Generation of a Failure Mode and Effects Analysis with smartIflow

Christian Müller

Department of Computer Science, Ulm University of Applied Sciences, Germany.
E-mail: christian.mueller@thu.de

Rüdiger Lunde

Department of Computer Science, Ulm University of Applied Sciences, Germany.
E-mail: ruediger.lunde@thu.de

Philipp Hönig

Department of Computer Science, Ulm University of Applied Sciences, Germany.
E-mail: philipp.hoenig@thu.de

Failure Mode and Effects Analysis (FMEA) is a common technique for the comprehensive analysis of technical systems against reliability and safety. Besides, it is often required by industry standards in many domains. While the information gained about the system is of high value, performing an FMEA manually is labor-intensive and costly. Therefore, it is mostly done only once near the end of the design. In addition, a manually performed FMEA is error-prone and its quality depends on the experience of the analysts. In order to face these problems, automating the FMEA process can be part of one way to solve it. In this paper, we will discuss an approach which generates an FMEA from qualitative models using model checking techniques instead of simulation. This is done by using two different algorithms, with the results combined at the end. The first one is based on state comparision, while the second uses Computation Tree Logic (CTL) for its analysis. The models used for this approach are modeled with the smartIflow modeling language, a language designed for modeling and analysis of technical system in early design stages. According to this, the here introduced approach can also be used very early and deliver information about possible hazards of the system concept. After discussing the procedure of our approach in detail, we will apply the approach using a diagonal car braking system as example model and have a look at the occurring problems, possible solutions and obtained results.

Keywords: FMEA, Qualitative analysis, Finite state machine, Safety analysis, Reliability, Formal languages, Temporal reasoning, Model checking, smartIflow.

1. Introduction

In the late 1940s, the US Armed Forces Military described in the document MIL-P-1629 (later revised as MIL-STD-1629A (1980)) procedures for conducting FMECA (Failure Mode, Effects and Criticality Analysis). A few years later, variations of FMEA were used in aerospace programs. Around the 1970s, the FMEA spread into many other industries. These days it is one of the most important tasks in many reliability programs and part of standards as the ISO 26262-1:2018 (2018) in the automotive industry or ARP4761 (1996) in the civil avionics industry. When done right, an FMEA provides valuable information about potential failures and their impact on the systems functionality. These information are useful for evaluating failure effects and estimate potential risks and weak points. Ideally, an FMEA is applied consistently starting at early concept stage and still maintained during the life of the product. Therefore, engineers from different divisions are necessary to identify each potential failure mode and its possible effects on the system. The more experience the enginieers have, the higher is the quality of the resulting FMEA. But because performing an FMEA manually is labor-intensive and costly, it is mostly done once near the end of the design. Unfortunately, the later an FMEA is performed, the harder it gets to correct design errors which can result in higher costs.

This situation could be improved by automation of the FMEA process. In this paper we will discuss an approach of an FMEA generator. It uses qualitative models and model checking techniques for the generation process. The models the approach is applied to are modeled with the smartIflow modeling language. This language is designed for modeling and analysis of technical systems in early design stages and thus fits perfectly to the ideal starting point of an FMEA.

Proceedings of the 30th European Safety and Reliability Conference and
the 15th Probabilistic Safety Assessment and Management Conference
Edited by Piero Baraldi, Francesco Di Maio and Enrico Zio
Published by Research Publishing, Singapore.
ISBN: 978-981-14-8593-0; doi:10.3850/978-981-14-8593-0

This paper is organized in the following way. In Section 2 and Section 3 we will discuss fundamentals and related work. Section 4 will describe the used algorithms for the FMEA generation with smartIflow. The following Section 5 introduces the diagonal car braking system, which is used as a case study. Section 6 will deal with the results and Section 7 will give a conclusion with an outlook to future enhancements.

2. Fundamentals

2.1. *smartIflow*

The smartIflow modeling language (see Hönig et al. (2017) for more details) is designed to describe systems on an abstraction level that is useful for safety analysis in an early product development stage. The language itself is object- and component-oriented, which allows to model systems as a combination of components with a hierarchical structure. A component is thereby modeled as a finite state machine. Components can be connected on their ports, which allows them to interact with each other by exchanging messages, so called properties. Port properties can either be messages in form of key-value-pairs, or energy values based on effort, flow or resistance. The state of a component is expressed by discrete variables. Values of those variables can be changed by internal state transitions triggered by value changes of port properties. A state change can also be performed by external events. The behavior of a component depends on its state. Here, it can change the connection network by connecting own ports or ports of sub components. For example, a valve in an opened state can connect its input and output port to allow the flow to pass trough. Another possibility is to set properties on a port instead of changing the connection network. A sensor, for example, can send its observation to an controller in this way. In the following listing an example component modeled with smartIflow can be seen.

```
class HydraulicLine {
 Ports:
  Hydraulical p1;
  Hydraulical p2;
 Variables:
  Enum[OK, Leakage] fm = OK;
 Events:
  Leakage{type=failure,p=0.5e-5};
 Behavior:
  connect(p1, p2);
  if (fm == Leakage)
   set(p1, {leakage=true});
 EventHandlers:
  when (Leakage) [fm == OK]
   fm = Leakage;
}
```

The system model in smartIflow represents an implicit description of the systems transition system, which describes the possible state space. While it is quite comfortable to express a systems possible state space this way, these descriptions are not well suited for model-checking algorithms as described in Baier and Katoen (2008). Therefore, the implicit description has to be transformed into an explicit representation of the transition system. The algorithm presented by Hönig et al. (2017) is able to perform such a transformation and results in an extended transition system in form of a labeled graph. In Figure 1 a visualization of such a graph is shown as it is represented by the smartIflow Workbench[a], a tool to model and perform analysis with smartIflow models.

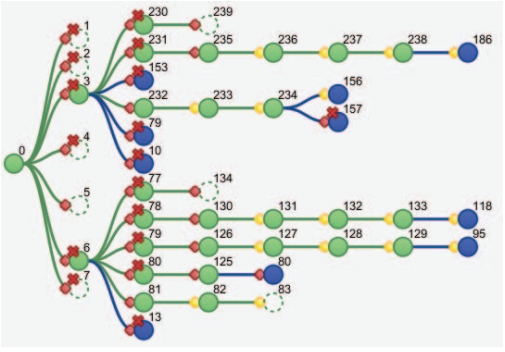

Fig. 1. Example visualization of an extended transition system in the smartIflow Workbench

A state node describes a unique state within the transition system and is represented as a green round node. A blue node indicates a reference to another already existing state node. In front of these nodes either a red or a yellow rectangle can be found. These rectangle indicate if the state node is reached by an external event (red) or an internal transition (yellow).

Because the state space can become very large, it is possible to limit the amount of unfolded states. This can be done by defining an *Event Trigger Specification*. These specifications are used during the unfolding process to decide if a path is further unfolded. For example, a specification can allow only two failure events per path. The unfolding algorithm will then only process states within a path till this specification is violated. Beside decreasing the necessary time and memory, they also provide a way to focus on certain aspects of the system. In earlier experiments with smartIflow such specifications proved already as

[a]A smartIflow Workbench test version is available at https://smartiflow.bitbucket.io/

very useful. This is shown for example in the experiment with the wheel brake system in the avionics domain, demonstrated in Müller et al. (2018), and the experiment with an railroad crossing system demonstrated in Lunde et al. (2018).

2.2. *Failure Mode and Effects Analysis*

A *Failure Mode and Effects Analysis (FMEA)* is perfomed to identify potential failures of a system and assess their effects. With this information it is then possible to define appropriate countermeasures to eliminate or at least minimize the potential risk of the system failures. The most common outcome is an FMEA table documenting each failure mode of each component. The columns of the table can differ between domains and industry. As described in MIL-STD-1629A (1980), an FMEA is focused on single faults only.

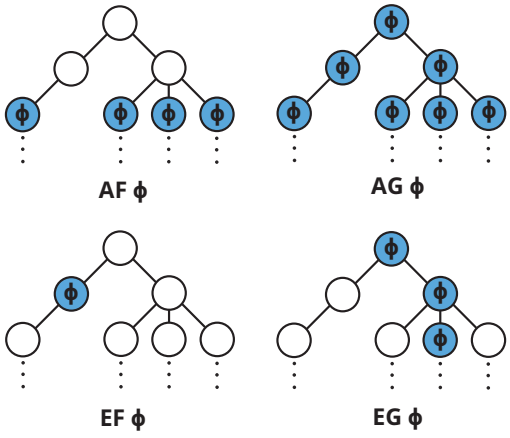

Fig. 2. Illustrated examples of CTL formulae

2.3. *Computation Tree Logic*

The *Computation Tree Logic (CTL)*, introduced in Clarke and Emerson (1982), is a branching-time logic based on a time model consisting of many different future paths. Thus, the time model has a tree-like stucture. Formulae defined in CTL are interpreted over a transition system and allow to verify if the transition system holds a certain property. Due to the tree-like structure it is possible to define formulae which can verify a nondeterministic state development. That is why CTL comes in useful for model checkers like NuSMV (Cimatti et al. (2000)) and PRISM (Kwiatkowska et al. (2001)) to check if a finite-state model meets a given specification.

Besides the logical operators $\neg$, $\wedge$, $\vee$, $\Rightarrow$ and $\Leftrightarrow$ the CTL syntax provides temporal operators, which are a combination of a quantifier over paths and a path-specific quantifier. The two possible quantifier over paths are *A* (holds on *all* paths) and *E* (at least one path *exists* holding the property). As path-specific quantifiers the syntax supports the three quantifiers *X* (at the *next* state), *F* (at one state in the *future*) and *G* (at all future states, *globally*). In combination, this gives the six temporal operators *AX*, *AF*, *AG*, *EX*, *EF* and *EG*. Figure 2 shows some examples illustrating the meaning of the operators.

3. Related Work

To facilitate the performance of an FMEA for technical systems, which become more and more complex, an automated process can be of great benefit giving the possibility to better integrate FMEA into the design and development process. While the wanted result is clear, getting an automatically generated FMEA table, there are various approaches to gather the necessary information to achieve this result.

The approach to automate FMEA described in Papadopoulos et al. (2004) is based on fault trees created by using the HiP-HOPS methodology introduced in Papadopoulos et al. (1999) and works on models made with MATLAB/Simulink. The first step in this approach is the manual establishment of the local failure behavior for each component in the system. The result is a set of failure expressions describing the components output failures and how they are caused by internal malfunctions and input deviations. Then the model structure together with the failure expressions are used to automatically construct fault trees. The minimal cutset of these trees represents the relation between failure effects of the system and the component's local failure mode. This information is used to create the FMEA table. The advantage hereby is, that fault trees can deliver additional information of system failures caused by more than one faulty component. This information can be used to improve the assessment of the significance of a failure mode. While large parts of the approach are automated, the most labor-intensive work, creating the sets of failure expressions for each component including the failure annotations and the definition of how failures propagate through it, must be done by hand. In smartIflow the possible faults and the fault behavior of a component are defined in the component class and thus are part of the model. During the unfolding process, the failures are propagated automatically. Simliar to this approach, the FMEA generator described in our paper is also using fault trees to find out effects on the system. In contrast to the approach with HiP-HOPS, the fault trees are automatically generated with a model checker.

Grunske et al. (2005) describes an automated FMEA approach based on a combination of behavior trees and model checking. The behavior

tree notation introduced in Dromey (2003) is a high-level graphical notation. It can be used to transform functional requirements given as descriptions in natural language into a formal representation. The model checking is done by using the SAL model checker. The therefore needed description of system hazards is done by using Linear Temporal Logic (LTL). According to this approach, the user has to create the behavior trees of the system as well as defining the system hazards with LTL. Before the behavior trees are automatically transformed into SAL code for the SAL model checker, faults must be injected into the behavior trees by adding the fault behavior to the trees. After that, the model checker verifies the LTL formulae with the generated SAL code. While the transformation of the system requirements into behavior trees can be done comfortably, the major problem of this approach is caused by the model checker, as the authors noted. Most model checkers, including the SAL model checker, provide only a single counterexample. Because the counterexample is used for the hazard analysis, the quality of the counterexample is the linchpin of the automated analysis process. Our approach also uses a model checker to gather information for the hazard analysis. But in contrast to this approach, the hereby used model checker is able to provide multiple counterexamples. This gives the possibility to recognize multiple failures leading to a violation of the formula at once.

In Price and Taylor (2002) an automated FMEA approach in the electrical domain is proposed. This approach is based on component-based simulation where descriptions of component failure behavior is available. At first, the system is simulated with all components working as intended. The functions occuring in each state are noted and used as nominal reference result. Then for each failure the system is simulated again with the corresponding failure active within the component. The here noted occuring functions are compared with the nominal reference. Differences between both versions are noted as effects of the failure mode. To assess the severity and detectability, it is possible to associate these values with the component's functions while the occurrence frequency is extracted from component failure information. The idea of comparing nominal system behavior with the behavior when a failure is active is also used in our approach. But instead of activating the failure at the begin and simulate the system behavior, the state space of the system is expanded. The unfolded model contains paths for all possible sequences of events. This means, arbitrary function and fault activations as well as deactivations are studied, and the obtained system reaction in every possible situation is taken into account.

4. FMEA Generator

The main purpose of an FMEA is to show the possible effects of a component's failure mode to the component itself, to other components of the system and the influence on the systems behavior. Gathering the therefore necessary information belongs to the basic problems when it comes to automate the generation of an FMEA report. The here introduced approach tries to solve this problem by using different algorithms to gather information. These algorithms will be reffered as *Effect Finders*. The extended transition system is passed to each *Effect Finder*, which applies its particular algorithm to investigate the system and gather information about the failure modes and their effects. At the end, the results of the different *Effect Finder*s are merged together to produce the FMEA report. In Figure 3 the structure of the FMEA generator is visualized schematically.

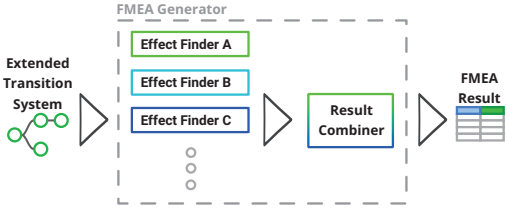

Fig. 3. Schematic structure of the FMEA Generator

In the following, two *Effect Finder*s are explained in more detail. The first one is focused on finding the effects of a failure mode based on state comparison. The second one tries to find out the effects of a failure mode on the systems functionality using CTL formulae. Both *Effect Finder*s are based on the assumption, that an external failure event can change the failure mode of a component to only one specific mode. But it is possible that multiple external failure events can change the failure mode of a component to the same mode. Before an FMEA can be generated, the variable indicating the failure mode of a component has to be marked. This can be done by addding the feature *type* with value *failureMode* as shown in the listing below.

```
Variables:
 Enum[OK, Leakage]
  fm{type=failureMode} = OK;
```

After that, the *Effect Finder*s can match their findings to the according failure mode.

4.1. *State-based Effect Finder*

The *State-based Effect Finder* searches for the effects of failure modes based on states. Therefore, it compares nominal and failure paths of

the extended transition system to find out the state changes caused by each external event of the components. Due to the relation between external event and failure mode, it is then possible to derive the local effects of the failure mode to the corresponding component. The procedure of this *Effect Finder* is split into five consecutive steps:

(1) Search and collect appropriate paths within the systems transition system
(2) Split the found paths into groups of nominal and failure paths
(3) Within both groups, form subgroups depending on the external events of the paths
(4) Compare matching nominal subgroups with failure subgroups to find out the effects of an event
(5) Merge all found effects of an event and assign them to the appropriate failure mode

We will now describe each step in more detail. At the beginning, the transition system is traversed and searched for appropriate paths. In this context, an appropriate path has several properties. First of all, it does not have any loops. This property prevents the algorithm to get stucked into an endless loop and also sorts out paths with no potential for new information. Second, a path ends with a state node where external events arise. Here, it is assumed that the internal transitions after the occurrence of an external event represent the processing of the event by the system. To further limit the traversed state space, an *Event Trigger Specification* is used to narrow the state space. The result after this step is a set of paths.

During the second step, the found paths are separated into nominal paths and failure paths. A nominal path shows the behavior of the system model when no external failure event occurs, and thus, no system failure exists. In contrast, a failure path shows the behavior of the system model when at least one external failure event is triggered and therefore a component entered a failure mode. The separation is necessary to distinguish between nominal states and failure states of the system model in later steps.

In the third step, subgroups within the group of nominal paths and failure paths are formed. These subgroups are based on an analysis of the external events within the paths. A subgroup is here reffered as *Event Combination*. The amount of events within an *Event Combination* is correlated with its degree. That means, that for example an *Event Combination* of three events is of degree three. The result of this step is a list of *Event Combinations* with the paths they are found in.

In the fourth step the comparison of nominal and failure paths takes place. Here, the last state of each path of a failure event combination of degree n is compared with the last state of all paths of a matching nominal event combination of degree $n-1$. The detected state differences are gathered as effects of the arised external failure event.

The last step assigns the found effects to the failure mode of a component based on the external events. Additional information, for example the probability of the triggered event, is also added and available for further calculations.

4.2. *CTL-based Effect Finder*

The main purpose of the *CTL-based Effect Finder* is to find out the effects of a component's failure mode on the whole system functionality. This is done by using the model checker described in Hönig et al. (2016). This model checker is able to return multiple counterexamples at once, instead of just a single one as most model checkers do. With CTL formulae the wanted nominal behavior of the modeled system is defined. In other words, these formulae are an temporal logical form of the system requirements. After verifying the extended transition system with respect to a CTL formula, the resulting counterexamples can be used to generate minimal cutsets of failure events. While it is possible to generate fault trees with these minimal cutsets as shown in Lunde et al. (2018), they are used here to conclude the effect of a failure mode on the system. Therefore, the relation between failure mode and external event is used again, to find out which failure modes are responsible for the violation of the CTL formula. Thereby, the system effect is that the corresponding requirement, represented by the CTL formula, cannot be satisfied.

4.3. *Combination of Effect Findings*

After each *Effect Finder* has investigated the extended transition system, the different findings are combined. This is done by taking the event the findings occurred as a key property. All findings for the same event of the same component are combined and stored as final result for the corresponding failure mode. Additionally, if these findings contain values for the Severity, Occurrence or Detection, the best suited value is selected for the result by calculating the maximum for Severity and Occurrence or calculating the minimum for the Detection.

4.4. *Textual Translation*

The output for local effects and end effects by the *Effect Finders* is on a very low level and may be not sufficient. For the *State-based Effect Finder* it is possible to define textual descriptions using the markup language YAML, which are shown if a certain pattern of values is found. It is possible to define the variable either by using the explicit path within the system model, as shown in the listing, or by using the path of the component's class on package level. While the first version allows

to define translations for certain components, the second version allows to define translations for all components of the same component class.

```
- description: "Pedal released"
  states:
   pedal.state:
    value: Released
```

For the *CTL-based Effect Finder* a simliar solution exists. Instead of defining the pattern of variables and their values, the CTL formula which must be violated is given. The description will then show up for every failure mode violating the defined CTL formula. Additional to the textual description it is also possible to add a severity value and a detection value. If multiple formulae are violated by the same failure mode, the maximum of these values is calculated and used in the FMEA. The in the example used CTL formula is defined in Section 5.

```
- description: "Safe braking in
    time is not possible"
  ctl: SafeBrakingInTime
  severity: 9
  detection: 4
```

4.5. *FMEA Table Layout*

As already mentioned in Section 2.2, the columns of the FMEA table can differ between domains and industry. Therefore, the table layout used by the FMEA generator can be configured to the specific needs. An FMEA layout in smartIflow consists of a configuration file and a script file. The configuration is also using YAML and can look like the following example.

```
domain: Automotive
name: Example Layout
columns:
  - title: Item
    identifier: item
  - title: Failure Mode
    identifier: failure_mode
  - title: End Effect
    identifier: end_effect
  - title: S
    identifier: severity
    script-function: s_format
```

The keywords *domain* and *name* are used to identify the configuration file. Under the keyword *columns* a list of the desired columns can be specified. The order of the entries corresponds to the order of the columns in the table. The keyword *title* defines the title of the column, while *identifier* determines which part of the found result should be placed in the column, e.g. *end_effect* places the found end effects into the column. With *script-function* it is possible to tell the generator to execute the given script function on each cell of the column. These script functions are located in the script file and are written in Python. An example function is given below.

```
def s_format(value,row,results):
    return "Unacceptable" if value
        > 0 else "Acceptable"
```

This example function replaces the severity value in the column with *Unacceptable* if the value is above zero or else with *Acceptable*. Every function defined in the script file can have parameters, which contain values passed by the generator. The first parameter *value* contains the value of the table cell, the second parameter *row* gives access to the data of a whole row. The third parameter can be used to access all results available. Beside simple formatting of the content, these parameters can further be used to calculate own values. Through the script functions it is also possible to include data from other data sources and extend the FMEA table.

5. Case Study: Diagonal Car Braking System

In this section we will define the system used as case study to take a look at an generated FMEA as described in Section 4. The system we will use is a diagonal car braking system as seen in Figure 4.

The diagonal car braking system consists of four disc brakes, four hydraulic lines, one master cylinder and one pedal. To keep it simple only three components are modeled with one possible failure mode. The pedal can break, a hydraulic line can have a leakage and the master cylinder can have a delayed response to the incoming signal of the pedal.

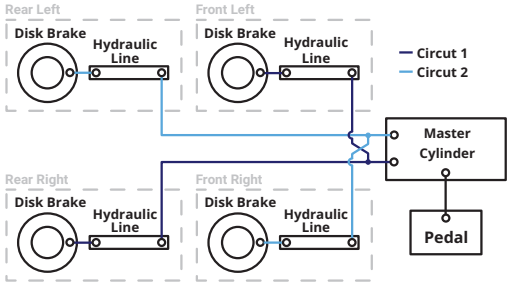

Fig. 4. Schematic diagram of the diagonal car braking system

The nominal behavior of the diagonal car braking system is as follows. When the pedal is pushed the master cylinder switches its state to *operated*

and sets the hydraulical lines under pressure. The pressure reaches the disc brakes via the hydraulic lines. The disc brakes are activated and the car begins to brake. If the pedal is released, the master cylinder sets no longer the hydraulic lines under pressure. The braking process is stopped and the car can drive again.

The *CTL-based Effect Finder* will check two system functions. First of all, it will check if the system is able to brake ideally. This means, that all four disc brakes are braking if the pedal is pressed. Furthermore, it will be checked if the system is able to brake in time. The CTL formulae for these two functions are in the listing below.

```
allBrakes :=
 frontLeftBrake.opm == Braking &&
 rearLeftBrake.opm == Braking &&
 frontRightBrake.opm == Braking &&
 rearRightBrake.opm == Braking;

IdealBraking :=
 AG(pedal.opm == Pushed
  -> AF allBrakes);

SafeBrakingInTime :=
 AG(pedal.opm==Pushed{stable>=3}
  -> AF{delay<=1}
   brakeOnEachSide{stable>=3});
```

The formulae are using the variables *allBrakes*, defined as all brake discs are in the operation mode braking, and *brakeOnEachSide*, defined as either the brakes of circuit 1 or circuit 2 are braking. Furthermore, as the formula *SafeBrakingInTime* suggests, the CTL notation used is extended with a time extension making it possible to add qualitative time constraints to the formula.

6. Results / Discussion

We will now take a look at the results of the FMEA generator, or more precisely on the results of each *Effect Finder*. Due to limitation of space, Figure 5 shows parts of the result of the *State-based Effect Finder*.

The figure shows three representatives of the component classes and their failure mode. The components affected by a component's failure mode are shown in the column *Affected Columns*, while the column *Component-wide Effects* shows the effects on the state of these components. As the table shows, the *leakage* of the *front-right brake line* affects both brakes within *circuit 2* as well as the *master cylinder*. Because the reservoir within the *master cylinder* is empty due to the leakage, the brakes within the circuit won't be able to brake. In case of a *broken pedal*, we can see that the *master cylinder* is not operated and thus all *brakes* won't brake. Both columns are empty if the *master cylinder* has the failure mode *Delayed*. This is because the *State-based Effect Finder* does not consider timing constraints.

In Figure 6 parts of the result of the *CTL-based Effect Finder* are shown.

As the table of the *State-based Effect Finder* results, the table shows three representatives of the component classes and their failure modes. The failure mode of the *frontRightBrakeLine* and the *pedal* are violating the CTL formula which describes the requirement for ideal braking. The formula describing safe braking in time is violated by the *master cylinder* and the *pedal*. In case of the *pedal* it should be clear, that if it is broken it is impossible to brake at all. The *delayed master cylinder* still provides the brakes with pressure, but this happens not in time and thus violates the requirement.

7. Conclusion and future enhancements

In this paper, we presented an approach to generate FMEA from a qualitative model based on smartIflow. This approach uses different algorithms to extract information from the system model and combines them for the final FMEA table.

As shown in the case study, both *Effect Finder*s have their advantages and disadvantages. The *State-based Effect Finder* is able to find out the effects of a failure mode on the component itself as well as on other components within the system. But that is only possible without time constraints. With the *CTL-based Effect Finder* it is possible to define requirements or system functions as CTL formulae and use them as input for the FMEA to find out effects on the system functionality. Due to an added time extension to the CTL notation, it is also possible to include timing constraints into the FMEA. However, defining a formula requires some time and effort. Combining the results of both *Effect Finders* makes it possible to generate an early FMEA with an abstract system model.

Due to the modular structure of the FMEA generator it is possible to extend the available information by new *Effect Finder*s. A possible new *Effect Finder* could be based on the properties exchanged between components. This way, the interaction between components could also be considered during the generation. Furthermore, the *CTL-based Effect Finder* could be extended to use the minimal cutsets with more than one failure mode to generate FMEAs with combined failure modes.

Acknowledgements

This work was partially funded by HES-SO//FR HEIA-FR ROSAS Center Fribourg in the project SysRO. The authors sincerely thank the project partners for the collaboration and support.

Item	Failure Mode	Affected Components	Component-wide Effects	Local Effect
frontRightBrakeLine	Leakage	frontRightBrake masterCylinder rearLeftBrake	frontRightBrake.opm = Idle masterCylinder.circuit2ReservoirLevel = Empty rearLeftBrake.opm = Idle	Hydraulic line has a leakage
masterCylinder	Delayed			Master Cylinder operates delayed
pedal	Broken	frontLeftBrake frontRightBrake masterCylinder rearLeftBrake rearRightBrake	frontLeftBrake.opm = Idle frontRightBrake.opm = Idle masterCylinder.opm = Idle pedal.state = Released rearLeftBrake.opm = Idle rearRightBrake.opm = Idle	Pedal is broken Pedal released

Fig. 5. Part of the table with information from the State-based Effect Finder

Item	Failure Mode	End Effect	S	O	D	RPN
frontRightBrakeLine	Leakage	Ideal braking is not possible	1	4	2	8
masterCylinder	Delayed	Safe braking in time is not possible	9	4	1	36
pedal	Broken	Ideal braking is not possible Safe braking in time is not possible	9	4	1	36

Fig. 6. Part of the table with information from the CTL-based Effect Finder

References

ARP4761 (1996). Guidelines and methods for conducting the safety assessment process on civil airborne systems and equipment. Standard, SAE International.

Baier, C. and J.-P. Katoen (2008). *Principles of Model Checking*. The MIT Press.

Cimatti, A., E. Clarke, F. Giunchiglia, and M. Roveri (2000, mar). NUSMV: a new symbolic model checker. *International Journal on Software Tools for Technology Transfer (STTT) 2*(4), 410–425.

Clarke, E. M. and E. A. Emerson (1982). Design and synthesis of synchronization skeletons using branching time temporal logic. In D. Kozen (Ed.), *Logics of Programs*, Berlin, Heidelberg, pp. 52–71. Springer Berlin Heidelberg.

Dromey, R. (2003, 10). From requirements to design: Formalizing the key steps. pp. 2 – 11.

Grunske, L., P. Lindsay, N. Yatapanage, and K. Winter (2005, 01). An Automated Failure Mode and Effect Analysis Based on High-Level Design Specification with Behavior Trees. In *Lecture Notes in Computer Science*, pp. 129–149. Springer Berlin Heidelberg.

Hönig, P., R. Lunde, and F. Holzapfel (2016). Formal Verification of Technical Systems Using smartIflow and CTL. In E. Pyshkin, V. Klyuev, and A. Vazhenin (Eds.), *Proceedings of the 2nd International Conference on Applications in Information Technology (ICAIT-2016)*, Aizu-Wakamatsu, Japan. The University of Aizu Press.

Hönig, P., R. Lunde, and F. Holzapfel (2017). Model Based Safety Analysis with smartIflow. *Information 8*(1).

ISO 26262-1:2018 (2018). Road vehicles Functional safety Part 1: Vocabulary. Standard, International Organization for Standardization.

Kwiatkowska, M., G. Norman, and D. Parker (2001, September). PRISM: Probabilistic symbolic model checker. In P. Kemper (Ed.), *Proc. Tools Session of Aachen 2001 International Multiconference on Measurement, Modelling and Evaluation of Computer-Communication Systems*, pp. 7–12.

Lunde, R., P. Hönig, and C. Müller (2018). Reasoning about Different Orders of Magnitude of Time with smartIflow. *IFAC-PapersOnLine 51*(24), 131–138.

MIL-STD-1629A (1980). Procedures for performing a Failure Mode, Effects and Criticality Analysis. Standard, Department of Defense USA.

Müller, C., P. Hönig, and R. Lunde (2018). Evaluation of smartIflow based on the Wheel Brake System from ARP4761. *IFAC-PapersOnLine 51*(24), 1255–1262.

Papadopoulos, Y., J. A. McDermid, D. of Computer Science, U. of York, Y. 5DD, and UK (1999). Hierarchically Performed Hazard Origin and Propagation Studies. In *In Proceedings of SAFECOMP 99, 18th International Conference on Computer Safety, Reliability and Security, Toulouse France, Lecture Notes in Computer Science, 1698:139-152, Springer Verlag, 1999.*

Papadopoulos, Y., D. Parker, and C. Grante (2004, March). Automating the failure modes and effects analysis of safety critical systems. In *Eighth IEEE International Symposium on High Assurance Systems Engineering, 2004. Proceedings.*, pp. 310–311.

Price, C. and N. Taylor (2002). Automated multiple failure FMEA. *Reliability Engineering & System Safety 76*(1), 1 – 10.