Reasoning with failures

Hamid Jahanian\textsuperscript{1} and Annabelle McIver\textsuperscript{2}

\textsuperscript{1} hamid.jahanian@hdr.mq.edu.au
\textsuperscript{2} annabelle.mciver@mq.edu.au

Abstract. Safety Instrumented Systems (SIS) protect major hazard facilities, e.g. power plants, against catastrophic accidents. An SIS consists of hardware components and a controller software – the “program”. Current safety analyses of SIS include the construction of a fault tree, summarising potential faults of the components and how they can arise within an SIS. The exercise of identifying faults typically relies on the experience of the safety engineer. Unfortunately the program part is often too complicated to be analysed in such a “by hand” manner and so the impact it has on the resulting safety analysis is not accurately captured. In this paper we demonstrate how a formal model for faults and failure modes can be used to analyse the impact of an SIS program. We outline the underlying concepts of Failure Mode Reasoning and its application in safety analysis, and we illustrate the ideas on a practical example.

1 Introduction

Plant accidents can have catastrophic consequences. An explosion at a chemical plant in eastern China killed over 70 people and injured more than 600 in 2019. Safety Instrumented Systems (SIS) are protection mechanisms against major plant accidents \cite{16}. Failure of SIS components can result in the SIS being unavailable to respond to hazardous situations. It is therefore crucial to analyse and address such failures. A typical SIS comprises physical components to interact with plant, and a software program\textsuperscript{3} that analyses the information and initiates safety actions. Such software can be highly complex, and even when it is not itself faulty still propagate input faults from the sensors to the safety actuators. This paper concerns a current omission in the standard safety engineering process: that of an accurate fault analysis of complex SIS program.

Well established methods, such as Fault Tree Analysis (FTA), already exist in the industry for analysing and quantifying SIS failure modes \cite{31}. FTA is a deductive method that uses fault trees for quantitative and qualitative analysis of failure scenarios. A fault tree is a graphical representation of the conditions that contribute to the occurrence of a predefined failure event. A fault tree will be created by a safety analyst and based on their knowledge and understanding of the failure behaviours in a system. Not only are such by-hand analyses inherently subject to human-error, they also require expertise, time and effort.

\textsuperscript{3} In this paper the term program refers to the software code run by SIS CPU; also known in safety standards as SIS Application Program \cite{16}. 
A new method, Failure Mode Reasoning (FMR), was recently introduced to circumvent the need for by-hand analysis of parts of SIS \cite{17}. Using a special calculus built on failure modes, FMR analyses the SIS program to identify the hardware faults at SIS inputs that can result in a given failure at its output. The main outcome of FMR is a short list of failure modes, which can also be used to calculate the probability of failure. In this paper we show how to use ideas from formal methods to justify FMR. We use an abstraction to model failures directly, and we show that such an abstraction can be used to track failures within the SIS program so that potential output failures can be linked to the potential input failures that cause them. We prove the soundness of the technique and illustrate it on a practical example.

The rest of this paper is organised as follows: Section 2 provides a brief explanation of the context and how FMR can enhance safety analysis. Section 3 formalises the underlying ideas of analysis of failures for SIS programs. Based on these concepts, Sections 4 and 5 formulate the concepts for composing the individual elements in FMR and the reasoning process on the interactions between these elements. Section 6 includes descriptions of how FMR is applied in practice and in particular in large scale projects. Finally Sections 7 and 8 wrap up the paper with a review of FMR’s position with respect to other research works and potential research in future.

2 SIS and FMR

An SIS consists of sensors, a logic solver, and final elements. The sensors collect data about the environment (such as temperature and pressure) and the logic solver processes the sensor readings and controls the final elements to intervene and prevent a hazard. Such interventions can include shutting down the (industrial) plant and they are referred to as Safety Instrumented Functions (SIFs). Fig. 1b illustrates a simple SIS consisting of two sensors, one logic solver and one final element. This SIS performs only one SIF, which is to protect the downstream process against high pressure in the upstream gas pipe. The sensors measure the gas pressure and the logic solver initiates a command to close the valve if the gas pressure exceeds a threshold limit.

SIS faults are typically modelled by using fault trees. For an accurate analysis a fault tree must reflect all potential faults caused by all components in the SIS. Clearly incorrect sensor readings are a significant factor in safety analysis as they can lead to hazardous scenarios. One of the problems in safety analysis is to understand how such deviations can be propagated by the SIS program and lead to faults at SIS outputs. If done by hand, such understandings depends critically on the analyst’s knowledge of the details of the SIS program.

Consider, for example the fault tree in Fig. 1a, which is meant to summarise the failures of SIS in Fig. 1b; the SIS fails if both sensors fail or if the logic solver fails or if the final element fails. The fault tree is built on the assumption that the two sensors provide redundancy, which means that provided one of the two sensors is in a healthy state, that is sufficient to detect potential hazards.
However, the validity of this assumption, and thus the validity of the fault tree, directly depends on the details of SIS program and how it computes the output from the input. For example, if the two inputs from sensors are averaged first and then compared to the high pressure limit as shown in Fig. 2a, the proposed fault tree (Fig. 1a) is incorrect; because failure of one sensor will affect the average of the two. But if each sensor reading is separately compared to the threshold limit first (as in Fig. 2b), the sensors can be considered redundant and the fault tree would summarise the failures accurately. While the two programs deliver the same functionality, they do not show the same failure behaviour; and the proposed fault tree can correspond to only one of them.

For variables $i_1, i_2, w \in \mathbb{R}$ and $v, z, o \in \mathbb{B}$, and parameter $K \in \mathbb{R}$:

\[
  w = \text{Avg}(i_1, i_2) = \frac{i_1 + i_2}{2}, \\
  o = \text{Gcom}_K(w) = (w > K) \text{ and } o = \text{Or}(v, z) = v \lor z.
\]

Fig. 2: Two possible implementations for the Logic Solver in Fig. 1a

In real world scenarios, SIS programs are large and complex. It is not unusual for a typical SIS program to have hundreds of inputs like $i_1$ and $i_2$ and thousands of Function Blocks like $\text{Avg}$ and $\text{Gcom}_K$. Conducting a detailed analysis of program of such scales will be a real challenge for a human analyst, but it nonetheless plays a crucial part in producing accurate results. In such scenarios an automated method such as FMR can be of a great help.

FMR is a technique for enabling the details of SIS programs to be accurately reflected in the full safety analysis of a system. The challenge we address is identifying the SIS input “failure modes” that cause a given SIS output failure
mode by analysing the SIS program that reads those inputs and produces that output. The results can then be incorporated in an overall safety analysis of SIS.

SIS programs are commonly developed in the form of Function Block Diagrams (FBD) [15]. Fig. 2 showed two very simple examples of FBDs. An FBD consists of function blocks and their interconnections, which we label with variable names. In Fig. 2a, \( o, w, i_1 \) and \( i_2 \) are the variables and \( \text{Avg} \) and \( \text{Gcom}_K \) are the function blocks. We will use this FBD as a worked example through this paper to demonstrate the FMR process.

The SIS program given at Fig. 2a is supposed to initiate a command to close the gas valve when the pressure rises above a given threshold. In normal circumstances, when all inputs report correct measurements from the sensor readings, an output of \( t \) causes the correct shut down command to be delivered when the pressure is high. Suppose however that the inputs \( i_1, i_2 \) are incorrectly recording the pressure. These inaccuracies propagate through the program and lead to an \( f \) at the output, meaning that the SIS will not initiate the safety action required to prevent the hazardous event.

In simple terms, this is how FMR analyses such output deviations: from \( o \) being \( f \) by fault we can conclude that \( w \) must be less than the threshold limit set in \( \text{Gcom}_K \): \( (o = f) \Rightarrow (w \leq K) \). Sentence \( (w \leq K) \) in turn implies that the average value of \( i_1 \) and \( i_2 \) must be less than the threshold limit: \( (w \leq K) \Rightarrow ((i_1 + i_2)/2 \leq K) \). Assuming that this is due to an input fault, we can conclude that either input \( i_1 \) must be reading lower than what it should, or input \( i_2 \).

Overall, we can conclude:

\[
(o \text{ being } f \text{ by fault}) \Rightarrow (i_1 \text{ reads too low}) \lor (i_2 \text{ reads too low})
\]

Notice that the actual values of inputs are not required, but only their categories in terms of whether they are “too high”, or “too low”. It turns out that we can take advantage of this abstraction to simplify the overall analysis. In the next section we describe a simple model of failures from which we derive an analysis that uses “failure modes” explicitly.

FMR completes the SIS safety analysis by incorporating the functionally most important part of the system – the program, and it does this by analysing the actual program rather than a synthesised model. The process is automated and thus it saves time and effort, and offers accuracy and certainty. The purpose of FMR is similar to fault tree analysis, but it adds rigour to the consideration of fault propagation in the SIS program.

3 Modelling failures

In this section we formalise the ideas underlying the identification and analysis of potential failures for SIS programs. In particular the result of the analysis should be the identification of potential faults and, in addition, to categorise

\footnote{Note that in Fig. 2b the FMR analysis would produce a different result, i.e. \( (o \text{ being } f \text{ by fault}) \Rightarrow (i_1 \text{ reads too low}) \land (i_2 \text{ reads too low}) \).}
them in terms of their “modes of failure”. This is an essential step in any safety engineering exercise.

In what follows we use well known constructions from relational-style modelling. Our contribution is to apply those ideas in this new setting for SIS programs. Let $V$ be an abstract state space; we use $P \times V$ for the power set over $V$. A partition of a set $X$ is a set of pairwise non-intersecting subsets in $P \times X$.

We begin with a simple abstract model for a generic SIS function. It is a function which takes inputs to outputs over an (abstract) type $V$.

**Definition 1.** An abstract model for an SIS function is a function of type $V \rightarrow V$.

An SIS function can be a function block (FB), a combination of FBs or the entire SIS program. As described above, the safety analyst can only access information about the safety status of the system through the SIS program variables. The challenge is that this reported status (i.e. the sensor readings) might be inaccurate or wrong. To model such faults we need to keep track of the values recorded in the SIS program variables and the value that should have been reported. When these values are not the same we say that there is a fault. The next definition shows how to keep track of these faults within a particular SIS setting.

**Definition 2.** Given an SIS function $f : V \rightarrow V$, a failure model is a function $\langle f \rangle : V^2 \rightarrow V^2$ defined by

$$\langle f \rangle (m, a) := (f(m), f(a)).$$

For the pair $(m, a) \in V^2$, the first component $m$ models the value reported by the SIS program variables, and the second component $a$ is the actual value that should be reported. We say that $(m, a)$ is a failure state whenever $m \neq a$.

For example, in Fig. 2a we model the simple SIS program as a function $T_{Avg}$ of type $\mathbb{R}^2 \rightarrow \mathbb{B}$, where the input (pair) corresponds to the readings of the variables $i_1, i_2$, and the output corresponds to the value of the output variable $o$. There are two possible output failure states wrt. $(T_{Avg}) \in (\mathbb{R}^2)^2 \rightarrow (\mathbb{B})^2$, and they are $(t, f)$ and $(f, t)$.

Observe however from Def. 2 that the only way an output failure state can occur is if the corresponding input is also a failure state (since we are assuming that no additional failures are caused by the SIS program itself). Given a function $f$, we say that failure output state $(m', a')$ was caused by input failure state $(m, a)$ if $\langle f \rangle (m, a) = (m', a')$.

In the case of Fig. 2a, the failure state $(f, t)$ can only be caused by input failure state $((m_1, m_2), (a_1, a_2))$ if either $m_1 < a_1$ or $m_2 < a_2$. Here the values $m_1, a_1$ correspond to the variable $i_1$ and $m_2, a_2$ correspond to the variable $i_2$ in the figure. In scenarios where e.g. $m_1 < a_1$ there is always some reported value

---

5 In our abstract model we use a single type $V$ for simplicity of presentation.
6 Note here that we are distinguishing the types in the example.
for \( m_2 \) such that the reported average \( (m_1+m_2)/2 \) is below the fixed threshold in \( G_{com}\), thus there exists a scenario satisfying the identified input constraints such that:

\[
\langle T_{\text{Avg}} \rangle ((m_1,a_1), (m_2,a_2)) = (f,t) .
\]

From this example we can see there are potentially infinitely many values for a failure state \((m,a)\) whenever \(m,a\) can take real values. Rather than a safety engineer needing to know these precise values, what is more relevant is a report of the (usually) finite number of classes or \textit{modes} describing the kinds of failure.

\textbf{Definition 3.} Given a set of states \( V \times V \) wrt. a failure model, the failure modes are defined by a partition \( P \) of \( V \times V \). Each subset in \( P \) defines a failure mode (relative to \( P \)). Two states \((m,a)\) and \((m',a')\) satisfy the same failure mode if and only if they belong to the same partition subset of \( P \).

Given a partition \( P \) defining a set of failure modes we define \( \text{md}_P : V \times V \rightarrow P \) which maps failure states to their relevant failure mode (partition subset).

Examples of failure modes are normally described by constraints on variables. For instance in Fig. 2a the failure modes for the initial failure state \(((m_1,m_2),(a_1,a_2))\) are summarised by “either \( i_1 \) is reading too low or \( i_2 \) is reading too low”. In terms of Def. 3 this can be characterised by part of a partition that includes \( \ell_1, \ell_2 \) and \( \ell \), where \( \ell_1 \) is the set of failure states such that \( m_1 < a_1 \land m_2 \geq a_2; \ell_2 \) is the set of failure states such that \( m_1 \geq a_1 \land m_2 < a_2 \) and \( \ell \) is the set of failure states such that \( m_1 < a_1 \land m_2 < a_2 \).

Given an output failure mode, we would like to compute all initial failure modes that could cause that final failure mode. We say that an initial failure mode \( e \) (to an SIS function) \textit{causes} an output failure mode \( e' \) (of an SIS function) if there exists a failure state satisfying \( e \) such that the output of the SIS function given that initial state satisfies \( e' \).

For a given SIS function \( f \), one way to do this is to compute all relevant failure states for \(\langle f\rangle\), and then use \( \text{md}_P \) to interpret the failure modes for each failure state. Our first observation is that, given a partition \( P \) defining the failure modes, we can simplify this procedure significantly by abstracting the behaviour of \( f \) to act directly in terms of the failure modes rather than failure states.

\textbf{Definition 4.} Let \( f: V \rightarrow V \) be an SIS function, and \( P \) be a partition of \( V^2 \) defining the set of failure modes as in Def. 3.

We define \( \lfloor f \rfloor_P : P \rightarrow P^P \) to be the failure mode abstraction of \( f \) as the (possibly nondeterministic) function satisfying the following constraint for any input \((m,a) \in V^2\):

\[
\text{md}_P \circ \langle f \rangle(m,a) \in \lfloor f \rfloor_P \circ \text{md}_P(m,a) .
\]

In Fig. 2a where the initial failure modes are \( \ell_1, \ell_2 \) and \( \ell \) explained above, and final failure modes are \( f = \{(f,t)\} \) and \( t = \{(t,f)\} \), we can see that \( [T_{\text{Avg}}]_P(\ell_1) \)
Reasoning with failures contains functions, where we are writing $\mathcal{P}$ to represent the partition defined by all initial and final variables.

We shall show below that there are a variety of functions that have well-defined failure mode abstractions. Our next task however, is to show that the abstraction defined by Def. 4 is compositional, i.e. the abstraction of $f;g$ of SIS functions $f$ and $g$ can be computed from the composition of their abstractions. We recall the well-known Kleisli lifting of set-valued functions as follows.

We write the composition $f;g$ to mean first $f$ is executed, and then $g$, or as functions the output from initial $s$ is $g(f(s))$.

Let $\rho : \mathcal{T} \to \mathcal{P}\mathcal{T}$, define $\rho^1 : \mathcal{P}\mathcal{T} \to \mathcal{P}\mathcal{T}$

$$\rho^1(K) := \bigcup_{k \in K} \rho(k) .$$  \hspace{1cm} (2)

**Lemma 1.** Let $f, g$ be SIS functions which have well-defined failure-mode abstractions as given by Def. 4. The failure-mode abstraction for the composition $[f;g]_\mathcal{P}$ is equal to $[g]_\mathcal{P}^1 \circ [f]_\mathcal{P}$, where $[g]_\mathcal{P}^1 : \mathcal{P}\mathcal{P} \to \mathcal{P}\mathcal{P}$ is the standard lifting set out at Eqn. 2 above. \hspace{1cm} (3)

**Proof.** (Sketch) We show, for any input $(m,a)$, that:

$$md_\mathcal{P} \circ \langle y \rangle \circ \langle f \rangle (m,a) \in [g]_\mathcal{P}^1 \circ [f]_\mathcal{P} \circ md_\mathcal{P}(m,a) ,$$

and that all failure modes arise in this way. The result follows from Def. 4 and standard manipulations of set-valued functions.

The failure mode abstractions $[f]_\mathcal{P}$ enable a significant simplification in the identification of possible failures in an SIS program. For example we shall see that $[T_{Avg}]_\mathcal{P} = [GcomK]_\mathcal{P}^1 \circ [Avg]_\mathcal{P}$ for abstractions of the function blocks $[GcomK]_\mathcal{P}$ and $[Avg]_\mathcal{P}$.

In general a safety analyst considers possible output failure modes and asks for the inputs that potentially cause them. In some circumstances some failure modes can never be satisfied by any input, and are deemed unreachable. The analyst is thus able to concentrate on reachable failure modes, defined next.

**Definition 5.** Given an SIS function $f$, and abstraction defined by Def. 4. A failure mode $m \in \mathcal{P}$ is reachable (wrt. $f$) if there is some input failure state $(i,i')$ such that $md_\mathcal{P} \circ \langle f \rangle (i,i') = m$.

Failure Mode Reasoning is based on backwards calculational reasoning. We use a weak transformer to compute all input failure modes which can possibly cause a given output failure mode. This is similar to the dual transformer of dynamic logic and the conjugate transformer for the well-known guarded command language.

\hspace{1cm} 7 More precisely we would define failure modes separately on inputs and outputs, and indeed this is what happens in practice. To simplify the presentation however we assume that there is a single partition which serves to define failure modes on a single set, without distinguishing between inputs and outputs.

\hspace{1cm} 8 Recall that for simplicity we assume that the function modes $\mathcal{P}$ applies to both functions $f$ and $g$. 

Definition 6. Given SIS function $f$, we define the inverse failure transformer $[f]_{\mathcal{P}} : \mathcal{P} \mathcal{P} \rightarrow \mathcal{P} \mathcal{P}$ as

$$[f]_{\mathcal{P}}(K) := \{ k \mid [f]_{\mathcal{P}}(k) \cap K \neq \emptyset \} .$$  \hspace{1cm} (3)

Def. 6 satisfies two properties. The first is that any initial failure modes computed from final failure modes are the ones that could cause the selected final failure modes. The second is that inverse failure transformers compute all initial failure modes from final reachable failure modes. The next two definitions formalise these properties.

Definition 7. Given SIS function $f$, we say an inverse failure transformer $t$ is sound wrt. $f$ if all $k \in t(K)$ implies $[f]_{\mathcal{P}}(k) \cap K \neq \emptyset$.

Definition 8. Given SIS function $f$, we say an inverse failure transformer $t$ is complete if for any set of reachable failure modes $\mathcal{F}$ and (initial) failure modes $\mathcal{I}$, we have the following:

$$\mathcal{I} \subseteq t(\mathcal{F}) \iff (\forall i \in \mathcal{I} \cdot [f]_{\mathcal{P}}(i) \cap \mathcal{F} \neq \emptyset) .$$ \hspace{1cm} (4)

Observe that given failure modes $m$ and $m'$ such that $m \in t\{m'\}$, then $m'$ is reachable if there is some $(i, i')$ such that $md_{\mathcal{P}}(i, i') = m$. In general the safety engineer is not concerned with “unrealistic” failure modes in the sense that no corresponding scenario comprised of failure states can be constructed.

It is clear from Def. 6 that $[f]_{\mathcal{P}}$ is a sound and complete transformer relative to $f$. The definition of completeness is important because it means, for the safety engineer, that all potential failure modes are accounted for by the abstraction. The next lemma records the fact that soundness and completeness is conserved by function composition.

Lemma 2. Let $f, g$ be SIS functions, and let $\mathcal{P}$ determine the failure modes so that $[f]_{\mathcal{P}}$ and $[g]_{\mathcal{P}}$ are sound and complete transformers. Then their composition $[f]_{\mathcal{P}} \circ [g]_{\mathcal{P}}$ is also sound and complete for the composition SIS function $f ; g$.

Proof. Follows from Def. 6 and standard facts about functions and their transformers [24,25].

In this section we have set out a formal methods treatment of failure modes for SIFs in SIS programs. We have demonstrated a simple model for failures and shown how this “application-oriented” approach supports a rigorous analysis of failure modes and how they are propagated in SIS programs. In the following sections we show how this can be used to justify the use of standard backwards-reasoning to compute all input failure modes that cause reachable failure modes.

\footnote{We do not treat non-termination nor partial functions.}
4 Failure mode reasoning

In this section we show how to apply the failures model introduced in Section 3 to the typical safety analysis.

Recall $T_{Avg}$ defined in Fig. 2a. In this example, the failure modes of interest relate to whether the readings of the various sensors accurately record the physical environment or not, and when they do not, which combinations of deviant readings have the potential to result in a hazard.

The safety analysis begins with the identification of hazardous outputs: these are outputs from the SIS program which would directly cause a hazard if it is not correct, in the sense that it deviates from the “true” result which would have been output had all the sensors accurately recorded the status of the plant.

For simplicity we assume that all readings are real-valued, thus we identify “True” with “1” and “False” with “0”. Following Def. 3 we set $\mathcal{V} = \mathbb{R}$ and identify a partition on $\mathbb{R} \times \mathbb{R}$ given as follows.

**Definition 9.** Define the failures partition as follows. Let $h, l, m$ respectively partition $\mathbb{R} \times \mathbb{R}$ defined by:

- $(r, r') \in h$ iff $r > r'$
- $(r, r') \in m$ iff $r = r'$
- $(r, r') \in l$ iff $r < r'$.

Here we have identified the common failure modes “reading too high”, corresponding to $h$ and “reading too low” corresponding to $l$. We have also included “reading correct” corresponding to $m$ which is not strictly speaking a “failure”, but is useful in the formal analysis. From our gas pressure example, the situation where the input recorded on $i_1$ is lower than the real pressure in the pipe is modelled by pairs of values that lie in $l$.

Safety engineers want to know the input failure modes that “cause” particular reachable output failures. Def. 8 and Lem. 2 above support a standard backwards reasoning method on failure modes directly.

For each variable $s$ in an SIS program we use $\hat{s}$ for a corresponding variable taking failure modes for values, which in this case is $\{h, l, m\}$.

**Definition 10.** Given an SIS function $f$ and a partition $\mathcal{P}$ defining the failure modes. A failure triple is written

$$\{ \hat{s} \in \mathcal{A} \} \quad f \quad \{ \hat{s}' \in \mathcal{A}' \},$$

where $\mathcal{A}, \mathcal{A}' \subseteq \{h, l, m\}$. The triple Eqn. 5 is valid if, for each failure mode $e \in \mathcal{A}$ there exists $(m, m')$ such that $md_{\mathcal{P}}(m, m') = e$ and $md_{\mathcal{P}}((f)(m, m')) \cap \mathcal{A}' \neq \phi$.

Note that as a special case where $\mathcal{A}$ is a singleton set $\{a\}$ we write “$\hat{s} = a$” rather than “$\hat{s} \in \mathcal{A}$”.

Def. 5 is reminiscent of a standard Hoare Triple for failure modes, however a failure triple is based on Def. 9. More importantly Def. 5 corresponds with the scenarios relevant for the assessment of failures. Whenever $f$ corresponds to an SIS function for example, the valid triple given by Eqn. 5 means that the initial
failure mode corresponding to a causes the final failure mode \( a' \). This effectively enables the identification of failure mode propagation, summarised in the next result.

**Theorem 1.** Let \( f \) be an SIS function and \( \mathcal{P} \) define the relevant failure modes. Let \( a' \) be a reachable final failure mode wrt. \( f \). Then for all \( a \in [f]^{\mathcal{P}} \{ a' \} \)

\[
\{ \hat{s} = a \} \ f \ \{ \hat{s}' = a' \}
\]

is a valid failure triple.

**Proof.** Definition of \([f]^{\mathcal{P}}\), Def. 6.

**Backwards reasoning for failure modes:** As mentioned above we can use Thm. 1 to compute the failure modes that are the cause of a given reachable final failure mode. A complex SIS program determining a SIF typically comprises multiple function blocks with clearly defined “input” variables and “output” variables, where the outputs are determined by the values on the inputs. The architecture of the SIS program is then equated with a composition of a series of function blocks. Now that we have a formal description in terms of failure triples, we are able to use the standard composition rule:

\[
\{ \hat{s} = a \} f_1 \ {\hat{s}_1 = \hat{b}} \ \wedge \ {\hat{s}_1 = \hat{b}} \ f_2 \ {\hat{s}' = a'} \\
\Rightarrow \ {\hat{s} = a} \ f_1; f_2 \ {\hat{s}' = a'}.
\]

From this we can now deduce failure triples of a complex SIS program by reasoning about failure triples for component function blocks. We illustrate this for \( \text{Avg} \) and \( \text{Gcom}_K \) in the next section.

## 5 Individual function blocks

A typical SIS program library, from which function blocks (FBs) are chosen, may include 100 types of FBs [30]. For each FB the relationships between FB input failure modes and FB output failure modes, can be summarised in a Failure Mode Block (FMB). An FMB is proposed based on the well-defined function of its corresponding FB. In this section we will propose FMBs for SIS functions \( \text{Avg} \) and \( \text{Gcom}_K \), which we used in our gas pressure example, and we will prove the soundness and completeness of the proposed FMBs. More sample FMBs are proposed and proven in the Appendix.

The \( \text{Avg} \) function block takes two inputs and computes the average. The relevant output failures therefore are whether the output reads too high or too low. The abstraction for failure modes is given below.
Definition 11. Let \( \text{Avg} \) be the function defined by: \( \text{Avg}(i_1, i_2) := \frac{(i_1 + i_2)}{2} \). Its associated FMB, \( \text{FAvg} \), is defined as follows:

\[
\begin{align*}
\{ \hat{i}_1 = h \lor \hat{i}_2 = h \} & \quad \text{Avg} \quad \{ \hat{o} = h \} \\
\{ \hat{i}_1 = l \lor \hat{i}_2 = l \} & \quad \text{Avg} \quad \{ \hat{o} = l \} \\
\{ \hat{i}_1 = h \land \hat{i}_2 = l \} & \quad \text{Avg} \quad \{ \hat{o} = m \}
\end{align*}
\]

Def. 11 tells us that if the output reads too high, then it must be because one of the two inputs also reads too high. Similarly, if the output reads too low then it can only be because one of the two inputs reads too low. On the other hand the output can deliver an accurate result for scenarios where one input reads too high and the other reads too low. At the qualitative level of abstraction, however, all of these possibilities must be accounted for.

\( \text{Gcom}_K \) is another typical function block which compares the input with a given threshold and reports whether the input meets the given threshold.

Definition 12. Let \( \text{Gcom}_K \) be the function defined by: \( \text{Gcom}_K(i) := (i > K) \). Its associated FMB, \( \text{FGcom} \), is defined as follows:

\[
\begin{align*}
\{ \hat{i} = h \} & \quad \text{Gcom}_K \quad \{ \hat{o} = t \} \\
\{ \hat{i} = l \} & \quad \text{Gcom}_K \quad \{ \hat{o} = f \} \\
\{ \hat{i} = l \lor \hat{i} = m \lor \hat{i} = h \} & \quad \text{Gcom}_K \quad \{ \hat{o} = m \}
\end{align*}
\]

Def. 12 tells us that the output reading \( f \) when it should read \( t \) can only happen when the input is delivering a lower value than it should, and similarly the output reading \( t \) when it should read \( f \) can only happen when the input reading is falsely reporting a high value. Notice that this definition is actually independent of \( K \), which is why \( K \) is suppressed in the FMB model.

The following theorem confirms that Def. 11 and Def. 12 are sound and complete in respect of their operational definitions.

Theorem 2. The \( \text{FAvg} \) and \( \text{FGcom} \) models Definitions Def. 11 and Def. 12 are the sound and complete failure models of \( \text{Avg} \) and \( \text{Gcom}_K \) (for all real-valued \( K \)).

Proof. Individual FMBs can be proven by using truth-tables. All possible combinations of faults at the inputs and outputs of a corresponding FB can be defined, based on which the soundness and completeness conditions can be examined. Detailed proof is given in Appendix.

6 FMR in practice

The FMR process consists of four main stages: composition, substitution, simplification and calculation. In the composition stage, FMBs and failure mode
variables are defined and connected in accordance with the SIS program. The model for our example SIS program (Fig. 2a) will include two FMBs: $FAvg$ and $FGcom$. Similarly, variables $o, w, i_1$ and $i_2$ in SIS program will have their own corresponding failure mode variables $\hat{o}, \hat{w}, \hat{i}_1$ and $\hat{i}_2$ in the model.

The reasoning process begins at the last FB, i.e. the one that produces the SIS output. In our gas pressure example, the given output fault is $\hat{o} = f$. Taking into account the function of $Gcom_K$ from Def. [12] we can say:

$$\{\hat{w} = \ell\} \quad Gcom_K \quad \{\hat{\dot{o}} = f\}$$

Statement (6) suggests that output $o$ being $f$ by fault implies that the input to the greater comparison FB, $w$, is reading lower than what it should.

The reasoning process continues through the SIS program until all the conclusion parts of the implication statements include no more intermediate variables. In our example, the next FB is $Avg$. Considering the function of $Avg$, if the fault $\hat{w} = \ell$ occurs at its output, we can conclude that from Def. [11]

$$\{\hat{i}_1 = \ell \lor \hat{i}_2 = \ell\} \quad Avg \quad \{\hat{\dot{w}} = \ell\}$$

This statement suggests that if the reported value at $w$ is lower than its intended value, then either input $i_1$ or $i_2$ may be reading lower. The reasoning sequence terminates here as the left hand side of (7) only includes SIS inputs.

In the second stage of FMR we use the logical composition rules to eliminate intermediate variables in order to reduce the set of FB failure reasons to only one relation that links SIS inputs to its outputs. In our example, the only internal variable is $\hat{w}$. By substituting (7) in (6) we can conclude:

$$\{\hat{i}_1 = \ell \lor \hat{i}_2 = \ell\} \quad Avg; Gcom_K \quad \{\hat{o} = f\}$$

which is very similar to the result (1) of our earlier informal description of FMR.

The third stage of FMR is simplification, where we use standard rules of propositional logic [6] to simplify (8) and create the FMR short list of failure triples. As (8) is already minimal, we can easily see that our short list of faults comprises $i_1 = \ell$ and $i_2 = \ell$.

Having the input failure modes identified, we can implement the last stage of analysis, calculation, in which we would assign probability values to individual failure events and calculate the overall probability of failure. We skip this stage for this simple example. A comprehensive safety analysis for a realistic case study is described in other work [17].

In a more recent project [18] we examined a larger case study where we integrated FMR with other model-base analysis methods HiP-HOPS [26] and CFT [20]. We demonstrated that not only is FMR able to handle larger examples with precision, but its output can also be of value to other safety analysis tools that are designed to model generic systems but not programs. The process we examined in this case study is briefly shown in Fig. 3: a SIS that protects a gas-fired industrial boiler against high level of water. The SIS program in this example consists of over 2170 function blocks. With close to 100 inputs and
over 25 outputs, the SIS performs a total of 34 safety functions (SIFs). The SIS program in this project was developed in FBD and saved in XML format.

The FMR analysis produced two short lists of failure modes, one for Dangerous Undetected (DU) failure and one for Spurious Trip (ST). The lists included a total of 39 failure scenarios. In the quantitative stage the failure data of SIS inputs were entered and the aggregated probability measures for DU and ST failures were calculated.

Provided that failure data are readily available, the whole analysis process for an SIS of this scale takes less than an hour to complete, using the experimental system incorporating FMR analysis [18]. Conducting similar analysis by hand would take days. To visualise the extent of work, consider manual implementation of a fault tree with around 3500 gates. Even if the analyst is prepared for such a challenge, the implemented model, and thus its outcome, will be prone to human error. In comparison, FMR is fast, accurate, consistent, and reliable.

7 Discussion and related works

Reasoning about faults is not a new research topic. Diagnostics based on systematic inference was extensively studied in the 1980’s. Some of the frequently cited articles include [8,14,27]. Generally speaking, the studies were aimed at answering one question: given an observed deviation at the output of a system, how can we identify the (potentially) faulty components by reasoning based on the knowledge of system structure and/or system function? Logic circuits, in particular, would make an interesting application as they typically consist of complex yet well-defined, logical structures. Unlike inference-based diagnostics, FMR is primarily designed to target probable input faults, rather than faulty system components. Input faults are external to the system and do not represent system failure scenarios.

FMR uses abstraction techniques, which is also a well-established area, particularly in formal methods [5]. One may find similarities between the abstraction in FMR and that of Qualitative Reasoning (QR), where quantitative aspects of
physical systems are replaced with qualitative representations [2,9]. It should be noted however that QR is a method for expressing physical entities, and with an application in AI; whereas FMR is a technique for reasoning about failures, and (at least, currently) focused on conventional safety systems.

FMR is in some respects similar to FTA. Both methods look at the root causes that can result in a given top event. Parts of the computation techniques are similar between the two methods as well. However, FMR and FTA are different in some conceptual respects. FTA is a generic method that can be applied to any fault in any type of system, whereas FMR is specifically designed for analysing SIS programs. FTA computes a Boolean state of failure-success, but FMR computes multiple failure modes. The top event in FTA is a single event, but the program output in FMR can be an array of variables. The main question FMR tries to answer is that: given an abstracted state of output and given the function that produces it, what are the possible (abstracted) states of inputs to that function. This is obviously different to FTA in which we “know” the failure behaviour of a system and we build a model (fault tree) to summarise our understanding. FTA relies on the knowledge and skills of the analyst whereas FMR extracts information directly from the system. In a general term, FTA is a failure modeling method while FMR is a mode calculation method.

FTA was first introduced in 1961 to study a missile launch control system. In almost six decades, many extensions and variations of the method have been introduced to solve other types of problems. Useful surveys are conducted on FTA and its extensions in recent years [28,19]. Thanks to the growing capabilities of today’s technology, attention has shifted towards modularity and automatic synthesis of fault trees, which can greatly assist with solving complex problems at less effort. Various model-based dependability analysis methods have been developed, such as HiP-HOPS [26,29], AADL [13] and AltaRica [22], which use FTA as their primary means and automate the synthesis process to some degrees. More recently, the concept of contract-based design has also been used for automatic generation of hierarchical fault trees from formal models [5].

The common concept in automatic hierarchical synthesis of fault trees is that if we have the typical definition of component fault trees, we can synthesise the system level fault tree by interconnecting the smaller fault trees of components. At a conceptual level, this idea is utilised by FMR too; however, the components in FMR are the FBs, as opposed to the other methods that analyse physical systems. Also, while FMR uses the actual SIS program for its analysis, the other methods rely on separate models or specifications in order to generate fault trees. The actual running program in SIS is always the most accurate, detailed, and specific source of information on the behaviour of system, and having that FMR does not require any additional models.

Model checking has been used in SIS related applications too (see [12,21] as examples). In model checking a formal specification of system (model) is checked against a formal specification of requirements. Such methods focus on verifying the program against the requirements, as opposed to FMR which aims to identify failure modes.
Satisfiability Modulo Theories (SMT) is about determining whether a first order formula is satisfiable with respect to some logical theory \cite{10}. SMT solvers are used in various applications in the field of computer and formal verification. With respect to FMR, SMT can potentially help with determining the SIS input values that can result in a given output value. While this makes a potential area for further research; our experiments so far indicate that any SMT analysis will require post-processing in order to transform the results into failure modes.

8 Conclusion

In this paper we have shown how techniques from traditional formal methods can be brought to bear on a challenging problem in safety engineering: that of determining with precision how faults arising from incorrect sensor readings propagate through complex SIS programs. Within the safety engineering discipline, FMR is a novel way to analyse failure modes in Safety Instrumented Systems. Future work will include more complex constructs for function blocks, including looping, timing and probabilistic analysis. Moreover, we are working on implementing FMR for identify systematic failures in SIS programs, where the input to the program is correct but the output is faulty due to a pre-existing error in program.

References

1. Rudolf Berghammer and Hans Zierer. Relational algebraic semantics of deterministic and nondeterministic programs. *Theoretical Computer Science*, 43:123–147, 1986.
2. Daniel G Bobrow. Qualitative reasoning about physical systems: an introduction. *Artificial intelligence*, 24(1-3):1–5, 1984.
3. Marco Bozzano, Alessandro Cimatti, Cristian Mattarei, and Stefano Tonetta. Formal safety assessment via contract-based design. In *International Symposium on Automated Technology for Verification and Analysis*, pages 81–97. Springer, 2014.
4. Edmund M Clarke, Thomas A Henzinger, Helmut Veith, and Roderick Bloem. *Handbook of model checking*, volume 10. Springer, 2018.
5. Patrick Cousot and Radhia Cousot. Abstract interpretation frameworks. *Journal of logic and computation*, 2(4):511–547, 1992.
6. Yves Crama and Peter L Hammer. *Boolean functions: Theory, algorithms, and applications*. Cambridge University Press, 2011.
7. Dexter Kozen David Harel and Jerzy Tiuryn. *Dynamic Logic*. Foundations of Computing. MIT Press, 2000.
8. Randall Davis. Diagnostic reasoning based on structure and behavior. *Artificial intelligence*, 24(1-3):347–410, 1984.
9. Johan De Kleer and John Seely Brown. A qualitative physics based on confluenes. *Artificial intelligence*, 24(1-3):7–83, 1984.
10. Leonardo De Moura and Nikolaj Bjørner. Satisfiability modulo theories: An appetizer. In *Brazilian Symposium on Formal Methods*, pages 23–36. Springer, 2009.
11. Edsger Wybe Dijkstra. *A discipline of programming*, volume 1. Prentice-Hall Englewood Cliffs, 1976.
12. Alessandro Fantechi and Stefania Gnesi. On the adoption of model checking in safety-related software industry. In *International Conference on Computer Safety, Reliability, and Security*, pages 383–396. Springer, 2011.
13. Peter Feiler and Julien Delange. Automated fault tree analysis from aadl models. *ACM SIGAda Ada Letters*, 36(2):39–46, 2017.
14. Michael R Genesereth. The use of design descriptions in automated diagnosis. *Artificial Intelligence*, 24(1-3):411–436, 1984.
15. IEC. Programmable controllers - Part 3: Programming languages. 2013.
16. IEC. Functional safety-Safety instrumented systems for the process industry sector - Part 1: Framework, definitions, system, hardware and application programming requirements. 2016.
17. Hamid Jahanian. Failure mode reasoning. In *2019 4th International Conference on System Reliability and Safety (ICRSRS)*, pages 295–303. IEEE, 2019.
18. Hamid Jahanian, David Parker, Marc Zeller, Annabelle McIver, and Yiannis Papadopoulos. Failure mode reasoning in model based safety analysis. In *7th International Symposium on Model-Based Safety and Assessment*, 2020.
19. Sohag Kabir. An overview of fault tree analysis and its application in model based dependability analysis. *Expert Systems with Applications*, 77:114–135, 2017.
20. Bernhard Kaiser, Peter Liggesmeyer, and Oliver Mäckel. A new component concept for fault trees. In *Proceedings of the 8th Australian workshop on Safety critical systems and software-Volume 33*, pages 37–46. ACS, Inc., 2003.
21. Jussi Lahtinen, Janne Valkonen, Kim Björkman, J Frits, Ilkka Niemelä, and Keijo Heljanko. Model checking of safety-critical software in the nuclear engineering domain. *Reliability Engineering & System Safety*, 105:104–113, 2012.
22. Shaojun Li and Xiaoxun Li. Study on generation of fault trees from altarica models. *Procedia Engineering*, 80:140–152, 2014.
23. Saunders Mac Lane. *Categories for the working mathematician*. Springer, 1978.
24. AK McIver, C Morgan, and JW Sanders. Application-oriented program semantics. *South African Computer Society (SAICSIT)*, 1997.
25. Carroll Morgan. Of wp and CSP. In *Beauty is our business, Feijen, W., Gasteren, A.J.M. van, Gries, D., Misra, J. (Eds.),* pages 319–326. Springer, 1990.
26. Yiannis Papadopoulos and John A McDermid. Hierarchically performed hazard origin and propagation studies. In *International Conference on Computer Safety, Reliability, and Security*, pages 139–152. Springer, 1999.
27. Raymond Reiter. A theory of diagnosis from first principles. *Artificial intelligence*, 32(1):57–95, 1987.
28. Enno Ruijters and Mariëlle Stoelinga. Fault tree analysis: A survey of the state-of-the-art in modeling, analysis and tools. *Computer science review*, 15:29–62, 2015.
29. Septavera Sharvia and Yiannis Papadopoulos. Integrating model checking with HiP-HOPS in model-based safety analysis. *Reliability Engineering & System Safety*, 135:64–80, 2015.
30. Siemens. *Industrial software S7 F/FH Systems - Configuring and Programming*. Siemens, 2015.
31. William E Vesely, Francine F Goldberg, Norman H Roberts, and David F Haasl. *Fault Tree Handbook (NUREG-0492)*. US Nuclear Regulatory Commission, 1981.
A Truth tables for individual FBs

All possible failure scenario related to Avg are summarised in Table 1. In this Table, the 2nd and 3rd columns indicate the relationships between reported and intended values at inputs $x_1$ and $x_2$ and the 4th column shows the type of fault at the output $y$, caused by the inputs $x_1$ and $x_2$. A question mark indicates that the relation between $m'$ and $a'$ cannot be determined; i.e., all faults are possible. The last three columns in Table 1 translate the 2nd, 3rd and 4th columns into failure modes, as used in the FMR modeling.

| no. | $(x_1 = m_1, a_1)$ | $(x_2 = m_2, a_2)$ | $(y = m', a')$ | $\hat{x}_1$ | $\hat{x}_2$ | $\hat{y}$ |
|-----|--------------------|--------------------|----------------|-------------|-------------|-------------|
| 1   | $m_1 < a_1$        | $m_2 < a_2$        | $m' < a'$      | $\ell$      | $\ell$      | $\ell$      |
| 2   | $m_1 < a_1$        | $m_2 = a_2$        | $m' < a'$      | $\ell$      | $m$         | $\ell$      |
| 3   | $m_1 < a_1$        | $m_2 > a_2$        | $m' ? a'$      | $\ell$      | $h$         | $a$         |
| 4   | $m_1 = a_1$        | $m_2 < a_2$        | $m' < a'$      | $m$         | $\ell$      | $\ell$      |
| 5   | $m_1 = a_1$        | $m_2 = a_2$        | $m' = a'$      | $m$         | $m$         | $m$         |
| 6   | $m_1 = a_1$        | $m_2 > a_2$        | $m' > a'$      | $m$         | $h$         | $h$         |
| 7   | $m_1 > a_1$        | $m_2 < a_2$        | $m' ? a'$      | $h$         | $\ell$      | $a$         |
| 8   | $m_1 > a_1$        | $m_2 = a_2$        | $m' > a'$      | $h$         | $m$         | $h$         |
| 9   | $m_1 > a_1$        | $m_2 > a_2$        | $m' > a'$      | $h$         | $h$         | $h$         |

Table 1: Truth-table for Avg

To use Table 1 for FB Avg defined by Def. 11, recall that $\hat{x}_1 \equiv \hat{i}_1$, $\hat{x}_2 \equiv \hat{i}_2$ and $\hat{y} \equiv \hat{o}$. To prove Theorem 2 all we need to do is to group the combinations of $\hat{x}_1$ and $\hat{x}_2$ that correspond to $\ell$, $h$, and $m$ in $\hat{y}$ column. It is evident from Table 1 that rows 1-4 compose $\hat{y} = \ell$, rows 6-9 compose $\hat{y} = h$, and rows 3, 5 and 7 compose $\hat{y} = m$. Compare these combinations with the ones given in Def. 11.

Likewise, Table 2 can be used to prove Theorem 2 for FB $G_{com}$ as defined by Def. 12.
In the remaining part of this Appendix we propose FMBs for function blocks Add, Sub, Abs, Lcom, Not, And and Or, and we present truth-tables that can be used to prove them.

**Definition 13.** Let Add be the function defined by: \( \text{Add}(x_1, x_2) := x_1 + x_2 \). Its associated FMB, FAdd, is defined as follows:

\[
\begin{align*}
\{ \hat{x}_1 = k \lor \hat{x}_2 = h \} & \quad \text{Add} \quad \{ \hat{y} = k \} \\
\{ \hat{x}_1 = l \lor \hat{x}_2 = l \} & \quad \text{Add} \quad \{ \hat{y} = l \} \\
\{ \hat{x}_1 = k \land \hat{x}_2 = l \lor \hat{x}_2 = k \} & \quad \text{Add} \quad \{ \hat{y} = m \}
\end{align*}
\]

**Table 3: Truth-table for Add**

| no. | \((x_1 = m_1, a_1)\) | \((x_2 = m_2, a_2)\) | \((y = m', a')\) | \(\hat{x}_1\) | \(\hat{x}_2\) | \(\hat{y}\) |
|-----|----------------------|----------------------|-------------------|----------------|----------------|----------|
| 1   | \(m_1 < a_1\)       | \(m_2 < a_2\)       | \(m' < a'\)      | \(l\)          | \(l\)          | \(l\)    |
| 2   | \(m_1 < a_1\)       | \(m_2 = a_2\)       | \(m' < a'\)      | \(l\)          | \(m\)          | \(l\)    |
| 3   | \(m_1 < a_1\)       | \(m_2 > a_2\)       | \(m' > a'\)      | \(l\)          | \(h\)          | \(a\)    |
| 4   | \(m_1 = a_1\)       | \(m_2 < a_2\)       | \(m' < a'\)      | \(m\)          | \(l\)          | \(l\)    |
| 5   | \(m_1 = a_1\)       | \(m_2 = a_2\)       | \(m' = a'\)      | \(m\)          | \(m\)          | \(m\)    |
| 6   | \(m_1 = a_1\)       | \(m_2 > a_2\)       | \(m' > a'\)      | \(m\)          | \(k\)          | \(k\)    |
| 7   | \(m_1 > a_1\)       | \(m_2 < a_2\)       | \(m' > a'\)      | \(h\)          | \(l\)          | \(a\)    |
| 8   | \(m_1 > a_1\)       | \(m_2 = a_2\)       | \(m' > a'\)      | \(h\)          | \(m\)          | \(k\)    |
| 9   | \(m_1 > a_1\)       | \(m_2 > a_2\)       | \(m' > a'\)      | \(h\)          | \(h\)          | \(h\)    |
Definition 14. Let Sub be the function defined by: \( \text{Sub}(x_1, x_2) := x_1 - x_2 \). Its associated FMB, \( \text{FSub} \), is defined as follows:

\[
\begin{align*}
\{ \hat{x}_1 = h \lor \hat{x}_2 = l \} & \quad \text{Sub} \quad \{ \hat{y} = h \} \\
\{ \hat{x}_1 = l \lor \hat{x}_2 = h \} & \quad \text{Sub} \quad \{ \hat{y} = l \} \\
\{ \hat{x}_1 = l \land \hat{x}_2 = l \lor \hat{x}_1 = h \land \hat{x}_2 = h \} & \quad \text{Sub} \quad \{ \hat{y} = m \}
\end{align*}
\]

Table 4: Truth-table for \( \text{Sub} \)

| no. | \((x_1 = m_1, a_1)\) | \((x_2 = m_2, a_2)\) | \((y = m', a')\) | \(\hat{x}_1\) | \(\hat{x}_2\) | \(\hat{y}\) |
|-----|----------------------|----------------------|-------------------|--------------|--------------|------------|
| 1   | \(m_1 < a_1\)       | \(m_2 < a_2\)       | \(m' < a'\)       | \(\ell\)     | \(\ell\)     | \(a\)      |
| 2   | \(m_1 < a_1\)       | \(m_2 = a_2\)       | \(m' < a'\)       | \(\ell\)     | \(m\)        | \(\ell\)   |
| 3   | \(m_1 < a_1\)       | \(m_2 > a_2\)       | \(m' < a'\)       | \(\ell\)     | \(h\)        | \(\ell\)   |
| 4   | \(m_1 = a_1\)       | \(m_2 < a_2\)       | \(m' > a'\)       | \(m\)        | \(\ell\)     | \(h\)      |
| 5   | \(m_1 = a_1\)       | \(m_2 = a_2\)       | \(m' = a'\)       | \(m\)        | \(m\)        | \(m\)      |
| 6   | \(m_1 = a_1\)       | \(m_2 > a_2\)       | \(m' < a'\)       | \(m\)        | \(h\)        | \(\ell\)   |
| 7   | \(m_1 > a_1\)       | \(m_2 < a_2\)       | \(m' > a'\)       | \(h\)        | \(\ell\)     | \(h\)      |
| 8   | \(m_1 > a_1\)       | \(m_2 = a_2\)       | \(m' > a'\)       | \(h\)        | \(m\)        | \(h\)      |
| 9   | \(m_1 > a_1\)       | \(m_2 > a_2\)       | \(m' ? a'\)       | \(h\)        | \(h\)        | \(a\)      |

Definition 15. Let Abs be the function defined by: \( \text{Abs}(x) := |x| \). Its associated FMB, \( \text{FAbs} \), is defined as follows:

\[
\begin{align*}
\{ \hat{x} = \ell \} & \quad \text{Abs} \quad \{ \hat{y} = t \} \\
\{ \hat{x} = h \} & \quad \text{Abs} \quad \{ \hat{y} = f \} \\
\{ \hat{x} = \ell \lor \hat{x} = m \lor \hat{x} = h \} & \quad \text{Abs} \quad \{ \hat{y} = m \}
\end{align*}
\]

Table 5: Truth-table for \( \text{Abs} \)

| no. | \((x = m, a)\) | \((y = m', a')\) | \(\hat{x}\) | \(\hat{y}\) |
|-----|----------------|-------------------|------------|------------|
| 1   | \(0 \leq m < a\) | \(m' < a'\)       | \(\ell\)   | \(\ell\)   |
| 2   | \(0 \leq m = a\)  | \(m' = a'\)       | \(m\)      | \(m\)      |
| 3   | \(0 \leq a < m\)  | \(m' > a'\)       | \(h\)      | \(h\)      |
| 4   | \(m < a < 0\)    | \(m' > a'\)       | \(\ell\)   | \(h\)      |
| 5   | \(m = a < 0\)    | \(m' > a'\)       | \(m\)      | \(h\)      |
Table 5: Truth-table for Abs

Definition 16. Let $L_{\text{com}}_K$ be the function defined by: $L_{\text{com}}_K(x) := (x < K)$. Its associated FMB, $FL_{\text{com}}$, is defined as follows:

$$
\begin{align*}
\{ \hat{x} = \ell \} & \quad L_{\text{com}}_K \quad \{ \hat{y} = t \} \\
\{ \hat{x} = h \} & \quad L_{\text{com}}_K \quad \{ \hat{y} = f \} \\
\{ \hat{x} = \ell \lor \hat{x} = m \lor \hat{x} = h \} & \quad L_{\text{com}}_K \quad \{ \hat{y} = m \}
\end{align*}
$$

Table 6: Truth-table for $L_{\text{com}}_K$

Definition 17. Let Not be the function defined by: $\text{Not}(x) := \neg x$. Its associated FMB, $F\text{Not}$, is defined as follows:

$$
\begin{align*}
\{ \hat{x} = t \} & \quad \text{Not} \quad \{ \hat{y} = f \} \\
\{ \hat{x} = f \} & \quad \text{Not} \quad \{ \hat{y} = t \} \\
\{ \hat{x} = m \} & \quad \text{Not} \quad \{ \hat{y} = m \}
\end{align*}
$$

Table 7: Truth-table for Not
Definition 18. Let \( \text{And} \) be the function defined by: \( \text{And}(x_1, x_2) := x_1 \land x_2 \). Its associated FMB, \( \text{FAnd} \), is defined as follows:

\[
\begin{align*}
\{ \hat{x}_1 = t \land \hat{x}_2 = t \} & \quad \text{And } \{ \hat{y} = t \} \\
\{ \hat{x}_1 = t \land \hat{x}_2 = m \} & \quad \text{And } \{ \hat{y} = f \} \\
\{ \hat{x}_1 = f \lor \hat{x}_2 = f \} & \quad \text{And } \{ \hat{y} = f \} \\
\end{align*}
\]

Table 7: Truth-table for \( \text{Not} \)

| \( i \) | \( m_1 \) | \( a_1 \) | \( m_2 \) | \( a_2 \) | \( m' \) | \( a' \) | \( \hat{x}_1 \) | \( \hat{x}_2 \) |
|---|---|---|---|---|---|---|---|---|
| 1 | f | f | f | f | f | f | m | m | m |
| 2 | f | f | f | t | f | f | m | f | m |
| 3 | f | f | f | t | f | f | m | t | m |
| 4 | f | f | f | t | f | f | m | m | m |
| 5 | f | t | f | f | f | f | f | m | m |
| 6 | f | t | f | t | f | t | f | f | f |
| 7 | f | t | t | f | f | f | f | t | m |
| 8 | f | t | t | f | f | t | f | m | f |
| 9 | t | f | f | f | f | f | t | m | f |
| 10 | t | f | f | t | f | f | t | f | m |
| 11 | t | f | t | f | t | f | t | t | t |
| 12 | t | f | t | t | t | f | t | m | t_u |
| 13 | t | t | f | f | f | f | m | m | m |

10 Mode \( u \) in column \( \hat{y} \) is explained later.
Table 8: Truth-table for And

Definition 19. Let Or be the function defined by: \( \text{Or}(x_1, x_2) := x_1 \lor x_2 \). Its associated FMB, \( \text{FOr} \), is defined as follows:

\[
\begin{align*}
\{ \hat{x}_1 = t \lor \hat{x}_2 = t \} & \quad \text{Or} \quad \{ \hat{y} = t \} \\
\{ \hat{x}_1 = f \land \hat{x}_2 = f \lor \} & \quad \text{Or} \quad \{ \hat{y} = f \} \\
\{ \hat{x}_1 = f \land \hat{x}_2 = f \lor \} & \quad \text{Or} \quad \{ \hat{y} = m \}
\end{align*}
\]

| no. | \( m_1 \) | \( a_1 \) | \( m_2 \) | \( a_2 \) | \( m' \) | \( a' \) | \( \hat{x}_1 \) | \( \hat{x}_2 \) | \( \hat{y} \) |
|-----|------|------|------|------|------|------|------|------|------|
| 1   | \( f \) | \( f \) | \( f \) | \( f \) | \( f \) | \( f \) | \( m \) | \( m \) | \( m \) |
| 2   | \( f \) | \( f \) | \( f \) | \( f \) | \( f \) | \( t \) | \( m \) | \( f \) | \( t_u \) |
| 3   | \( f \) | \( f \) | \( f \) | \( f \) | \( f \) | \( f \) | \( m \) | \( t \) | \( t \) |
| 4   | \( f \) | \( f \) | \( t \) | \( f \) | \( t \) | \( t \) | \( m \) | \( m \) | \( m \) |
| 5   | \( f \) | \( t \) | \( f \) | \( f \) | \( f \) | \( f \) | \( m \) | \( f \) | \( t_u \) |
| 6   | \( f \) | \( t \) | \( f \) | \( t \) | \( f \) | \( f \) | \( f \) | \( f \) | \( f \) |
| 7   | \( f \) | \( t \) | \( f \) | \( t \) | \( t \) | \( f \) | \( f \) | \( f \) | \( f \) |
| 8   | \( f \) | \( t \) | \( t \) | \( t \) | \( f \) | \( t \) | \( f \) | \( m \) | \( m \) |
| 9   | \( t \) | \( f \) | \( f \) | \( f \) | \( f \) | \( t \) | \( m \) | \( t \) | \( t \) |
| 10  | \( t \) | \( f \) | \( f \) | \( f \) | \( t \) | \( f \) | \( t \) | \( f \) | \( m \) |
| 11  | \( t \) | \( f \) | \( f \) | \( f \) | \( t \) | \( f \) | \( t \) | \( t \) | \( t \) |
| 12  | \( t \) | \( f \) | \( t \) | \( t \) | \( t \) | \( f \) | \( m \) | \( m \) | \( m \) |
| 13  | \( t \) | \( t \) | \( f \) | \( f \) | \( t \) | \( f \) | \( m \) | \( m \) | \( m \) |
Table 9: Truth-table for Or

|   | t | t | f | t | t | t | m | f | m |
|---|---|---|---|---|---|---|---|---|---|
| 14|   |   |   |   |   |   | m | t | t |
| 15|   |   | t | f | t | t | m | t | m |
| 16| t | t | t | t | t | t | m | m | m |

Practical implementation: For FMBs $F_{And}$ and $F_{Or}$ (Tables 8 and 9), we used subscript $u$ in column $\hat{y}$ to represent uncertain faults [17]. These are the scenarios for which a judgment on the propagation of fault cannot be made. Consider line 2 in Table 8 as an example. We know that the reported values at inputs $x_1$ and $x_2$ and output $y$ are all $f$, and that matches up the function of Or. However, we cannot make a judgment as to which input dominates that state of the output; neither input $f$ has priority over the other. This is different to the scenario in line 3, for instance, because the reported output is certainly dominated by the faulty input, which reports $t$. For an FB output to be in a faulty state, not only the reported state of the output should be reachable by the given inputs, but its fault status should be a certain cause of a dominating input fault too. Otherwise, we cannot make a statement on whether and how the fault can propagate through the FB.

In practice, uncertain failure scenarios such as rows 12 and 15 in Table 8 and rows 2 and 5 in Table 9 will not be used for tracking failure modes in SIS programs. While this early filtering at the FB level may compromise the theoretical condition of completeness in some cases, it helps prevent producing combinations of non-faulty inputs as “failure” modes at the program level.

Another filtering mechanism in FMR’s backward tracking concerns the mode $m$. During the tracking process, when we come across a variable with that particular mode, we do not continue tracking from that point any further. This is on the basis that, for whatever reasons, if a combination of input faults have resulted in no fault at the output, the combination is not important; i.e. the program output is immune against the failure combination in question.

In summary, the practical implementation of $F_{And}$ and $F_{Or}$ will include the following limited scenarios, which as can be seen are similar to the normal functions of AND and OR gates with two inputs:

- \{ $\hat{x}_1 = t \land \hat{x}_2 = t$ \} $\text{And} \ \{ \hat{y} = t \}$
- \{ $\hat{x}_1 = f \lor \hat{x}_2 = f$ \} $\text{And} \ \{ \hat{y} = f \}$
- \{ $\hat{x}_1 = t \lor \hat{x}_2 = t$ \} $\text{Or} \ \{ \hat{y} = t \}$
- \{ $\hat{x}_1 = f \land \hat{x}_2 = f$ \} $\text{Or} \ \{ \hat{y} = f \}$