09AE-0150

Semi-Automatic FMEA supporting complex systems with combinations and sequences of failures

Martin Walker, Yiannis Papadopoulos, David Parker
University of Hull, UK

Henrik Lönn
Volvo Technology Corporation

Martin Törngren, DeJiu Chen
Royal Institute of Technology

Rolf Johansson
Mentor Graphics Corporation

Anders Sandberg
Mecel AB

ABSTRACT

Failure Modes and Effects Analysis (FMEA) is a well established safety analysis technique used for the assessment of safety critical engineering systems in the automotive industry. Although FMEA has been shown to be useful, the analysis is typically restricted to the effects of single component failures; even partial analysis of combinations or sequences of multiple failures is in practice considered too complex, laborious and costly to perform. In this paper, we describe a new technique in which FMEAs are semi-automatically built from the topology of a system and component-level specifications of failure data. The proposed technique allows an extended form of "combinatorial & sequential FMEA" in which assessment of the effects of combinations and sequences of failures becomes feasible and cost effective. We show how this technique can address difficulties encountered in classical FMEA and, drawing from a simplified brake-by-wire example, we show how it can improve the assessment of safety critical automotive systems.

1 INTRODUCTION

Failure Modes and Effects Analysis (FMEA) is an inductive technique in which analysts study a system to identify the potential component failure modes and then infer the effects of those failure modes on the rest of the system. System models, such as engineering diagrams or functional block diagrams, are typically used to assist the analysts in understanding how component failures can affect other parts of the system. Once complete, the analysts can then assess the likelihood of those failures and the severity of their effects before estimating the potential risk they pose. The system can then be redesigned to avoid or minimise the effects of the most critical failures identified in the analysis and thereby improve the safety of the system.

Although FMEA is generally seen as a useful technique, it remains manual, laborious and expensive to apply. The analysis typically remains incomplete, being restricted to the assessment of the effects of single component failures, as exhaustive assessment of combinations of failures is very difficult to achieve in large systems. If, for example, a system has 1000 potential component failures, assessment of the effects of combinations of 2 such failures requires that the analysis is repeated approximately half a million times. This combinatorial explosion problem is further exacerbated in cases where the effects of component failures depend on the temporal ordering – or sequence – of those failures. This type of temporal safety analysis is simply impossible in a classical FMEA.

The problem of combinatorial explosion can be partially mitigated by automating the process, and a number of software tools have been developed to help with the process of conducting an FMEA. Unfortunately, they were often designed to assist solely with the clerical aspects of the analysis and enforce consistency. Therefore, the analysis of the system itself often remains a manual process – a team of analysts still has to study the system models, identify component failures, and determine their effects on the rest of the system. With

the increasing complexity of systems, many of which integrate different technology domains, it is becoming more difficult to be able to perform thorough analyses and determine the full consequences of individual failure modes; a problem exacerbated by a trend towards shorter design periods. Even if the FMEA is restricted to single points of failure, as system complexity increases, it becomes more likely that the analysis will contain either errors or omissions.

To overcome this problem, there has been a considerable body of work in recent years focused on ways of automating FMEA. One such methodology makes use of fault simulation [1-5]. Simulation of this sort requires domain modelling which restricts its usage to domains for which simulators and their requisite models have been developed. For example, fault simulation has been successfully applied to electronic systems in [6]. However, such approaches are computationally very expensive and do not help to overcome the inability of FMEA to analyse combinations or sequences of failures.

Another way to master the complexity is to take advantage of the hierarchical description of a system. In [14] the authors propose a method where sub-components are analysed independently of the overall system, and the analysis that takes place on higher integration levels abstracts away from details that are refined on lower component levels. Although this strategy would improve the situation, combinatorial explosion in a hierarchical FMEA would still cause difficulties at each level of the analysis.

In this paper, we outline an approach to synthesis of combinatorial FMEAs using a recently proposed but by now well established compositional safety analysis technique called Hierarchically Performed Hazard Origin and Propagation Studies (HiP-HOPS) [7]. This is a semi-automated approach to FMEA that can be applied to a greater range of systems, be they mechanical, electrical, hydraulic, software-based, or some combination thereof. Because HiP-HOPS generates the FMEA via automatically generated fault trees – which are deductively produced, starting with the system failure and working back to determine the root causes – it does not suffer in the same way from combinatorial explosion when considering multiple failure modes. Furthermore, by making use of recent advances in temporal safety analysis – known as Pandora [8] – HiP-HOPS enables the analysis to incorporate the effects of different sequences of failures as well, resulting in a new form of 'sequential' FMEA.

In section 2, we describe recent extensions to the HiP-HOPS tool that enable semi-automatic synthesis of combinatorial FMEAs. In section 3, we explain how the technique is capable of producing sequential FMEAs by introducing Pandora, and then in section 4 we present a small case study that demonstrates the benefits of such an approach in producing more comprehensive and inclusive information about the safety of a system. Finally, in section 5 we present our conclusions and state how this new technique provides useful advances in the field of FMEAs.

2 COMBINATORIAL FMEA IN HIP-HOPS

The HiP-HOPS tool performs its analysis on the basis of an annotated system model that provides the tool with the topology of the system: the components and the connections between them. The system model can be flat or hierarchical, with components containing architectures of subcomponents which can in turn contain more subcomponents. This also means that different models can be used at different stages of the design process; for example, a relatively simple functional block diagram might be used early in the design and could be refined with a schematic showing the detailed hardware architecture of the system. Later on in the design lifecycle, programmable components in a hardware architecture may be further refined, enclosing software architectures described using functional or data flow diagrams. In the context of work with EAST-ADL [15], it has also been shown that HiP-HOPS studies can also be performed on layered architectures that combine different levels abstraction, functional, hardware and software.

The annotations in the model provide HiP-HOPS with *local failure data*. This mainly takes the form of a set of logical expressions that show how failures at the component's outputs (known as output deviations) can be caused by a combination of input deviations – failures detected at the component's input (e.g. an omission of signal) – and internal failure modes of the component itself. Boolean operators such as AND and OR can be used to create the expression and this enables HiP-HOPS to analyse the effects of more than one failure mode occurring in combination.

Input and output deviations are given a *failure class*. This gives some indication of the nature of the failure and generally falls into one of three categories: provision, e.g. an omission or unexpected commission of a signal; timing, e.g. late or early reception of an input; and value, which indicates an error in input/output (e.g. high pressure, low voltage etc). Internal failure modes do not have a failure class but can include failures of many different types, such as environmental influences external to the component (e.g. EMI, temperature), material failures of the component itself, and human error in the control of the component.

As an example of local failure data, consider a simple pump. It could respond to and originate failures in a number of ways, e.g.

- Omission-outflow = Omission-power OR mechanical_failure OR blockage OR Omission-inflow
- Commission-outflow = Commission-power
- Low-outflow = partialBlockage OR Low-power
- High-outflow = High-power

In some cases, such as with the commission of output flow, it effectively propagates a failure at an input to a failure at its output; in other cases, such as with the omission of output flow, there are many possible causes, both internal and external to the component. The failure class of a deviation is the first part, prior to the dash, and the input/output ("port") of the component is the second part of the deviation; internal failures such as "blockage" do not have a specific port as they originate within the component and do not represent effects on inputs or outputs.

Similar components can reuse the same local failure data. Because the data is mostly context independent, identical or similar failure data could be used for all pumps in the system, and similarly for other types of H/W or S/W components. This makes it easier to annotate large models by minimising the amount of manual work required; and annotations can even be stored as part of a component type in a library so that the local failure data is already present when the component is added to the model[1].

Since the effects of component failures are rarely limited to the components themselves, a key objective in safety analysis is tracing the propagation of component failure through the model up to a related hazard on system level. HiP-HOPS achieves this by linking the output deviations of one component to the input deviations of another, so for example an omission of flow from a pump could cause a hydraulic actuator to stop functioning which in turn could have wider ramifications on the rest of the system. HiP-HOPS even allows finer control over this propagation by also allowing the connections between components to be annotated with logical expressions.

HiP-HOPS starts this process at the outputs of the system as a whole by taking a system failure (a hazard when doing safety analysis) and then working backwards through the model, following the connections between components, until it can go no further. The result of this is a set of interconnected fault trees which show how individual component failures in the model can combine to cause one or more system failures.

These fault trees can then be flattened using qualitative fault tree analysis (FTA) techniques to obtain the *minimal cut sets* of the fault tree. These are the combinations made up of the smallest numbers of failures necessary to cause a system failure, and illustrate the direct links between original cause and ultimate failure. It is then possible to use these results to construct a multiple-event FMEA, showing not only the direct effects on the system of each failure mode, but also the further effects that failure mode can have if it occurs in conjunction with other failure modes (i.e. the other failure modes in the minimal cut sets).

For example, assume that HiP-HOPS-style FTA produced five minimal cut sets for two system failures as follows:

- System Failure 1 = {A}, {B, C}
- System Failure 2 = {A}, {B}, {C, D}

The cause-effect relations captured by the HiP-HOPS tool for the conditions specified above is shown in the simplified FMEA Table 1 below[2]:

Failure	Further Failure	Effect
A	None	Sys Failure 1
		Sys Failure 2
B	None	Sys Failure 2
	C	Sys Failure 1
C	B	Sys Failure 1
	D	Sys Failure 2
D	C	Sys Failure 2

Table 1: Combinatorial FMEA

The table illustrates several features that would not be possible in a normal, single-failure FMEA. For example, it shows that although B can cause System Failure 2 by itself, it can also cause System Failure 1 in conjunction with C. It also shows that both C and D can cause system failures if they occur with one other event (including each other); in an ordinary FMEA, neither C nor D would be shown with any effects since they have no direct effect on the system. The table also shows that A is a common cause of two system failures – a clear vulnerability in the design – but this is a result which an ordinary FMEA should also produce.

Furthermore, because this table was generated from a fault tree analysis – a deductive method, working backwards from the system failure to determine the root causes – it avoids much of the combinatorial explosion that afflicts inductive techniques like FMEA, where the analysis starts with the failure and attempts to determine its effects. This in practice means that, once a system model has been annotated with local failure/propagation data, the whole process would typically take only seconds to perform on systems that contain hundreds of components.

For the same reason, HiP-HOPS also makes it possible to perform a quantitative analysis on the system, determining the probability of system failures by

[1] These ideas for reuse of component failure data have been implemented in a commercial tool called Simulation X, which has recently been extended with an interface to HiP-HOPS.

[2] A typical FMEA, and indeed the FMEAs generated by the HiP-HOPS tool, contain more columns which for simplicity and clarity have been omitted in Table 1. Only the important relationship between causes (component failures A, B, C) and effects (system failures 1 and 2) is explored here

calculating the probabilities of individual failure modes, given that proper input failure frequencies are estimated. There are a number of formulae available, including simple constant failure & repair rates or more complex Weibull failure models. This enables the analyst to see what contribution each failure mode makes towards the system failure it ultimately causes, and efforts to improve the reliability of the system can be focused accordingly. This kind of quantitative analysis is however only applicable for the H/W sub-systems, especially when analysing safety properties.

Finally, system failures can also be assigned severity or criticality values to give an indication of the seriousness of their consequences, thus generating a so-called FMECA (Failure Mode Effect and Criticality Analysis). In combination with the probability values, HiP-HOPS can then calculate a risk value for each system H/W failure. This adds extra information to the combinatorial FMEA and makes it possible to compare unlikely but catastrophic events with more likely but less critical failures.

All of these values are calculated automatically from the component annotations; the more information provided, the more comprehensive the results. Overall, the result is a much more powerful and expressive form of FMEA that provides the analyst with a greater degree of understanding of the system failure behaviour, therefore making it easier to decide which parts of the system need to be improved to increase reliability and safety.

3 INTRODUCING SEQUENCES TO FMEA

HiP-HOPS allows us to create FMEAs that take into account the effect of multiple failure events occurring in combination, but what if the sequence in which they occur is also important? This section describes how HiP-HOPS has been extended to be able to analyse this sort of dynamic behaviour, which is increasingly important in complex systems.

For example, consider the simple system in Figure 1, consisting of two components – a primary and a standby – and a switch that activates the standby when the primary fails.

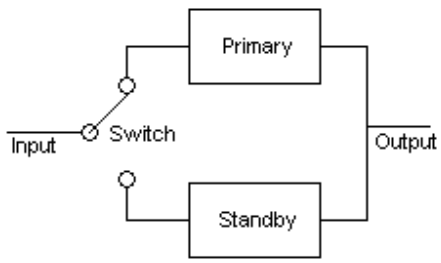


Fig 1: Simple system with dynamic failure behaviour

This behaviour should allow the system to continue operating even if one component fails. If we were to analyse this system using classical FMEA techniques, we might produce results such as those in Table 1:

Component	Failure	Effect
Primary	ComponentFault	Causes stand-by to activate
Switch	SwitchFault	Standby will not activate
Standby	ComponentFault	System failure

Table 2: Simple FMEA

However, these results do not truly represent the failure behaviour of the system. Although it indicates that a failure of the switch will mean the standby does not activate, it gives no indication of the sequence of events – and in this case, the sequence of events can make a large difference. If the switch fails *after* the primary has failed, then it has no effect on the system because the standby has already been activated; whereas if it fails first, before the primary fails, then it will lead to a system failure because the standby will not be activated. Thus the results in Table 1 are incomplete or at best pessimistic.

It would be better if we had some way of representing the sequence of events to better model the dynamic behaviour of the system. One approach is the Dynamic Fault Tree methodology [9, 10]. DFTs use a set of special logic gates such as FDEP (functional dependency) and SEQ (sequence) to describe the dynamic behaviour of a system. However, the DFT methodology is designed for quantitative fault tree analysis rather than the primarily qualitative analysis approach used in an FMEA, which means that DFTs are not the best solution to this problem. Another option is to represent sequences of events as events themselves – a kind of conditional event. For example, to represent the situation where the switch fails first, we could define an event such as "SwitchFailure AND PrimaryFailure AND SwitchFailure BEFORE PrimaryFailure". This kind of approach is used in [11] and [12], but because the dynamic behaviour is encapsulated in a separate event instead of being part of the logic that links the failure events, it is hidden in the context of any logical analysis, e.g. calculation of minimal cutsets. This, however, can be problematic, as any two such hidden temporal relations may share events, in which there is potential for logical reduction or identification of conflicting temporal orders (e.g. A BEFORE B and B BEFORE A). However, analysis of such relations is simply impossible in this approach.

Instead, we can use Pandora [8, 13]. Pandora is a temporal extension to Boolean logic that allows us represent sequences of failures as well as combinations of failures. It does this by introducing a set of new logical

operators that indicate a sequential relationship between failure events.

The first of these operators is the Priority-AND or PAND, which represents a "BEFORE" relationship. X PAND Y means that X occurs before Y occurs. It is a subset of the AND operator and therefore both X and Y must occur. The PAND applies only when events do not occur at the same time; to represent a simultaneous occurrence of events, Pandora offers the Simultaneous-AND or SAND. The SAND is true if all of its input events occur at the same time. Although in the general case this might be considered a very rare occurrence, it can be useful in modelling multiple failures with a common cause, where the probability of simultaneous occurrence is significant and cannot be ignored. Finally, there is also the Priority-OR gate. This also represents a "BEFORE" relationship, but unlike the PAND, only the first event must occur; e.g. X POR Y means that when X occurs, Y must not have occurred yet. A POR is true regardless of whether the subsequent events occur or not, whereas a PAND is true only if all of its input events occur.

A summary of the three operators is given in Table3.

	PAND	**SAND**	**POR**
Symbol	<	&	|
Meaning of X *op* Y	X occurs before Y, both occur	X occurs at the same time as Y	X occurs first; Y may or may not occur

Table 3: Temporal operators

These "temporal operators" are only relative operators, concerned with the relative order in which events occur and not with the exact time of occurrence. The sequence of events can be represented by a *sequence value*, indicated by S, e.g. S(X) is the sequence in which X occurred relative to the other events. The sequence value is an integer; zero (0) is a special case and indicates that the event did not occur, while any positive number means the event occurred and the value gives the order. So if S(X) = 1 and S(Y) = 2, then X occurred first and Y second. Two events occurring simultaneously will have the same sequence value. The sequence value can be seen as a refinement of the Boolean value of an event; a sequence value of 0 is equivalent to false and a sequence value greater than 0 is equivalent to true.

Operators also have sequence values and can be seen as compound events whose values, both logical and sequential, depend on the values of their inputs. For example, an OR gate (symbol "+") is true as long when at least one of its input events occurs; therefore it will have the same sequence value as the first input event to occur, or 0 if no events occur. Similarly, an AND gate (symbol ".") is true when all of its input events have occurred and so has the same sequence value as the last input event to occur.

Assigning both events and operators in this way allows us to produce *temporal truth tables* or TTTs, which are analogous to normal Boolean truth tables but extended to use sequence values instead of just true and false. TTTs make it easier to see the behaviour of expressions containing temporal operators and also make it possible to demonstrate equivalence between temporal expressions: if two expressions have identical TTTs, then those expressions are equivalent, just as identical Boolean truth tables indicate equivalence.

A TTT showing the sequence values of the five operators in Pandora is shown in Table 4:

X	**Y**	**X+Y**	**X.Y**	**X<Y**	**Y<X**	**X&Y**	**X|Y**	**Y|X**
0	0	0	0	0	0	0	0	0
0	1	1	0	0	0	0	0	1
1	0	1	0	0	0	0	1	0
1	1	1	1	0	0	1	0	0
1	2	1	2	2	0	0	1	0
2	1	1	2	0	2	0	0	1

Table 4: Temporal Truth Table for the operators

Note that the PAND and POR gates are not commutative; X PAND Y is not the same as Y PAND X, since the sequence of events is different in each case. However, the SAND is commutative since all its input events occur simultaneously.

As with normal Boolean logic, Pandora also supplies a number of rules or laws that govern the behaviour of the three temporal operators and describe how they relate to the existing Boolean operators AND and OR. These laws fall into two categories: versions of existing Boolean laws extended to apply to the temporal operators and new laws designed to deal with the consequence of representing sequences.

In the first category, there are new versions of the Absorption, Distributive, Idempotent and Associative laws, amongst others. These do not always apply in the same way as their original Boolean versions; for example, PAND is not left distributive over OR, but it is right distributive:

$$(Y + Z) < X = (Y < X) + (Z < X)$$

$$X < (Y + Z) = (X \mid Y) \,.\, (X \mid Z) \,.\, (Y + Z)$$

Conversely, the POR is not right distributive over OR but is left distributive:

$$(Y + Z) \mid X = (Y|X) \,.\, (Z|X)$$

$$X \mid (Y + Z) = (X|Y) + (X|Z)$$

Some of the most useful laws are the Absorption Laws, as these allow us to reduce and simplify expressions. For example:

$$X \,.\, (X<Y) = X<Y \qquad Y \,.\, (X<Y) = X<Y$$

$$X \,.\, (X\&Y) = X\&Y \qquad Y \,.\, (X\&Y) = X\&Y$$

$$X \,.\, (X|Y) = X|Y \qquad Y \,.\, (X|Y) = X<Y$$

These Conjunctive Absorption Laws enable us to remove repeated events from within a cut set, while the Disjunctive Absorption Laws:

$$X + (X<Y) = X \qquad Y + (X<Y) = Y$$

$$X + (X\&Y) = X \qquad Y + (X\&Y) = Y$$

$$X + (X|Y) = X \qquad Y + (X|Y) = X + Y$$

allow us to remove redundant cut sets.

There are also several new laws to handle the implications of introducing sequences to the logic. One such implication is the possibility of *contradictions*. For example, the expression (X<Y).(Y<X) is a contradiction; it is not possible for X to occur before Y AND for Y to occur before X, so this expression will always evaluate to false, i.e. it can never occur.

This type of scenario is detected by the Laws of Mutual Exclusion, which state that only one temporal relation (before, after, or simultaneous) is possible at once. Another example is:

$$(X<Y) \,.\, (X\&Y) = 0$$

Other contradictions are highlighted by the Laws of Simultaneity, which are versions of the Idempotent law specific to the temporal operators:

$$X<X = 0 \qquad X|X = 0 \qquad X\&X = X$$

There are also the Laws of Extension, which reveal implicit temporal relationships between events. for example, one Law of Extension states that:

$$(X<Y) \,.\, (Y<Z) = (X<Y) \,.\, (Y<Z) \,.\, (X<Z)$$

In other words, if X is before Y, and Y is before Z, then X is also before Z. The same applies to the other operators too. Applying the Law of Extension is important in particular because it can reveal hidden contradictions that would not otherwise be detected by the Laws of Mutual Exclusion or Simultaneity; for example, it is possible to have circular redundancies, where a sequence of events forms a chain:

$$(X < Y) \,.\, (Y < Z) \,.\, (Z < X)$$

This does not immediately appear to be a contradiction, but applying the Law of Extension reveals its true nature:

$$(X<Y) \,.\, (Y<Z) \,.\, (Z<X) \,.\, (X<Z) \,.\, (Y<X) \,.\, (Z<Y)$$

This clearly violates the Law of Mutual Exclusion and so there must be a contradiction, since we have both (X<Y) and (Y<X).

Finally, there are the three Completion Laws. These laws are particularly important because they link the new temporal operators to the existing AND and OR operators from Boolean logic, showing how Pandora provides an extra layer of precision allowing us to represent when, and not just if, events occur. These relationships can best be seen from a diagram:

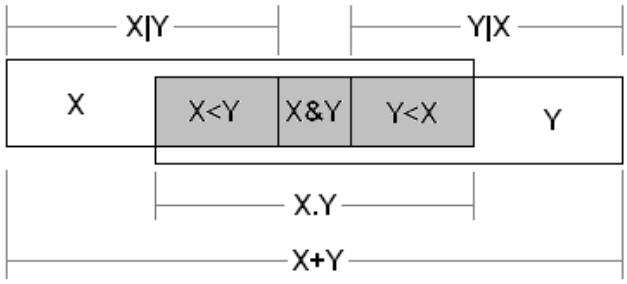


Fig 2: Completion Laws

This set diagram shows how the temporal gates are subsets of the logical gates. The rectangle X represents the situations where X occurs and the rectangle Y represents situations where Y occurs; the shaded area represents the overlap, i.e. X AND Y, and can be exactly subdivided into three parts: X < Y, X &Y, and Y < X. This is because if two events X and Y both occur, then either X occurred before Y, Y occurred before X, or they both occurred simultaneously, and only one of these three possibilities can be true at once (as stated by Mutual Exclusion). This is the Conjunctive Completion Law:

$$X.Y = (X<Y) + (X\&Y) + (Y<X)$$

There is a similar relationship between OR, POR, and SAND: the area taken up by X+Y can be exactly subdivided into X&Y, X|Y, and Y|X, and this is the Disjunctive Completion Law:

$$X + Y = (X|Y) + (X\&Y) + (Y|X)$$

This is because X|Y covers both X<Y and the case where X happens but Y does not, and similarly for Y|X.

Finally there is the Redundant Completion Law, which shows how an event can be temporally redundant:

$$X = (Y<X) + (X\&Y) + (X|Y)$$

In this case we can remove an event (i.e. Y in the above) entirely because its occurrence in one of the specified orders in relation to another event (i.e. X) has no influence on the outcome.

It is important to note that all the laws can be proven using TTTs. For example, the Redundant Completion

Law, with the equivalence highlighted in bold, is shown in Table 5:

X	Y	Y<X	X&Y	X|Y	(Y<X) + (X&Y) + (X|Y)
0	0	0	0	0	**0**
0	1	0	0	0	**0**
1	0	0	0	1	**1**
1	1	0	1	0	**1**
1	2	0	0	1	**1**
2	1	2	0	0	**2**

Table 5: Proving the Redundant Completion Law

Temporal expressions containing the Pandora operators can be entered in place of normal Boolean expressions in HiP-HOPS. For example, going back to the simple standby example in Figure 1, we might define the following HiP-HOPS expressions:

- O-System = O-Primary. O-Standby
- O-Primary = PrimaryFault
- O-Standby = O-Switch + StandbyFault
- O-Switch = SwitchFault < O-Primary + SwitchFault & O-Primary

Here we can use a PAND and a SAND to indicate that an omission (failure class O) of the switch signal is only relevant if it occurs before or at the same time as an omission from the primary component.

HiP-HOPS then joins these together to produce a set of "temporal" fault trees – fault trees that contain temporal gates. However, because of these temporal gates, it is now necessary to use more complex methods to reduce the fault trees and calculate the minimal cut sets (and thus produce the FMEA). Whereas there are only a few important Boolean laws to use during normal fault tree analysis, there are many possible temporal laws that can be employed.

The end result are the *minimal cut sequences* (MCSQs) for the tree – the smallest possible sequences of events necessary to cause the system to fail. Each one is a conjunction of events or of sequences of two events (so a sequence like X < Y < Z would become X<Y . Y<Z . X<Z instead). These can then be used to produce a temporal FMEA. This differs from a normal FMEA or even the combinatorial FMEA mentioned earlier by the presence of a sequence column, which indicates the constraining sequence in which the further effects must occur.

Staying with the example system from Fig 1, the temporal FMEA produced by the HiP-HOPS tool would be as shown in Table 6. This FMEA shows not only what the effects of combinations of failures would be (e.g. short circuits in both primary and standby components) but also what the effects would be when the events occur in different sequences.

Comp-onent	Failure	Further Failures	Sequence (if any)	Effects
Primary	Primary Fault	Standby Fault	Any	Omission of output
		Switch Fault	SwitchFault < Primary Fault	Omission of output
		Switch Fault	SwitchFault & Primary Fault	Omission of output
Switch	Switch Fault	Primary Fault	SwitchFault < Primary Fault	Omission of output
		Primary Fault	SwitchFault & Primary Fault	Omission of output
Standby	Standby Fault	Primary Fault	Any	Omission of output

Table 6: Temporal FMEA

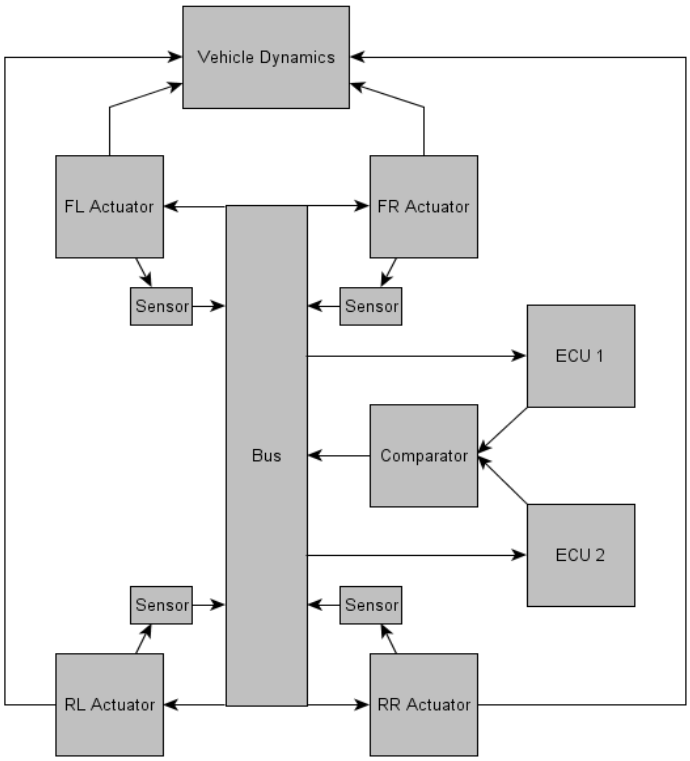
Notice that this time, if the switch fails after the primary, it does not lead to an omission of system output. This sequence is not included in the FMEA table, because it has no effect in the system. If however the tool was instructed to derive separately the effects on the two outputs of primary and standby, only omission of primary would have been derived as effect.

4 CASE STUDY

Figure 3 shows a simplified model of a vehicle braking system. It consists of four brake actuators, one at each wheel, and each with a connected rotation sensor. The actuators are controlled via a bus, which also carries the signals from the sensors. These signals are fed into a pair of electronic control units that control the brakes. The output of both ECUs must agree (as determined by a comparator) for the braking commands to be sent to the actuators. There is also a "vehicle dynamics" component which is a virtual component representing the effect the brakes have on the handling of the vehicle. This can be thought of as the "output" of the system.

The first step is to provide local failure data for each of the components. Since all four brake actuators are identical, we can focus on just one of these, and for the purposes of this case study we will consider only *permanent* commission failures (represented by C for commission), which in the context of braking can cause the wheels of the vehicle to lock. Note that the model and its annotations have been greatly simplified to reduce the analysis to a manageable size and thereby focus on the principles of representation and analysis of failure sequences; as such, there is little propagation of

failure among components and, therefore, causes link almost directly with the effects.



ACTUATOR

Commission failure of the actuator is caused by communication or brake signal commission failure:

- C-Actuator = ActCom + C-brakeSignal

BUS

Commission failure of brake signal is caused by communication or brake signal commission failure:

- C-brakeSignal = BusCom + C-brakeSignal

COMPARATOR

Commission failure of brake signal is caused by communication or brake signal commission failure:

- C-brakeSignal = C-ECU1 . C-ECU2

ECU

Commission failure of output is caused by sensor commission failure:

- C-output = C-sensor

SENSOR

Commission failure of sensor is caused by internal sensor failure:

- C-sensor = SensHigh

Figure 3: Braking system model and failure data

Failure annotations show that a permanent commission of braking pressure (leading to locked wheels) is caused either by a failure of the actuator (commission of the brake) or a commission of the signal to the brake. This in turn is caused either by a bus failure (for example, memory stuck failures that could corrupt messages and eventually result in maximum pressure applied to all wheels) or by a commission of the brake signal from the comparator; for this to happen, there has to be a commission from both ECUs. As the ECUs rely on readings from the same sensor to control each brake, if the sensor fails high a locked wheel can be perceived by both ECUs as having normal rotation and both ABS algorithms will fail to react. Although in practice there will be plausibility checks to help prevent this scenario, we assume that there are still circumstances in which an undetectable sensor bias can arise, thus leading to an incorrect response from the ECU.

These annotations can be used to synthesise a simple fault tree for the event of permanent commission of braking (C-Actuator), which is the same for each wheel. This fault tree has only three minimal cutsets:

- ActCom
- BusCom
- SensHigh

At this point, we can look at the vehicle dynamics where the effect of sequencing of failures becomes evident. For the purposes of this case study, we examine the effect of the left two brakes (Front-Left and Rear-Left) locking in different sequences; the analysis could of course be extended to take into account not only other pairs of brakes, but also combinations of three or even four brakes locking. The three scenarios therefore are:

1. C-FL Actuator < C-RL Actuator
2. C-FL Actuator & C-RL Actuator
3. C-RL Actuator < C-FL Actuator

Since the brakes are symmetrical, we need only look at the first two; the MCSQs for the third will be the same as the first, except with the order reversed.

Minimal Cut Sequences for Scenario 1

For scenario 1, the expression is as follows:

(ActComFL + BusCom + SensHighFL) < (ActComRL + BusCom + SensHighRL)

Next we apply a Distributive law to obtain:

ActComFL < (ActComRL + BusCom + SensHighRL) + BusCom < (ActComRL + BusCom + SensHighRL) + SensHighFL < (ActComRL + BusCom + SensHighRL)

We can eliminate the BusCom disjunction using Simultaneity. First we expand it using the law X<(Y+Z) = X|Y . X|Z . (Y+Z)

BusCom |ActComRL . BusCom |BusCom . BusCom |SensHighRL . (ActComRL + BusCom + SensHighRL)

BusCom|BusCom violates Simultaneity and is a contradiction, since an event cannot happen before itself. This is then equivalent to:

BusCom |ActComRL . FALSE . BusCom |SensHighRL . (ActComRL + BusCom + SensHighRL)

And because of the Boolean law X . FALSE = FALSE, the entire conjunction reduces to just FALSE. Then using the Boolean law X + FALSE = X, we are left with just:

ActComFL < (ActComRL + BusCom + SensHighRL) + SensHighFL < (ActComRL + BusCom + SensHighRL)

Next we expand this to get:

ActComFL|ActComRL . ActComFL|BusCom . ActComFL|SensHighRL . (ActComRL + BusCom + SensHighRL) +

SensHighFL|ActComRL . SensHighFL |BusCom . SensHighFL |SensHighRL . (ActComRL + BusCom + SensHighRL)

and using the Boolean distributive law, we can expand this to six disjunctions, the first of which is:

ActComFL|ActComRL . ActComFL|BusCom . ActComFL| SensHighRL. ActComRL

Then using a temporal Absorption law, Y . X|Y = X<Y, we are left with six MCSQs:

- ActComFL<ActComRL . ActComFL|BusCom . ActComFL|SensHighRL
- ActComFL|ActComRL . ActComFL<BusCom . ActComFL|SensHighRL
- ActComFL|ActComRL . ActComFL|BusCom . ActComFL<SensHighRL
- SensHighFL<ActComRL . SensHighFL|BusCom . SensHighFL|SensHighRL
- SensHighFL|ActComRL . SensHighFL<BusCom . SensHighFL|SensHighRL
- SensHighFL|ActComRL . SensHighFL|BusCom . SensHighFL<SensHighRL

In other words, the front left brake locking before the rear left brake is caused either by the actuators failing in that order (FL < RL), the sensors failing in that order, the FL sensor failing before the RL actuator, or the FL actuator failing before the RL sensor. It is important to note that POR gates in these results show that sequencial wheel locking does not occur when a bus commission (BusComm) failure happens first. This is because a bus commission is a common cause failure and will cause simultaneous locking of all wheels. However, the results show that it is possible for the FL sensor or actuator to fail first and then a bus failure occurs, leading to sequential locking of the two wheels.

Minimal Cut Sequences for Scenario 2

For scenario 2, where both brakes fail simultaneously, the starting expression is:

(ActFaultFL + BusFault + SensHighFL) & (ActFaultRL + BusFault + SensHighRL)

And ultimately this reduces down to five minimal cut sequences:

- ActComFL & SensHighRL
- ActComRL & SensHighFL
- ActComFL & ActComRL
- SensHighFL & SensHighFR
- BusCom

Since a commission failure of the bus is a common cause failure that affects all four brakes, this will always lead to a simultaneous failure regardless of what occurs afterwards.

Results

With these MCSQs, we can now build a temporal FMEA, a part of which (focusing only on the front-left wheel and the bus) is shown in Table 7.

In this FMEA, various combinations of component failures cause system level effects –E1 – E4 that represent the following conditions:

- E1 – Front left brake locks. This is not critical, as three wheels are unaffected, and supporting systems are able to mitigate some of the deceleration and yaw.
- E2 – Caused by Scenario 1: front left brake locks followed by rear left brake. This is relatively critical because it may cause the vehicle to veer sideways into oncoming traffic (assuming vehicles drive on the right).
- E3 – Caused by Scenario 2: both wheels lock simultaneously. This is very critical as the driver will be surprised by a large yaw torque and sudden braking. Ability to automatically compensate is limited.
- E4 – Caused by Scenario 3: front left brake locks after the rear left brake. This has limited criticality as stability is initially reduced but not lost as one rear wheel still has traction.
- The system failure of the rear-left brake locking on its own is not included in Table 7 but would also be present in the FMEA.

Comp-onent	Failure	Further Failures	Sequence (if any)	Effects
FL Actuator	Com-mission	None	None	E1
		RLActuator. Commission	FLActuator < RLActuator	E2
			FLActuator & RLActuator	E3
			RLActuator < FLActuator	E4
		RLSensor. SensHigh	FLActuator < RLSensor	E2
			FLActuator & RLSensor	E3
			RLSensor < FLActuator	E4
		Bus. Commission	FLActuator < Bus.Comm	E2
FL Sensor	Sens High	None	None	E1
		RLActuator. Commission	FLSensor < RLActuator	E2
			FLSensor & RLActuator	E3
			RLActuator < FLSensor	E4
		RLSensor. SensHigh	FLSensor < RLSensor	E2
			FLSensor & RLSensor	E3
			RLSensor < FLSensor	E4
		Bus. Commission	FLSensor < Bus.Comm	E2
Bus	Com-mission	None	None	E3

Table 7: Temporal FMEA results

The ability to incorporate sequential and combinatorial information has increased the amount of information provided by the FMEA. The results show how the sequence in which failure events occur can have different effects – and different levels of criticality.

A traditional single-failure FMEA would only give results for single brake failures and common causes like a commission of the bus; an attempt to exhaustively examine the effect of combinations of component failures would have required enumeration and assessment of all possible combinations which for anything beyond trivial is an intractable task. Furthermore, analysis of the effect of sequences of component failures would have been impossible using a classical approach, and yet it is the effects of such sequences (in terms of the different types of multiple brake failures) that are most interesting and critical.

Overall, temporal FMEAs such as that of Table 7 provide a greater degree of precision in the results than classical single failure FMEAs. Being able to analyse not only combinations of failures but also sequences of failures allows a deeper insight into how a system can fail, even in simple systems like the braking system presented here.

In this particular example, we have shown that the information gleaned from temporal and combinatorial analyses can reveal considerable differences in the criticality of effects, and yet that information could be produced from only a small set of simple logical expressions for each component in the system.

CONCLUSION

Classical FMEA is a useful technique when determining the reliability and safety of a system, but it is not without its faults. Chief amongst these are its inability to analyse combinations of failures and the lack of support for considering the effects of sequences of events. However, sometimes the sequence of events can make a major difference to the criticality of a system failure or even reveal that a system failure may not occur at all if events occur in a certain sequence. Furthermore, as shown by the case study, the ability to consider multiple failures and sequences allows us to analyse more complex outcomes involving multiple system failures, giving us a better insight into the ways the system can fail and what effects those failures have on the safety of the system.

This sequential capability is provided by Pandora, which allows analysts to represent the sequences of failures using three temporal operators. HiP-HOPS can then analyse temporal expressions containing those operators automatically, by applying temporal laws to reduce them, and therefore obtain the minimal cut sequences. These can then be used to produce a temporal FMEA that contains both the effects of combinations of failures and of sequences of failures. Because the FMEA is generated via fault tree analysis, a deductive method, it minimises the problems of combinatorial explosion and allows the analysis to take place automatically.

The result is a more comprehensive analysis of the system, which provides the analyst with better information with which to estimate the safety and reliability of the system. Due to the automation and speed of analysis that HiP-HOPS provides, it also makes it possible to apply these more detailed analyses iteratively as part of the design process, thereby detecting design flaws earlier and helping to produce more reliable and safer systems.

ACKNOWLEDGMENTS

This work was supported by the EU Projects SAFEDOR (Grant IP-516278) and ATTESST-2 (Grant 224442)

REFERENCES

1. Collins M.A., Handley S., Collins R.J. Automated Abstraction of Failure Effects Using a Formal, Hierarchical, Hardware Specification Language. MSc Thesis, Cranfield University, 1994.
2. Bull D.R., Burrows C.R, Edge K. A., Hawkins P. G., Woollons D.J. "A tool for FMEA of hydraulic systems." IMECE '96, International Mechanical Engineering Congress and Exposition, Atlanta, Georgia, 1996.
3. Barnard R.F., Dohanich S.L., Heinlein P.D. "System for failure mode and effects analysis." *Computer Integrated Manufacturing Systems* 10(2):172. 1997.
4. Hawkins P.G., Atkinson R.M., Woollons D.J., Bull D.R., Burrows R. "An approach to FMEA using multiple models." IFMAA'96, International Functional Modelling Application Association Conference, Athens, 1996.
5. Hawkins P.G., Woollons D.J. "Failure modes and effects analysis of complex engineering systems using functional models." *Artificial Intelligence in Engineering* 12: 375-397, 1998.
6. Lehtela M. "Computer-Aided FMEA of Electronic Circuits." *Microelectronics and Reliability*, 30(4):761-773, 1990.
7. Parker D., Walker M., Papadopoulos Y., Grante C. "Component-Based, Automated FMEA of Advanced Active Safety Systems." FISITA'06, 31st World Automotive Congress, Yokohama, Published by JSAE, ISBN: 4-915219-83-6, 2006.
8. Walker M.D., Bottaci L., Papadopoulos Y. "Compositional Temporal Safety Analysis." 26th International Conference on Computer Safety, Reliability and Security (SAFECOMP'07), LNCS 4680:105-119, Springer, 2007.
9. Dugan J., Venkataraman B., Gulati R. "DIFtree: A software package for the analysis of dynamic fault tree models." Annual Reliability and Maintainability Symposium, Philadelphia, USA. 13-16 January 1997.
10. Vesely W.E., Stamatelatos M., Dugan J.B., Fragola J., Minarick J., Railsback J. Fault Tree Handbook with Aerospace Applications. NASA Office of Safety and Mission Assurance, USA, 2002.
11. Gorski J., Wardzinski A. "Deriving Real-Time Requirements for Software from Safety Analysis." Proceedings of the 8th Euromicro Workshop on Real-Time Systems, pp. 9-14. IEEE CS Press. 1996
12. Hansen K.M., Ravn A.P. "From Safety Analysis to Software Requirements." *IEEE Transactions on Software Engineering*, 24(7):573-584. 1998
13. Walker M.D., Papadopoulos Y.I. "PANDORA 2: The time of Priority OR gates." DCDS'07, 1st IFAC Workshop on Dependable Control of Discrete Event Systems, Paris, 13-15 June 2007.
14. Domis D., Trapp M. "Integrating Safety Analyses and Component-Based Design" 27th International Conference on Computer Safety, Reliability and Security (SAFECOMP'08), LNCS 5219:58-71, Springer, 2008.
15. Chen d., Johansson R, Lönn H., Papadopoulos Y., Sandberg Y., Törner F., Törngren M., "Modelling Support for Design of Safety-Critical Automotive Embedded Systems", 27th International Conference on Computer Safety, Reliability and Security (SAFECOMP'08), LNCS 5219:72-85, Springer, 2008

DEFINITIONS, ACRONYMS, ABBREVIATIONS

DFT Dynamic Fault Tree

FMEA Failure Modes & Effects Analysis

FTA Fault Tree Analysis

H/W Hardware

MCS Minimal cut set – the smallest combination of events necessary to cause a system failure. These are the results of a qualitative FTA.

MCSQ Minimal cut sequence – the smallest sequence of events necessary to cause a system failure. These are the results of a qualitative FTA in Pandora.

PAND Priority-AND – represents a BEFORE relationship between two events. Both must occur.

POR Priority-OR – represents a BEFORE relationship between two events. Only the first need occur.

SAND Simultaneous-AND – represents the situation where multiple events occur at the same time.

S/W Software

TTT Temporal Truth Table – a logic table used to show sequence values of a temporal expression.