Power Plant Safety 2015–2018 (SAFIR2018). This feasibility study examines potential approaches for integrating probabilistic risk assessment methods with a formal method called model checking. Some of the integration approaches are also tried out in practice using a small example model.

We wish to express our gratitude to the organizations involved in the research programme and all those who have given their valuable input in the meetings and discussions during the project.

Espoo, January 2016

Authors

Contents

1. Introduction

Instrumentation and control (I&C) systems play an important role in the operation of nuclear power plants (NPP). The nuclear industry is undergoing changes as the old analogue I&C systems are being replaced with new digitalized ones. Due to many unique features of digital systems the safety and reliability analysis of such systems can be challenging.

There are several methods used for analysing the safety and reliability of digital systems in nuclear power plants. In this paper, we focus primarily on two methods: model checking [14] and fault tree analysis (FTA) [29] in the context of probabilistic risk assessment (PRA).

The disadvantage of PRA is that it does not have a mature methodology for handling software failures (spurious failures of automation software in particular), even though research in this area is on-going, see e.g. [2, 9]. On the other hand, PRA is an efficient method for analysing hardware failures.

Model checking is a computer-aided verification method that is used to formally verify the correct functioning of a system model by examining all of its possible behaviours. In our previous work, we have created model checking methodology for modelling hardware failures that is based on the notations and conventions used in PRA models, see [20, 21]. In this model checking approach the situation is opposite to PRA: software errors, especially errors in the logic design, can be analysed in fairly large models, but the method does not scale well to large systems when hardware failures are assumed as well.

Currently, neither approach can sufficiently analyse situations in the middle ground: scenarios involving both software design errors and hardware failures. In this paper, we look for different ways to handle this middle ground, and to integrate PRA and model checking to enable more extensive or practical safety analysis of digital systems.

An ultimate goal of this work is to develop an integration approach that utilizes both of the methods and provides results that cannot be computed with either of the methods alone. In this paper, we propose one concrete concept-level coupling approach, and experiment with it in practice using a small example model.

The rest of this study is structured as follows. Model checking and fault tree analysis are introduced in section 2. Related work is presented in section 3. Section 4 discusses the use of the methods individually. Potential integration approaches of model checking and PRA are identified and discussed in section 5. A small example system is presented in section 6. In section 7, a concrete concept-level coupling approach is proposed, and the proposed technique is used for verification of the example model. Finally, section 8 concludes this study.

2. Methodologies

2.1 Model checking

Model checking [14] is a computer-aided formal verification method traditionally used for the verification of safety critical systems. In model checking, the analysed system is first modelled, and the system requirements are formalised using e.g. temporal logics such as Linear Temporal Logic (LTL) or Computation Tree Logic (CTL), see [14] and [3]. Given the model and the formalised requirements as input, a model checking algorithm automatically determines whether the system has violated its requirements. If a violating behaviour is found the model checker will give a concrete counter-example execution of the system demonstrating how a system requirement can be violated. The main benefit of model

checking is that, unlike simulation or testing, model checking tools are able to perform an exhaustive verification with respect to the formalised requirements.

In this work, we have used the model checking tool NuSMV [13], which was originally designed for synchronous digital hardware model checking. The NuSMV model checker does not have a notion of continuous time. Instead, timers and other delays have to be modelled using discrete counter variables. NuSMV employs both symbolic Binary Decision Diagram (BDD) -based [14] and propositional satisfiability solving (SAT) -based model checking algorithms.

2.2 Fault Tree Analysis

Fault tree analysis (FTA) [29] is a top down approach used for failure analysis. A fault tree is a graphical representation of undesired events that can lead to an undesired state of a system represented by a top event. In the analysis, the individual or combined lower level failures that can cause the top event are determined. The lower level events are further traversed until the basic events are reached. Generally, basic events represent some sort of component failures, for example a pump failure. The fault tree approach has been found to be particularly efficient together with event tree (ET) analysis as part of probabilistic risk assessment (PRA).

PRA is a systematic and comprehensive methodology to assess the risks of complex systems. For example, in the nuclear domain, PRA is used to continuously find ways to improve safety, justify plant modifications, optimise annual maintenance revisions and fulfil regulations. PRA integrates information from various sources such as plant design, component reliability, accidents, operating practices and human behaviour, historical operational data, and thermal hydraulic plant response.

Generally, FTA methods are based on the minimal cut set (MCS) approach. A cut set is a set of basic events that can cause the top event. An MCS is a cut set that is not a cut set anymore if one of the basic events is removed. Minimal cut sets are computed for a fault tree or accident sequences of an event tree. Numerical risk estimates are calculated based on the minimal cut sets and the probabilities of basic events.

The fault tree/event tree methodology is the most commonly used approach for PRA. However, it is recognized that other, more advanced, approaches can in certain situations be better suited for the reliability analysis of dynamic systems, such as digital I&C systems, than traditional fault tree/event tree analysis [11]. In these cases, the static fault tree approach might not be accurate enough or the results it provides are too conservative.

3. Related work

Several previous approaches where FTA is used together with model checking exist. For example, a safety assessment that combines formal verification and fault tree analysis is presented in [23]. In this work, model checking was used to prove functional correctness with respect to a formal model, whereas FTA was used to validate the model and to consider technical defects. The combined approach uncovered a safety flaw, led to a precise requirement specification for the software, and showed various ways to improve system safety.

In another paper, the model checker UPPAAL was used to validate the correctness of FTA [10]. In this study, it was possible to examine exhaustively all possible states of the system and to determine if the assumed failure modes of components were indeed correct. Model checking was able to identify a subtle and previously omitted failure mode.

Koh and Seong [17] developed an approach in which fault trees are generated from symbolic model verifier (SMV) models. The goal was to overcome problems (e.g. errors) originating from when fault trees are generated from natural language specifications and reduce the work load of safety verification. Model checking was used to verify the safety properties of the SMV model.

There is also previous work describing the utilization of model checking to analyze system faults and fault-tolerance. One example of this is the FSAP/NuSMV-SA safety assessment platform [6] that can be used for injecting faults into a system model and verifying the system fault-tolerance using model checking.

Formal verification and analysis of failures are also combined in the Altarica language. Altarica [1] was designed for formally specifying the behavior of failing systems. The language has been used e.g. in combination with model checking to assess safety requirements of the AIRBUS A320 hydraulic system [5].

Model checking has also been used to analyse the fault tolerance of a spacecraft controller, see [28]. Process algebra based modelling approaches for formalizing fault-tolerant systems have also been developed [4, 8].

Both in FTA and model checking, the approaches used to solve the respective models are quite similar. BDD-based approaches have been utilized in FTA [24, 25] and model checking [14] for quite a while. SAT-based approaches have been used in model checking but their use in FTA has been more limited. An FTA computation engine based on the ‘branch and deduce’ algorithm is presented in [26]. Branch and deduce algorithms are the core of modern SAT-solvers. The advantages of the approach are that it seems to be efficient on large PRA models, it works both on coherent and non-coherent models, and it can be parallelized to take advantage of multi-core architectures.

4. Using the methods individually

4.1 Model checking

Model checking has proven to be a useful tool for design verification in several application areas. We have previously used model checking to verifying the correctness of digital I&C system designs. In this work, we have found errors using model checking that were previously undetected by traditional testing and verification methods, see e.g. [19]. The errors typically discovered by model checking are caused by the designer of the system not taking into account (1) mistimed manual actions, (2) events outside the actual logic, such as sensor, communication or hardware failures, (3) physical events occurring in an unexpected order or (4) simultaneity of several signals.

Recently we have developed more systematic methodology for modelling hardware failures, see [20, 21]. The intention of this methodology is to provide means to perform exhaustive analyses in which both the hardware failures and the detailed functionality of the digital I&C systems are taken into account. This kind of analysis can enable e.g. a more comprehensive fault-tolerance analysis of the plant. The developed methodology is also strongly linked to PRA. A PRA model (see [2]) was used a case study, and the PRA conventions related to handling failures were closely followed when the corresponding model checking methodology was developed. The conclusion of this case study is that the approach as such does not scale to large models consisting of several many-redundant safety systems. Verification of such a plant-level model was only possible when heavy simplifications and abstractions were used. Exhaustive fault-tolerance analysis in which such simplifications are not used is limited to individual safety systems or to a single redundancy of a safety system.

The benefit of model checking when compared to FTA is that it focuses on the detailed functionality of the automation systems and can point out problematic sequences even on the level of the I&C system clock cycle. Analysis of the plant level model can also be used to find design errors in the automation logics that only manifest themselves when a certain single failure is simultaneously present.

4.2 PRA

An example nuclear power plant with reactor protection system was developed to demonstrate PRA modelling of digital I&C, see [2]. Both detected and undetected failures of hardware and software components of the I&C system were considered in the model. The hardware component failure modes included loss of function, latent loss of function and spurious function. Software failures examined in the model included: failure of all subsystems of the reactor protection system, failure of one subsystem, failure of a redundant set of I&C units, and failure of one or more application software functions.

Different fail safe principles and intelligent voting logics were incorporated in the model. Along with undetected failures, detected failures contributed significantly to the core damage frequency in all cases. Especially, spurious signals caused by detected failures had a large impact.

Common cause failures between I&C components were also studied in the model. In most cases, common cause failures of I&C components were dominant over single failures. Thus, the estimation of CCF parameters for I&C components has to be studied carefully.

The example PRA model provided considerable information on the relative importance of different I&C failure types and the quality of different safety designs and modelling options. Even if the example was rather simple, the model for digital I&C became rather complex and can be hard to review. This is not a feature of the PRA technique but it reflects the nature of I&C. One possibility could be to simplify the model by including only those failure modes which have significant contribution to the core damage frequency, e.g., common cause failures. It is however difficult judge beforehand what can be excluded from the model.

5. Coupling of model checking and PRA

Section 4 presented what the different methods by themselves have established. A more extensive or practical analysis of digital systems could be achieved by coupling these methods. In the following subsections, we identify 12 different approaches to couple model checking and PRA, and discuss their feasibility.

Many of the coupling approaches relate somehow to fault tolerance analysis. In fault tolerance analysis, a collection of analyses is performed to demonstrate that an NPP design and construction fulfils fault tolerance requirements specified in e.g. YVL guides [16]. Both model checking and PRA can be used as a part of this framework. Generally, FTA and PRA are already used in fault tolerance analysis. A more extensive utilization of model checking could also be considered.

When the methods are coupled, they support each other and, thus, improve the results of fault tolerance analysis as well. As an example, a coupled method for assessing spurious actuations is discussed in section 5.3. The methods can also provide valuable information to each other. Model checking can produce new failure modes or scenarios for PRA (section 5.5), whereas PRA can help focusing model checking to the most important components or functions (section 5.12). A coupled approach can also be used in success criteria assessment (section 5.4). A concrete practical approach for coupling PRA and model

checking is presented in section 7.1. The practical approach incorporates basically all the discussed coupling approaches related to fault tolerance analysis.

5.1 A single plant-level model

As the same data can be used as input for both approaches, it could be beneficial to only have a single plant-model that is used for both analysis methods. The plant model would contain all the information needed for building both models. The plant-level model could also be accompanied with software tools capable of generating the PRA model and the model used for model checking. The idea is illustrated in Figure 1.

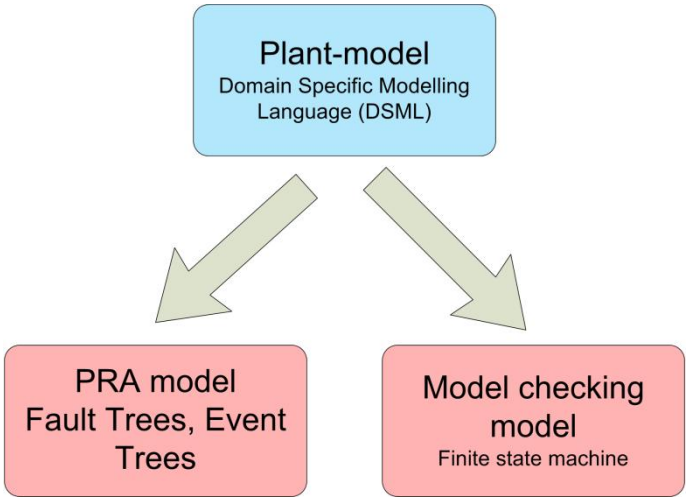


Figure 1. A single plant-model

The plant-level model could be based on a Domain Specific Modelling Language (DSML). Domain Specific Modelling Languages are languages that are specifically created for the purpose of modelling in some problem domain. In a DSML the concepts of the problem domain can be directly used. Typically the purpose of using a DSML is to allow the automatic generation of implementations based on the model. In our case we would use the plant-level model to generate other models. A DSML can be either a textual or a visual language, or a combination of both aspects.

Optimally, this approach could be implemented on top of any existing system design tool. Another more light-weight approach could be to use a e.g. an Excel file for storing all the data related to failure modes, probabilities, the system architecture, physical placement of equipment, and the allocation of I&C functions to the computers that perform the calculations. In addition to this, simple skeletons of the PRA model and the model checking model would be created. Actual models would then be generated by completing the skeleton models with the data in the Excel sheets.

The main benefit of the single plant-level model is that both analyses always use the same data, making e.g. version control problems non-existent. The approach could also decrease the amount of modelling effort needed; especially if tools are created that can automatically generate specific models from the plant-model. In the optimal case both analysis tools are immediately available once the plant model has been created.

This integration approach requires quite a lot of effort including work such as: properly defining the used DSML, implementing a modelling environment, and developing the software tools needed for automatic generation of PRA models and model checking models.

Using a single plant-model is a very valuable approach, but it could be more worthwhile to first identify practical coupling methods between PRA and model checking before it is used.

5.2 Cross-checking of analysis results

After the minimal cut sets have been calculated in PRA, this result could be verified using model checking. Two versions of this approach are possible:

1) **Verifying that the minimal cut sets have been calculated correctly by the PRA tool.** A model checking model can be built that corresponds to the fault tree. Using the model checking tool, it is possible to verify that the top event occurs when the components in the minimal cut set fail, and that the minimal cut set is such that if any of the components in the set is removed, the top event does not occur.

2) **Verifying that the fault tree is correctly built according to some higher-level description (plant-level model).** The purpose of this approach is to verify that the fault tree used for PRA has been built correctly according to a common plant model or other reference material used for building the fault tree. In this approach a model checking model of the fault tree is built according to the common plant model, and it is verified that the resulting fault tree is according to the one used for PRA.

The first approach is very feasible, but may not be worth the effort as it does not provide new results that cannot be produced by PRA alone. However, it could be used if there is doubt in the calculation method used in the PRA tool.

The second approach is only applicable if a common plant model is used. The plant-model approach would possibly be accompanied with tools used for generating the fault tree automatically, and this kind of cross-checking may not be reasonable. In this case the focus should be in testing the tool generating the fault tree instead.

The cross checking task can become computationally challenging if there are a lot of minimal cut sets (> 10000). In these cases, only the most relevant cut sets can be analysed. The presented example cross checking approaches can only be used to verify that the computed cut sets are relevant, not if there are missing cut sets.

5.3 Spurious actuations

It is not completely clear how spurious actuations should be analysed in PRA. An approach to analyse systematically and comprehensively the development of spurious actuations is needed [2]. Model checking is an efficient method to check, for instance, how a faulty input signal (e.g. faulty measurement or mistimed operator action) can propagate through the software logic and if it can cause a spurious output signal (spurious actuation). If a spurious actuation signal is possible, PRA can be used to assess how it will impact the related safety systems and the overall safety. On the other hand if a spurious actuation is not possible the event can justifiably be screened out from PRA.

The benefit of the approach of utilising model checking is that it provides a systematic way to analyse possible spurious events. Utilizing model checking can also improve the quality of the respective PRA model. A challenge can be to determine what systems or functions should be model checked. Also, it can be challenging to determine if a model checking discovery should be accounted for in PRA. For the coupled method to be applicable a systematic approach to adapt model checking results for PRA use is needed.

This approach is related to “New top events or intermediate events” approach discussed in section 5.5. The analysis of spurious events can find new scenarios that have not been accounted for in the PRA.

5.4 Success criteria assessment

In model checking, the verification of the fault-tolerance of the plant can be verified by taking into account the failure modes of the hardware components. In this kind of analysis a success criterion is defined for each postulated initiating event. These criteria are deterministic conditions under which the plant is expected to sustain the initiating event. An example of such a criterion could be that “one out of four emergency feed water pumps are started”.

Model checking could be used to analyse if an I&C system contains a design error that together with a combination of hardware failures will prevent the fulfilment of a success criterion. For assessing a success criterion the combination of hardware failures should not by themselves prevent the fulfilment of a success criterion. For example, if the success criterion is “one out of four emergency feed water pumps are started”, model checking could be used to verify that the system doesn’t contain a design error that together with failure of three pumps would prevent the fourth pump from starting. PRA could be used to compute different hardware failure combinations.

If the deterministic criteria are not fulfilled, probabilistic considerations should be used to assess the risk importance of the situation. However, more detailed and systematically defined practices for interpreting model checking results in PRA are still needed.

A benefit of the approach is that different success criteria from an I&C point of view can be assessed in a structured way. Additionally, model checking could discover errors design errors that manifest themselves when a combination of hardware failures is simultaneously present. A challenge in verifying large plant-level models using model checking is that the resulting model can become very large. The concrete practical approach for coupling PRA and model checking presented in section 7.1 is strongly linked to this coupling approach.

5.5 New top events or intermediate events

FTA is a technique for identifying the causes of a hazardous event. The completeness of a fault tree can be difficult to ensure, as unforeseen events might exist that also form a cause or a sub-cause of the top event.

Model checking is a method that can find new concrete scenarios that lead to a compromised state in the plant. If these scenarios are not yet part of the PRA model, we can use the scenario described by the model checking counter-example as a new top event or intermediate event, and modify the fault logic accordingly. This approach is a bit similar to the approach presented in [10], where the system is modelled using the input language of a model checking tool, and fault trees are automatically generated based on the model.

The identification of new scenarios using model checking could be useful in the analysis of dynamic systems, digital I&C systems being a prime example, where it can be difficult to identify complex failure scenarios, without exhaustive verification methods such as model checking. Basically, the new scenarios identified by model checking can be divided into the following categories:

- Scenarios due to hardware failures only.
- Scenarios due to software failures, or human interaction.
- Scenarios due to a combination of hardware failures, and software/human behavior

If model checking reveals a scenario that involves only hardware failures, it means that the fault tree has been incorrectly or incompletely built in the PRA model, otherwise the same scenario would have been found already in the PRA analysis.

Scenarios that are due to software failures or human interaction simply cannot be found using only fault trees, and a more detailed analysis of the system is required. Model checking can find scenarios caused by e.g. mistimed operator actions or software anomalies. If an operator action can cause a critical failure it should be also accounted for in the PRA model.

The main advantage of this approach is that the fault trees become more accurate, leading to better PRA results. The biggest challenges are that the PRA model's description of the digital I&C system can become increasingly complex, and that model checking may scale poorly to analysing large systems with dynamic features and a wide range of hardware failures. One solution is to focus the model checking analysis to a smaller set of hardware failures or to single safety functions.

We also note that if a critical scenario is found that involves a software error, the finding may lead to a change in that system. Updating the fault tree based on something that is about to change may not be that valuable. The findings, however, could be beneficial in estimating the probability of design errors in other similar systems, for instance, designed by the same vendor.

5.6 Computation of the Minimal Cut Set

Solving large scale PRA models is computationally a challenging task. To improve the computational efficiency, the use model checkers or algorithms employed by model checkers for MCS computation or quantification have been proposed (e.g. [6, 22]). A model checking algorithm can be used to produce a MCS as a counter-example. Most of the available model checking engines, however, cannot directly be used for computing minimal cut sets, as they produce only a single counter-example at a time. They should be modified to produce all counter-examples and not just one. The counter-examples also do not necessarily represent a minimal combination of failures. Thus, model checkers need to be modified further to collect a minimal set of failures. Alternatively, some iterative cut set minimization approach could be possible.

Algorithms and data structures used in model checking have also been utilized in fault tree computations. For example, BDDs have been used in the quantification of a top event of a MCS list [27]. They are useful especially in cases where the results include several events with high probability or where the success paths in the event tree sequences impacts the results.

Since BDDs have some scalability issues, more advanced approaches used in model checking such as SAT-based solutions could prove to be more efficient, see e.g. [21]. A possible simple SAT based approach:

- Map the computation problem into a SAT-problem
- Find a solution (a MCS)
- Update the problem with the solution so that the same solution is not found anew
- Solve the problem as many times as MCSs are found or the length of the MCSs reaches a threshold length

However, it may be challenging to develop minimal cut set search algorithms based on model checkers that would be more efficient than the currently employed solutions.

Instead of developing completely new algorithms, it could be more fruitful to study if some parts of current MCS algorithms could be replaced or complemented by approaches used in model checking. For example, if the fault tree structure could be optimized with model checking methods.

The benefits of utilizing model checking related algorithms in MCS computation is that computational efficiency could be improved. Additionally, the algorithms are constantly being improved or new more advanced algorithms are being developed, e.g. PDR model checking algorithm [7]. However, developing model checking based algorithms for MCS computation can be quite a laborious task.

5.7 Software reliability assessment

An analytical assessment method for the analysis of software reliability is suggested in [9]. In this approach, it is assumed that the number of faults in software is mainly dependent on two metrics:

- the complexity of the software, and
- the level of verification and validation (V&V) applied on the system.

The complexity assessment is based on analysing function logic diagrams depicting the system design. It is assumed that systems based on complex design diagrams are more likely to fail than systems based on simple design diagrams if the level of V&V applied on both systems is approximately the same. It is also assumed that a piece of software that has been rigorously verified is more reliable than software whose verification has been negligible if the complexity of the two systems is approximately the same.

To benefit software reliability assessment, model checking could be used in the analysis of the complexity of the piece of a function block diagram. For example, the NuSMV model checker can compute the state space of the model, which could be used directly as a complexity measure. A challenge is that the state space is computed for the model and not for the system itself and, thus, the state space depends on the used modelling methodology. A way to at least partly overcome the challenge would be to generate the model checking model directly from the plant model. This way the state space would depend less on subjective modelling approaches.

Besides supporting the determination of complexity, model checking can be used as a verification method improving the level of V&V applied on the system. In software reliability context, this could mean that if a piece of application software was verified using model checking it would have a positive influence on the failure probability of the software (assuming that no errors were found).

Software reliability assessment in the context of PRA is also part of another task in SAUNA (MODIG).

5.8 Verification of PRA actuation logic

The logic in PRA models is typically described on a higher abstraction level compared to model checking models. One possibility of coupling model checking and PRA could be to compare PRA actuation logic against the actuation logic implemented in the model checking model.

One way of verifying the PRA actuation logic is to see whether the minimal cut sets make sense when a more detailed model of the actuation logic is used instead. Using the more detailed model checking model it could in theory be verified whether the top events always occur when the components in the minimal cut set fail; and whether the minimal cut set is

such that if any of the components in the set are removed and all other hardware components work correctly, the top event does not occur.

The problem in this approach is that actuation logic used in PRA is very seldom completely correct as the actual logic used in the real automation system likely uses many dynamic features that cannot be described using a simple propositional formula. Consequently, all attempts to simplify the logic are deemed to be wrong, and it is unclear how the simplified actuation logic should be compared with the more detailed one.

This kind of coupling could, however, be useful, e.g., if a safety I&C system contains smart voting logics for actuation signals to the actuators. Modelling of the actuation logic correctly can be somewhat laborious in fault trees especially if the actuation logics of different functions account for degraded voting logic differently. In such cases model checking could be used to verify that the actuation logic is correctly accounted for in all different cases.

In conclusion, a full-scale verification of the PRA actuation logic is probably not practical. Instead, focusing the verification on certain important parts of the logic that do not involve time-related behaviour (such as smart voting logics) is a more feasible approach. Verifying a small part of the fault tree also requires quite a little effort. However, it should be noted that since the verification of a voting logic is a relatively small task, other verification methods besides model checking can also be used. Simple verification methods such as brute force exhaustive testing may be more suitable for such small scale verification tasks.

5.9 Probabilistic model checkers

There are model checking tools that incorporate probabilities in the models. Tools such as PRISM (see e.g. [17]) can analyse many kinds of models including discrete-time Markov chains, continuous-time Markov chains (CTMC), Markov decision processes (MDP) and probabilistic extensions of timed automata. The PRISM model checker can be used to analyse many quantitative properties such as performance and reliability, for example: “the probability of a shutdown occurring within 1.0s is 0.99”. Probabilistic model checking is even more computationally challenging than traditional model checking, and thus does not scale well to large systems. Nevertheless, these probabilistic model checkers could also be used to provide supplementary information to the PRA analyses. The following approaches are identified:

- Probabilistic model checkers could be useful in the analysis of repairable systems, i.e. systems containing repair dependencies, to produce more accurate risk probability estimates.

- Probabilistic model checkers could be used to quickly quantify the total risk for different realization of the same risk scenario, in order to quickly identify the best and the worst cases as well as the intermediate ones [12]. In other words, probabilistic model checking could operate as a sensitivity analysis tool.

- Probabilistic model checking also allows the calculation of the effectiveness of different actions to reduce risk, see also [12]. For example, it could be possible to analyse how the risk of a certain scenario changes when a particular subsystem is activated / disabled.

Many existing PRA tools already implement sensitivity analyses, and are able to calculate risk importance metrics. These PRA tool features seem to correspond to the information produced by the second and the third item in the above list.

5.10 Model checking extension to dynamic features of PRA

Fault tree/event tree analysis is generally used for performing state-of-the-art PRA. However, it is recognized that other, more advanced methods, may be better suited for reliability analysis of dynamic systems such as digital I&C. The FinPSA tool [15] includes an alternative approach to model I&C systems, in which I&C systems are modelled using communication equations written in a text file. In communication equations of the I&C model, the dependencies of a system are represented using success logic instead of failure logic. The I&C model is linked to fault trees and it is transformed into fault trees when minimal cut sets are generated. Benefits of the I&C model are the compact and simple representation and ease of making modifications.

One possible way to improve the modelling of digital systems is to extend the I&C model feature of FinPSA to include a description of dynamics (e.g. fault propagation, spurious function activation, modelling of priority logic and smart voting logics, and timing issues). The extension in its simplest form could mean that a new operator/gate type is developed for smart voting logics that can automatically account for the degraded voting logic in presence of failures. Since model checking can be used to model the behaviour of digital I&C systems and can be used to verify of the logical designs exhaustively, it is tempting to try to utilize features of model checking in the modelling of such systems in PRA. This could for example mean that the same modelling language used in a model checking tool is also taken in use in PRA modelling. This could be relevant, e.g. in modelling more complex logics, such as smart voting logic.

This kind of extension could improve the capability of PRA to analyse dynamic behaviour. However, developing such an extension can be a laborious task. It would require both theoretical method development and tool development. Additionally, no concrete way to utilize model checking in extending PRA modelling language was identified.

5.11 Automatic generation of model checking models

A PRA model of a plant contains a lot of information that is also needed for building a model used for model checking. This is especially the case when model checking analysis is extended to hardware failures as well. If a PRA model already exists, it could be possible to generate certain parts of the model checking model automatically, instead of manually remodelling the plant. The failure modes (including common cause failures) of the different hardware components could be automatically translated, but the consequences of each failure mode would have to be thought over manually. In addition, the fault trees of the PRA model could be used for constructing the success criteria used in the model checking model. Initiating events and related event trees also carry information that could be used in the model checking model.

In practice, this automatic generation could work by first extracting information out of the PRA model. In the FinPSA tool, most of the information can be extracted in a tabular format. After this, a software tool would have to be built that translates the tabular data into some preliminary NuSMV model code. The NuSMV code would then have to be manually finalised.

This approach would probably only have minor benefits. The problem is that the PRA model does not include an explicit description of the physical system architecture and hardware configuration that would be essential for many of the features needed in the model checking model. A more comprehensive approach would be to use a common plant model as described in section 5.1 that would enable the all-inclusive generation of the model checking model.

5.12 Focusing model checking effort to most critical components

The rationale behind this approach is that all hardware failures may not be equally important to the analysis. It could be useful to leave out the some of the insignificant hardware failures such as the failures of individual wires.

As a solution, it could be possible to extract the most critical hardware components and failures modes from the PRA model and restrict the focus of model checking to only these failures to improve the performance of the verification. The selection of the failures could be based on risk importance measures, or on minimal cut sets.

For example, if the smallest minimal cut sets (e.g. of length 1-3) are extracted from the PRA model, model checking could focus only on failures that appear in these sets, and completely exclude all other failures.

Another approach would be to focus on common cause failures, as common cause failures are typically the most critical failures. Model checking could be used to analyse the behaviour of the system in a common cause failure scenario more accurately, possible revealing dynamic interaction in the system that has not been accounted for in the PRA model. Model checking could also be used to analyse the significance of the timings of the failures. This kind of information could also be useful in dynamic PRA approaches.

In practice, once the most relevant hardware failures have been selected, they are manually incorporated in the model checking model. Furthermore, the model checking performance could be improved by only postulating 1-2 hardware failures at a time, so that all other failures are disables in the model. In the NuSMV model checking tool, this could be implemented using `INVAR` declarations. `INVAR` declarations are used to specify a set of states that has to be invariant in the model. A declaration of the form "`INVAR ! failure1`" could be added to the model for all failures that are not included in the analysis. The clause restricts the model to behaviours where the failure does not occur.

If the above practice is insufficient, a method based on program slicing could be developed that completely removes parts of the model (e.g. hardware failures that are not currently examined) on the fly, and creates a simpler copy of the model that is used. As several failure combinations are examined, also several versions of the original model are needed, and the versions are analysed separately. The modelling methodology for hardware failures is quite modular, and such a tool could be based on replacing some of the models modules with trivial dummy modules. This approach would also require the use of a dependency graph of the model, in order to determine the modules that depend on a certain hardware failure. Under these premises, this program slicing method could be implemented as a fully automatic software tool.

Focusing only on certain hardware failures could increase the scalability of model checking considerably. The disadvantage of the approach is that the analysis does not cover all hardware failure combinations. After all, it is possible that the I&C system contains a software error that only manifests itself during some otherwise minor hardware failure scenario. In this approach, it is assumed that critical scenarios are more likely to be found involving failure modes that have a higher risk importance.

An alternative way of focusing model checking effort could be to use PRA to first determine which safety functions are most critical, and then use model checking to verify the control logic of these safety functions.

6. Example system description

In what follows, we first introduce a small example model, and then try out some of the integration approaches in practice using the model. The example system is a simple feedwater tank system (Figure 2). In the feedwater system there is a constant flow to the water tank. The water level in the water tank is controlled by the control valve. The control valve is operated based on the values of the water level measurement. The objective of the system is to keep the water level between a high limit value and a low limit value.

When the high level limit is reached, the water level measurement device sends an open command to the control valve, causing the tank to discharge. When the low level limit is reached, the water level measurement device sends a close command to the control valve. Undesirable situations are too high level of the tank (overfilling or overpressurisation) and too low level of the tank. This may happen due to failure of the components.

Besides the control valve the system contains two identical water level sensors, two acquisition and processing units (APU), and a single voting unit (VU) for controlling the valve. Figure 3 illustrates the control logic and the division of functions between the different units.

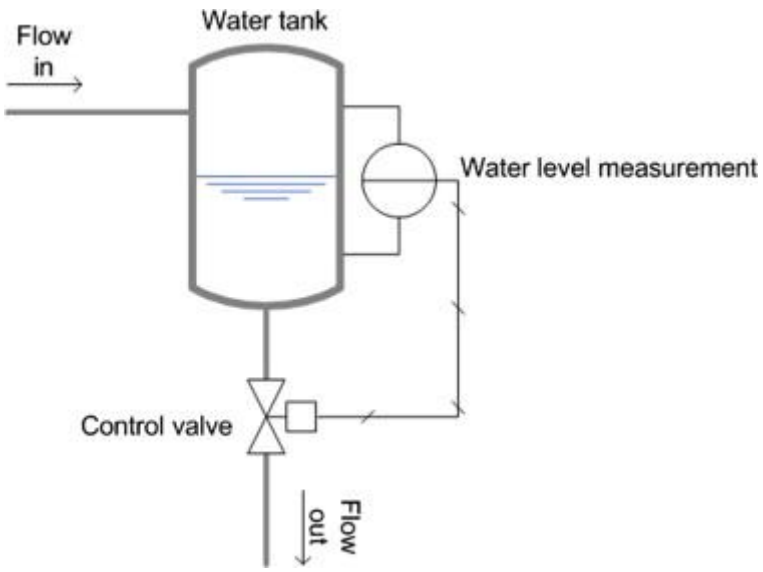


Figure 2. An Example feedwater tank system

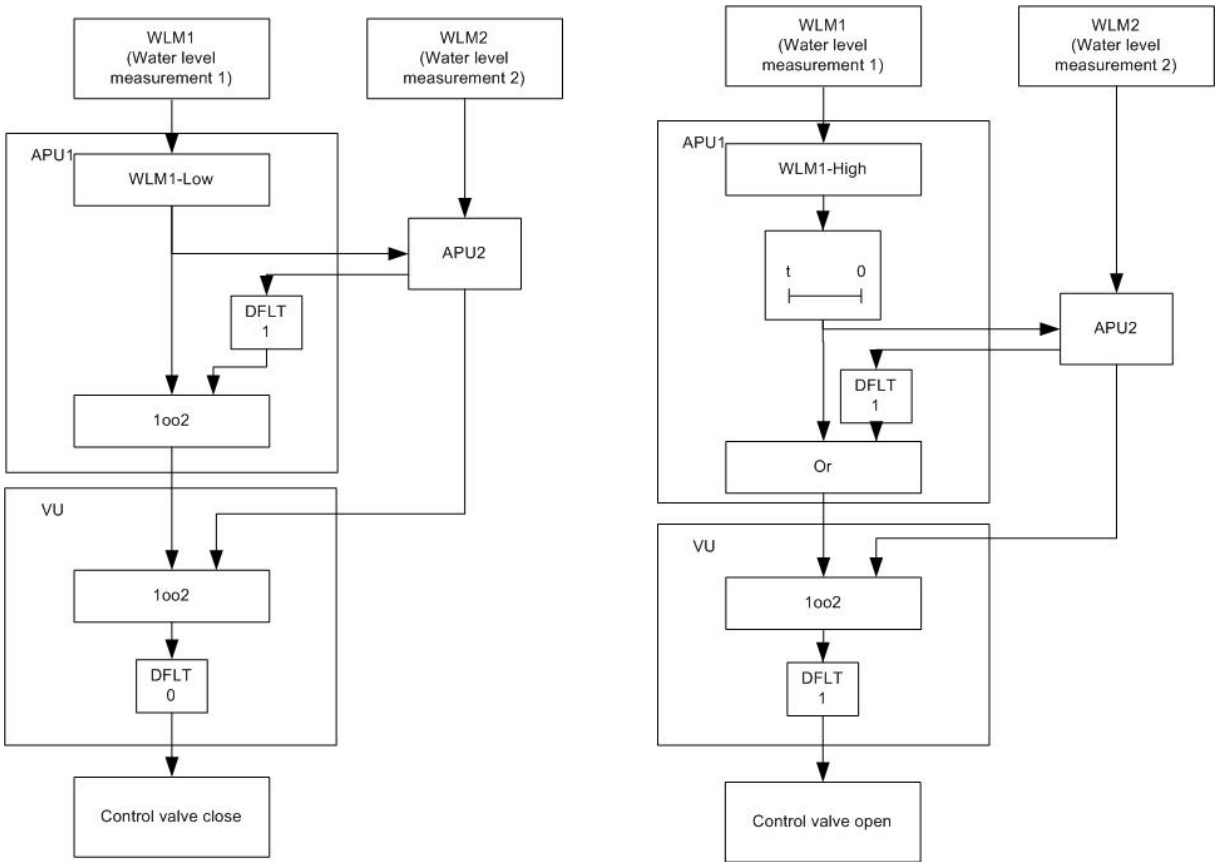


Figure 3. The control logic of the example system

7. Applying a coupled approach on the example model

In this section we first introduce a concrete practical approach for coupling PRA and model checking that is based on the ideas discussed in section 5. Then we apply the approach on the example system, and analyse the results.

7.1 Coupled use for detecting combinations of software and hardware failures

The general integration approach is illustrated in Figure 4. The approach is based on:

- restricting the model checking analysis to a smaller set of hardware failures, making the model checking phase more scalable to large models, and
- the verification of a single hardware failure combination at a time.

PRA is first used to calculate minimal cut sets of the system, and only failures in the shortest minimal cut sets (shorter than some predetermined length) are included in the model checking model. This approach can significantly reduce the size of the model.

To further limit the scope of the analysis, minimal cut sets can be computed for a single safety system, and assuming a single initiating event only. In this case, the minimal cut sets basically correspond to failure combinations that cause the failure of the success criterion given that initiating event. These initiating events and related success criteria are then incorporated in the model checking model.

In the verification phase, we examine subsets of these short minimal cut sets one by one, and analyse the fulfilment of the success criteria exhaustively in each case. The subsets of

the shortest minimal cut sets are interesting because basic events in short minimal cut sets are essentially more critical than basic events in longer minimal cut sets. We basically investigate whether software design errors occurring together with these more critical hardware failures could also lead to the failure of the system. Model checking can identify these previously unknown scenarios if they exist. Finally, if such a scenario is found using model checking, the risk importance of the scenario can be analysed using PRA.

The proposed approach can be used for:

- Identifying spurious failures manifested during hardware failures,
- Verifying the fault-tolerance of a system and identifying scenarios where the success criteria are not fulfilled, and
- Identifying new scenarios, in which a safety function of the plant is not achieved.

An alternative way of selecting the hardware failures on which model checking focuses on could be to rank component failures according to their risk importance and focus only on the most important failures.

The proposed approach is quite attractive, as it may extend safety assessment to an area that the individual approaches themselves cannot reach. When this approach is used on larger scale, an automatic solution for focusing the verification on a given failure combination may be needed, especially if the number of failure combinations becomes large. On the example model we experiment with the approach manually.

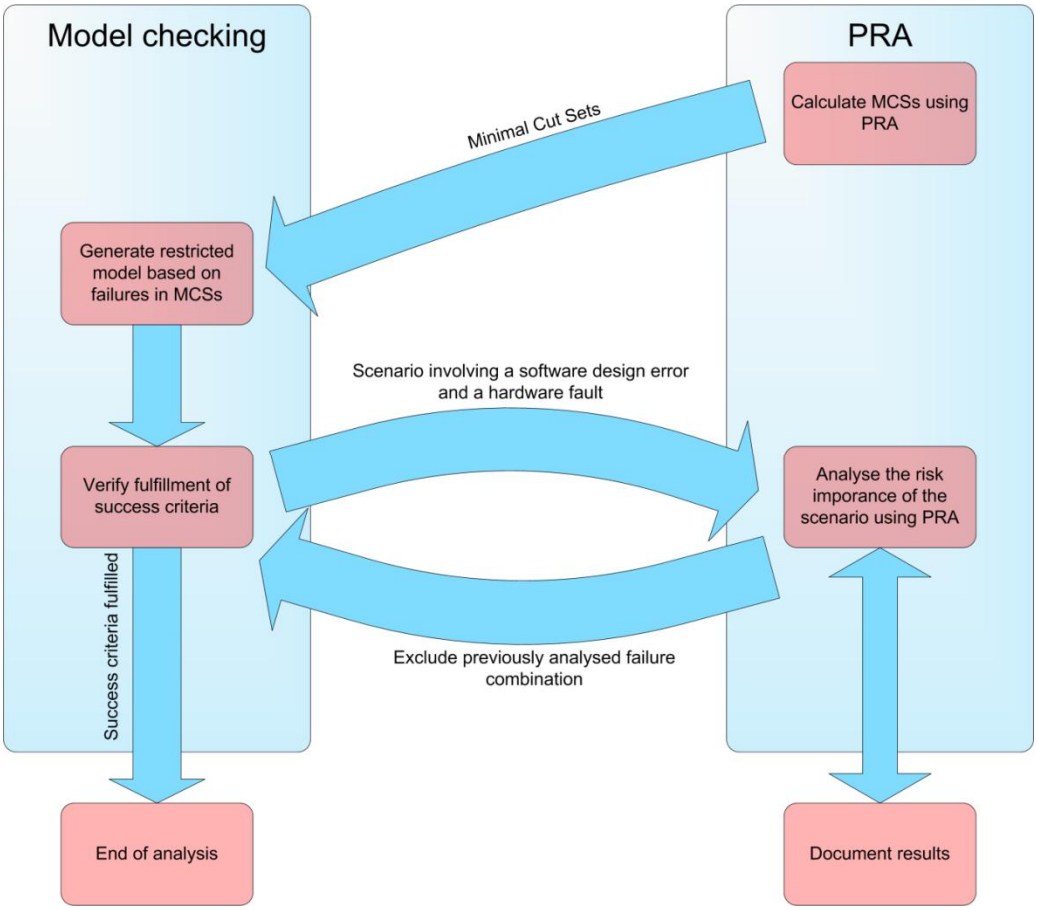


Figure 4. Coupled use of model checking and PRA for detecting scenarios involving both a software error and a hardware error

7.2 Application of the coupled approach on the example model

In order to try out the coupling approach in practice, two models of the example system were first created: a PRA model, and a corresponding model checking model based on the methodology for modelling hardware failures, see [20, 21]. The postulated failure modes were the same in both models. The environment model in the model checking model was very simple, allowing the water level to change linearly based on input water flow. The effect of the valve position to the water level was not implemented in the model. Instead, an over-approximation is used in which the water level may change more freely. Each I&C signal was accompanied with fault status information, so that detected failure modes could be marked as faulty in the model.

The minimal cut sets of the example system that consist of only one or two component failures are listed in Table 2 in the Appendix. In this demonstration, we focus solely on minimal cut sets that have two elements, and check whether software design errors exist such that only one of the failures in the cut set can cause the top event. Alternatively, we could have modelled the failures in the minimal cut sets of length one as well. In this case we would have verified the assumption that this failure can cause the top event, by checking that the model checker can produce a concrete scenario in which this happens. This kind of verification can be used to make sure that the PRA model has been built correctly. We also could have selected the modelled failures based on risk importance measures such as the risk increase factors in Table 3 in the Appendix, or simply based on failure probabilities of basic events.

For model checking, another model was then built in which only failures in minimal cut sets of length two were modelled. Only four different components and six different failures needed to be included into this restricted model (compared to 17 components and 35 failures in the full model).

Table 1. Comparison of full model and restricted model used in model checking

	Full model	**Restricted model**
Full state space	6.1×10^{16} reachable states out of 2.8×10^{20}	1.6×10^{10} reachable states out of 1.4×10^{13}
State space with restriction to a particular single failure	2.2×10^{7} reachable states out of 2.8×10^{20}	2.2×10^{7} reachable states out of 1.4×10^{13}
Verification time (assuming a particular single failures)	0.27 s	0.13 s

The property that was verified on the restricted model was: “Whenever the water level becomes low, the valve is eventually closed”. Formally the property can be written in temporal logic as:

```
LTLSPEC G(processmodule.Water_level < LIMIT -> F (VALVE_CLOSED))
```

Six separate verification runs were performed. Each time a different component failure was assumed in the model.

The verification time of the restricted model when compared to the full model was shorter, see Table 1, but this was not noticeable in practice since both verification times were so short (0.27s versus 0.13s). This is mainly because the example system is so simple that both verification problems are quite trivial. In this case, comparing the size of the state spaces of the models can be more meaningful. We can see that the size of the restricted model (1.6×10^{10} reachable states out of 1.4×10^{13}) is several orders of magnitude smaller than the full model (6.1×10^{16} reachable states out of 2.8×10^{20}). On larger models this difference in the size of the state spaces can potentially be a lot more significant. From the table, we can also see that if the behaviour of both models is restricted so that only a single hardware failure is possible, the reachable state spaces of both models are the same, as is expected.

In this simple example system, no new scenarios involving software design errors were found. During modelling, however, we noticed that model checking can be an effective way for noticing omissions in the PRA model. When a component failure is modelled as a part of the model checking model, the modeller must make an interpretation on the practical effect of that failure mode on actual signal values. If the effect on signals is not self-evident, it is typically safer to model the failure so that the signals involved can get any value during the failure. This can be a much broader assumption than what has been made in the PRA model, and may lead to differing results in the verification phase. Analysing the differences in the results can lead to observations of failure modes not included in the PRA model due to mistakes made in modelling or due to a very low probability of that type of a failure. In the model checking model, the undetected failure of the APU processor was initially interpreted so that the output signals of the APU can have arbitrary values. This assumption enables a scenario in which this failure in the APU processor causes the top event. Such a scenario was detected when the modelled system was analysed using model checking. In the PRA model, however, the APU processor failure is not able to cause the top event. After short analysis of the conflict, it was discovered that the APU processor failure assumed in PRA corresponded to a situation in which the computer system becomes inoperable and spurious commands are not output. The failure mode in which a processor error causes arbitrary spurious commands was not included in the PRA model due to its low probability.

If model checking would find a failure not accounted for in the PRA, the risk importance of the error should be computed. The approach presented in section 5.4 can be used. To get some conception of the risk importance of the error, the error can be modelled simply as a basic event. Adding the basic event to the appropriate place in the fault tree might not be a trivial task. Additionally, what reliability data should be used for the basic event?

A possible option to account for the discovered error is to add the subset of the minimal cut set (the subset that was required with the design error to cause the top event) and the error under an And-gate and to connect the And-gate through an Or-gate to the top event. If the design error manifests itself always when the failure combination of the minimal cut set is present, the failure probability of the error can simply be set to 1 or the status of the basic event representing the error can be set to ‘failed’. However, if the error requires some additional conditions to manifest itself, such as a mistimed operator action, more advanced techniques are required to compute the failure probability.

For example, if we assume that an error in the software always occurs when the measurement device WLM-1 fails undetectably, the original fault tree (Figure 5) would be complemented with a sub-tree accounting for the design error (Figure 6).

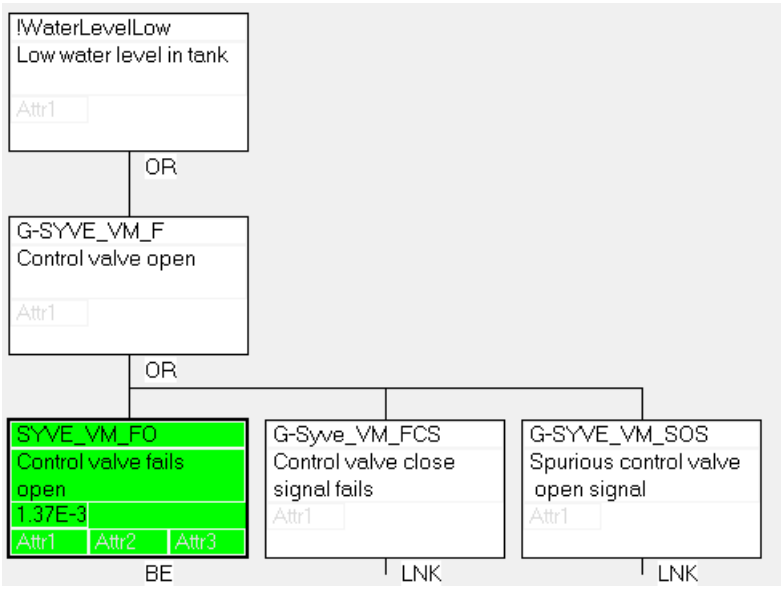


Figure 5. Master fault tree for top event low water level in tank.

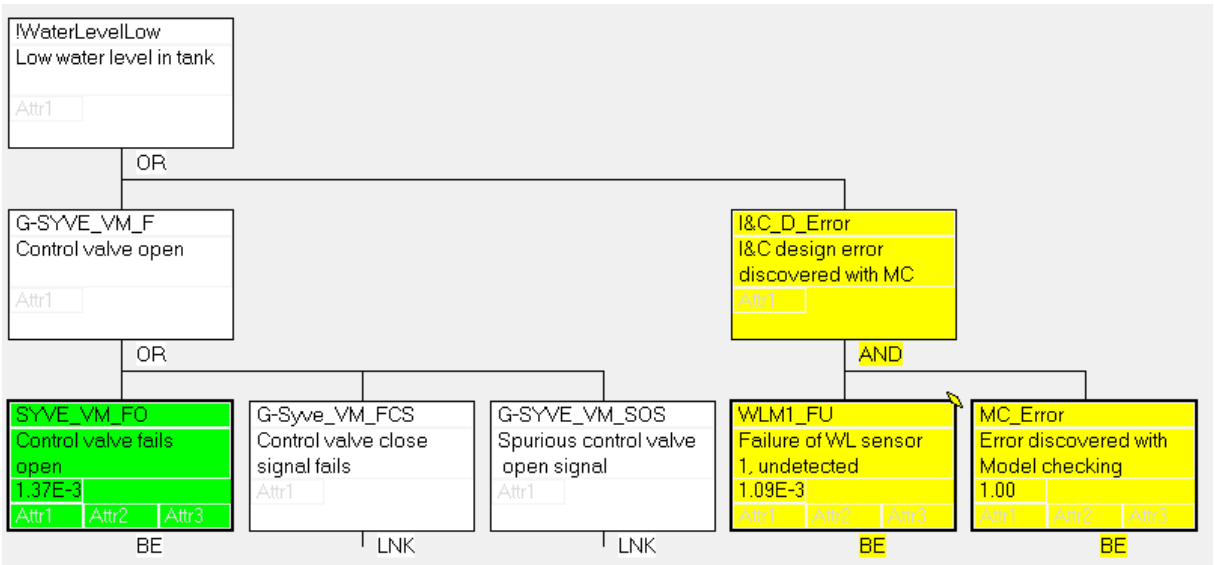


Figure 6. Master fault tree for top event low water level in tank including design error.

8. Conclusions

Model checking and PRA focus on different aspects of plant safety. The models used by the methods are, however, based on the same input information, e.g., the system architecture, hardware configuration, automation logic design documentation, and the failure modes and probabilities of the different hardware and software components. This close proximity suggests that some synergies between the methods could be found.

In this paper, we have discussed several potential integration approaches, and analysed their feasibility in practice. We have also demonstrated the use of some of the approaches on a small example model. A practical coupling technique was proposed that achieves a more extensive safety analysis. In the proposed technique model checking effort is focused to most critical components only, and new events are created to the fault trees based on scenarios found using model checking.

In future, we plan on developing the proposed technique further, and experimenting on larger system models. To further extend the fault tolerance analysis related approaches, the aim is

also to follow the progress made in another task in the SAUNA project (MODIG) regarding the evaluation of defence-in-depth of I&C design using PRA. The evaluation of defence-in-depth of I&C design can bring forth additional promising coupling methods that have not yet been accounted for.

References

1. A. ARNOLD, G. POINT, A. GRIFFAULT, A. RAUZY, “The AltaRica Formalism for describing Concurrent Systems”, Fundam. Inf. 40 (2, 3), 109 (August 1999).

2. S. AUTHÉN, J.-E. HOLMBERG, T. TYRVÄINEN, L. ZAMANI, Guidelines for reliability analysis of digital systems in PSA context - Final Report, NKS-330, Nordic nuclear safety research (NKS), Roskilde (2015).

3. C. BAIER, J. P. KATOEN, Principles of Model Checking, MIT Press (2008).

4. C. BERNARDESCHI, A. FANTECHI, S. GNESI, “Model Checking Fault Tolerant Systems”, Softw. Test., Verif. Reliab, 12, 4, 251 (2002).

5. P. BIEBER, C. CASTEL, C. SEGUIN, “Combination of Fault Tree Analysis and Model Checking for Safety Assessment of Complex System,” Proc. 4th European Dependable Computing Conference, volume 2485 of LNCS, Springer-Verlag (2002).

6. M. BOZZANO, A. VILLAFIORITA, “The FSAP/NuSMV-SA Safety Analysis Platform”, International Journal on Software Tools for Technology Transfer 9, 1, 5 (2007).

7. A. R. BRADLEY, “Sat-based Model Checking without unrolling”, In Jhala, R., Schmidt, D.A., eds.: VMCAI. Volume 6538 of Lecture Notes in Computer Science, 70-87, Springer (2011).

8. G. BRUNS, I. SUTHERLAND, “Model Checking and Fault Tolerance”, In Johnson, M., ed.: Algebraic Methodology and Software Technology, Volume 1349 of Lecture Notes in Computer Science, 45-59, Springer Berlin Heidelberg (1997).

9. O. BÄCKSTRÖM, J.-E. HOLMBERG, M. JOCKENHÖVEL-BARTTFELD, M. PORTHIN, A. TAURINES, T. TYRVÄINEN, “Software reliability analysis for PSA: failure mode and data analysis,” NKS-341, Nordic nuclear safety research (NKS), Roskilde, 2015

10. S. CHA, H. SON, J. YOO, E. JEE and P. H. SEONG, “Systematic Evaluation of Fault Trees using Real-Time Model Checker UPPAAL,” Reliability Engineering & System Safety, 82, 1, 11 (2003).

11. T.L. CHU, M. YUE, W. POSTMA, “A summary of Taxonomies of Digital System Failure Modes provided by the DigRel Task Group, Proc. 11th International Probabilistic Safety Assessment and Management Conference & The Annual European Safety and Reliability Conference, Helsinki, Finland 25-29 June 2012, paper 10-Th4-4 (2012).

12. G. CICOTTI, and A. CORONATO, "Towards a Probabilistic Model Checking-based Approach for Medical Device Risk Assessment," Medical Measurements and Applications (MeMeA), 2015 IEEE International Symposium on. IEEE, (2015).

13. A. CIMATTI, E. M. CLARKE, E. GIUNCHIGLIA, F. GIUNCHIGLIA, M. PISTORE, M. ROVERI, et al. NuSMV 2: An Opensource Tool for Symbolic Model Checking, In: E. Brinksma, K. G. Larsen editors. CAV; Vol. 2404 of Lecture Notes in Computer Science, p. 359-64, Springer.

14. E. M. CLARKE, Jr., O. GRUMBERG, D. A. PELED, Model Checking, The MIT Press (1999).

15. FinPSA -tool for Professional living PSA (2014). http://www.vtt.fi/files/research/ism/hms/FinPSA_tool_for_professional_living_PSA.pdf

16. P. HUMALAJOKI and I. NIEMELÄ, “NPP Failure Tolerance Analyses,” Nordic PSA Castle Meeting 2015, 8-10 September, Porvoo, 2015.

17. K. Y. KOH and P. H. SEONG, “SMV Model-Based Safety Analysis of Software Requirements,” Reliability Engineering & System Safety, 94, 2, 320 (2009).

18. M. KWIATKOWSKA, G. NORMAN, and D. PARKER. "PRISM: Probabilistic Symbolic Model Checker." Computer performance evaluation: modelling techniques and tools. Springer Berlin Heidelberg, 200-204, (2002).

19. J. LAHTINEN, J. VALKONEN, K. BJÖRKMAN, J. FRITS, I. NIEMELÄ, K. HELJANKO, “Model Checking of Safety-Critical Software in the Nuclear Engineering Domain,”, Reliability Engineering and System Safety 105, 0, 104 (2012).

20. J. LAHTINEN, “Verification of Fault-Tolerant System Architectures using Model Checking”, In Proc. 1st International Workshop on DEvelopment, Verification and VAlidation of cRiTical Systems (DEVVARTS), Florence, September 8th, 2014, Volume 8696 of Lecture Notes in Computer Science, 195-206, Springer (2014).

21. J. LAHTINEN, “Hardware Failure Modelling Methodology for Model Checking”, Research report: VTT-R-00213-14, VTT Technical Research Centre of Finland (2014) Available online: http://www.vtt._/inf/julkaisut/muut/2014/VTT-R-00213-14.pdf.

22. V. MANQUINHO, A. OLIVEIRA, J. MARQUES-SILVA, “Models and algorithms for computing minimum-size prime implicants,” In: Proceedings of the International Workshop on Boolean Problems. (1998).

23. F. ORTMEIER, G. SCHELLHORN, A. THUMS, W. REIF, B. HERING and H. TRAPPSCHUH, “Safety Analysis of the Height Control System for the Elbtunnel,” Reliability Engineering & System Safety, 81, 3, 259 (2003).

24. A. RAUZY and Y. DUTUIT, “Exact and Truncated Computations of Prime Implicants of Coherent and Non-coherent Fault Trees within Aralia,” Reliability Engineering & System Safety, 58, 127 (1997).

25. A. RAUZY, “Mathematical Foundation of Minimal Cutsets,” IEEE Transactions on Reliability, 50, 389 (2001).

26. A. RAUZY, “Anatomy of an Efficient Fault Tree Assessment Engine,” Proc. 11th International Probabilistic Safety Assessment and Management Conference & The Annual European Safety and Reliability Conference Helsinki, Finland 25–29 June 2012, Paper 16B-We4-2, (2012).

27. RiskSpectrum Magazine, 2014, Lloyd’s Register Consulting (2014). http://www.riskspectrum.com/upload/RiskSpectrum%20Magazine/RiskSpectrumMagazine2014_small-1.pdf

28. F. SCHNEIDER, S. M. EASTERBROOK, J. R. CALLAHAN, G. J. HOLZMANN, “Validating Requirements for Fault Tolerant Systems using Model Checking,” ICRE, IEEE Computer Society, 4-13 (1998).

29. W. E. VESELY, F. F. GOLDBERG, N. H. ROBERTS, D. F. HAASL, Fault Tree Handbook. NUREG-0492. U.S. Nuclear Regulatory Commission (1981).

Appendix 1. PRA results

Table 2. Minimal cut sets of the example system (Top event low water level)

#	Freq.	Prob	Name	Comment
1	1.37E-03	1.37E-03	SYVE_VM_FO	Control valve fails open
2	2.40E-04	2.40E-04	APU2_SR_FD	Failure of processor subrack, detected
3	2.40E-04	2.40E-04	APU1_SR_FD	Faliure of processor subrack, detected
4	2.40E-04	2.40E-04	VU_SR_FD	Failure of processor subrack, Detected
5	1.33E-04	1.33E-04	VU_OM_FU	Failure of output module, undetected
6	1.09E-04	1.09E-04	WLM_U-12	2x CCF
7	9.62E-05	9.62E-05	VU_OM_FD	Failure of output module, detected
8	8.76E-05	8.76E-05	VU_PM_FU	Failure of processor, undetected
9	4.75E-05	4.75E-05	APU2_PM_FD	Failure of processor, detected
10	4.75E-05	4.75E-05	APU1_PM_FD	Failure of APU1 processor, detected
11	4.75E-05	4.75E-05	VU_PM_FD	Failure of processor, detected
12	3.53E-05	3.53E-05	APU2_IM_FD	Failure of APU2 input module, detected
13	3.53E-05	3.53E-05	APU1_IM_FD	Failure of APU 1 input module, detected
14	1.80E-05	1.80E-05	WLM2_FD	Failure of WL sensor 2, detected
15	1.80E-05	1.80E-05	WLM1_FD	Failure of WL sensor 1, detected
16	1.18E-05	3.43E-03	APU1_IM_FU	Failure of APU 1 input module, undetected
		3.43E-03	APU2_IM_FU	Failure of APU2 input module, undetected
17	3.75E-06	3.43E-03	APU2_IM_FU	Failure of APU2 input module, undetected
		1.09E-03	WLM1_FU	Failure of WL sensor 1, undetected
18	3.75E-06	3.43E-03	APU1_IM_FU	Failure of APU 1 input module, undetected
		1.09E-03	WLM2_FU	Failure of WL sensor 2, undetected

19	1.80E-06	1.80E-06	WLM_D-12	2x CCF
20	1.20E-06	1.09E-03	WLM1_FU	Failure of WL sensor 1, undetected
		1.09E-03	WLM2_FU	Failure of WL sensor 2, undetected
21	3.00E-07	8.76E-05	APU1_PM_FU	Failure of APU1 processor, undetected
		3.43E-03	APU2_IM_FU	Failure of APU2 input module, undetected
22	3.00E-07	3.43E-03	APU1_IM_FU	Failure of APU 1 input module, undetected
		8.76E-05	APU2_PM_FU	Failure of APU2 processor, undetected
23	9.58E-08	8.76E-05	APU1_PM_FU	Failure of APU1 processor, undetected
		1.09E-03	WLM2_FU	Failure of WL sensor 2, undetected
24	9.58E-08	8.76E-05	APU2_PM_FU	Failure of APU2 processor, undetected
		1.09E-03	WLM1_FU	Failure of WL sensor 1, undetected
25	7.67E-09	8.76E-05	APU1_PM_FU	Failure of APU1 processor, undetected
		8.76E-05	APU2_PM_FU	Failure of APU2 processor, undetected

Table 3. Risk increase factor for the basic event (Top event low water level)

#	Name	Risk incr.	Comment
1	SYVE_VM_FO	3.59E+02	Control valve fails open
2	VU_PM_FD	3.59E+02	Failure of processor, detected
3	VU_SR_FD	3.59E+02	Failure of processor subrack, Detected
4	VU_PM_FU	3.59E+02	Failure of processor, undetected
5	VU_OM_FU	3.59E+02	Failure of output module, undetected
6	VU_OM_FD	3.59E+02	Failure of output module, detected
7	APU2_IM_FD	3.59E+02	Failure of APU2 input module, detected
8	WLM2_FD	3.59E+02	Failure of WL sensor 2, detected
9	APU1_PM_FD	3.59E+02	Failure of APU1 processor, detected
10	APU1_SR_FD	3.59E+02	Faliure of processor subrack, detected
11	APU1_IM_FD	3.59E+02	Failure of APU 1 input module, detected
12	WLM_D-12	3.59E+02	2x CCF
13	WLM1_FD	3.59E+02	Failure of WL sensor 1, detected

14	APU2_PM_FD	3.59E+02	Failure of processor, detected
15	APU2_SR_FD	3.59E+02	Failure of processor subrack, detected
16	WLM_U-12	3.59E+02	2x CCF
17	APU1_PM_FU	2.65E+00	Failure of APU1 processor, undetected
18	APU2_PM_FU	2.65E+00	Failure of APU2 processor, undetected
19	WLM2_FU	2.64E+00	Failure of WL sensor 2, undetected
20	APU2_IM_FU	2.64E+00	Failure of APU2 input module, undetected
21	WLM1_FU	2.64E+00	Failure of WL sensor 1, undetected
22	APU1_IM_FU	2.64E+00	Failure of APU 1 input module, undetected