Extending The Application Of Formal Methods To Analyse Human Error And System Failure During Accident Investigations

C.W. Johnson and A.J. Telford,
Department of Computing Science,
University of Glasgow, Glasgow, G12 8QQ.
Tel: +44 141 330 6053
Fax +44 141 330 4913
Email: {johnson,alastair}@dcs.gla.ac.uk
http://www.dcs.gla.ac.uk/~{johnson,alastair}

Abstract

Recent disasters at Bhopal, Chernobyl, Habsheim and Kegworth illustrate the point that software is rarely the sole cause behind major accidents. Operator intervention, hardware faults, even the weather conditions and malicious acts all combine to create the conditions for failure. In the aftermath of these accidents, it seems difficult for software engineers, systems developers, forensic scientists and interface designers to predict all of the ways in which systems can fail. It is, therefore, important that we learn as much as possible from those failures that do occur. Unfortunately, it is often difficult to gain a coherent overview from the mass of detail that is, typically, contained in many accident reports. This makes it difficult for readers to identify the 'catastrophic' events that produced the necessary conditions for disaster. The following pages argue that formal specification techniques from the field of software engineering can be used to resolve these problems. In particular, Lamport's Temporal Logic of Actions is used to build a unified account of the human errors and system failures that contributed to the Three Mile Island accident. This notation provides high-level abstractions that can be used to strip away the mass of irrelevant details which often obscure important events during disasters. Formal proof techniques can then be applied to the model as a means of identifying the causal relationships that must be broken in order to prevent future failures.

Keywords: *Accident analysis; formal methods; human factors; Temporal Logic of Actions.*

1 Introduction

Accident reports, typically, contain the detailed findings of experts drawn from many different disciplines. Forensic scientists, metallurgists, meterologists as well as software engineers and human factors experts all present their findings through these documents [1]. The level of detail presented in many accident reports makes it difficult for readers to obtain a clear and coherent overview of the events leading to major accidents. Further problems are created by the fact that the findings of each disciplines are, typically, separated into different chapters. Critical events in one account are often missing in other accounts. This makes it difficult for readers to gain an accurate impression of the complex ways in which human and system failure combine during major accidents [9, 10]

1.1 The Role Of Formality In Accident Analysis

This paper argues that the application of mathematical specification techniques can be extended from the fields of software and hardware engineering to describe the sequences of events that lead to major accidents. We intend to use formal representations to identify safety requirements *after* major accidents just as fault-trees and other formal notations are used to analyse possible failures before a system is built. A central objective in this work is to develop a common framework that can be used to analyse both system and human failures. This approach is appropriate because a number of commercial and regulatory organisations have mandated the use of formal notations in the engineering of safety-critical systems [14, 15].

Human error has played an important role in the course of many major accidents. It is, therefore, important that accident investigation techniques provide means of representing and reasoning about the role of operator intervention. Mathematical specifications have not been widely used to support this approach. They have, however, been used to support the design of interactive systems. For instance, Harrison and Thimbleby recruit algebraic notations to specify high level requirements for interactive systems [8]. Dix has extended this work to investigate timing problems in distributed, multi-user systems [5]. Sufrin and He have exploited the Z schema calculus to represent interaction with a text editor [16]. The following pages build on this work to show how formal specification and proof techniques can be extended from the field of software engineering and interface design to support the analysis of human factors and system failures.

1.2 Temporal Logic of Actions

Graphical notations have an intuitive appeal for accident analysis [11]. They provide positional cues to the sequence of events in a specification. For example, Figure 1 represents the initial events that led to the Kegworth disaster. Such diagrams provide means of representing the human factors impact of software engineering problems. For instance, the transition labelled `1st Officer incorrectly polls the AVM (airborne vibration monitor)` is enabled by the place `AVM is displaying out of range reading`. In other words, the implementation and presentation of the airborne vibration monitor was a contributory factor to the First Officer's mistake. Unfortunately, this example also illustrates some of the problems with graphical notations. They can quickly become intractable as more and more events are represented. Such problems can be reduced by introducing structuring mechanisms, such as hierarchical Petri nets. This hides necessary detail and can increase the problems associated with tracing the sequences of critical failures that lead to major accidents.

The following pages exploit Lamport's Temporal Logic of Actions (TLA) [13]. This textual notation offers a number of benefits for accident analysis. It provides well developed proof techniques. It also provides means of representing the temporal properties that frequently have a profound impact upon the course of an accident. For instance, the $\Box$ (read as 'always') operator can be used to describe properties that do not change throughout the course of an accident. The $\Diamond$ (read as 'eventually') operator can be used to describe properties that occur at an indeterminate point during interaction. This is a significant advantage because, given limited sensing technology and partial sources of evidence, it is frequently impossible to develop a precise timeline for all of the events during an accident. The important point here is that analysts can use TLA operators, such as $\Box$, to be precise when they have the necessary evidence. Less deterministic operators, such as $\Diamond$, can be used when there is no exact information about the temporal ordering of particular failures during major accidents.

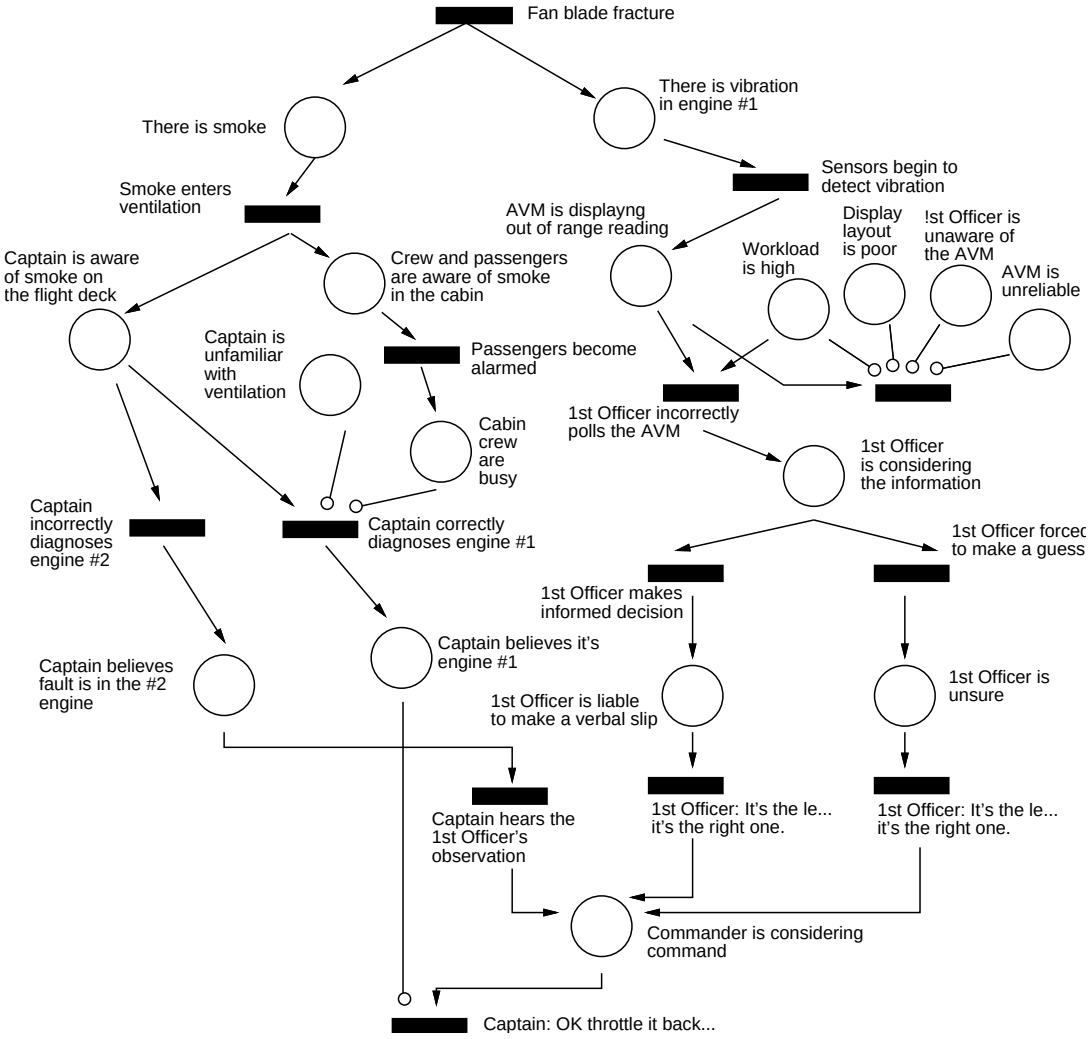

Figure 1: The Events Leading To The Kegworth Disaster.

1.3 The Three Mile Island Accident Overview

The Three Mile Island Accident provides an appropriate example for this paper because it illustrates the pathological mix of system failures and operator errors that pose the greatest challenge for software engineers and human factors analysts. Figure 2 presents a schematic overview of the main components in this accident. The problems in this reactor started when the main feedwater system malfunctioned. This was the result of a demineraliser failure. The auxiliary feedwater supply was supposed to have started automatically. It did not because several manual valves in the auxiliary system had been inadvertently left closed after a test of the system. Without the feedwater, the steam generators dried out. This deprived the reactor of its main heat sink and so there was a rise in temperature and pressure inside the primary coolant circuit. The rise in pressure automatically shut-down the reactor but the radioactive core continued to give out heat. The pressure relief valve opened and became stuck so that there was a continuous release of radioactive water from the main steel pressure vessel into a quench tank and the containment sumps. Within two minutes the emergency coolant system started automatically and began to increase the coolant level. Unfortunately, the pressure level indicator

malfunctioned and went off the top of the scale when in fact the pressure level was too low. This

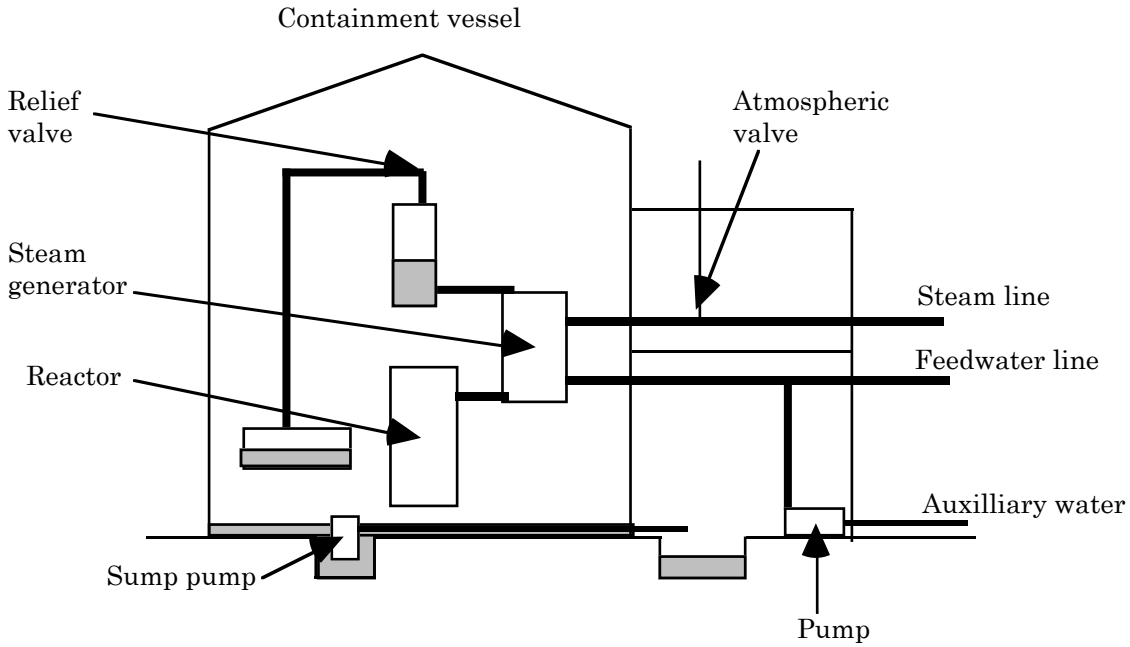

Figure 2: Diagram of the Three Mile Island Pressurised Water Reactor.

original loss of coolant accident (LOCA) was exacerbated by the operators who responded to the pressure warning by manually overriding the two emergency core cooling injection pumps. By fifteen minutes into the accident, there were almost 40,000 liters of primary coolant on the containment floor as the liquid began to escape from the sumps. Radioactive isotopes of krypton, iodine and xenon escaped to the environment through the building's ventilation system.

1.4 Outline Of The Paper

This section has introduced the argument that is presented in this paper. Section 2 builds on this and describes a high level framework that can be used to represent and reason about the operation of complex, interactive systems. This work has direct parallels with the abstract models that Lamport and others have developed to support the software engineering of concurrent programs. Section 2 then goes on to show how Lamport's model can be used to reason about human error and system failure. Unfortunately, this high level model is too abstract for accident investigations. Section 3, therefore, demonstrates that additional detail can be introduced to represent the specific human factors and systems problems that lead to particular accidents. It is then possible to identify a 'catastrophic' subset of these problems which are the primary causes of an accident. Section 4 shows how these can be drawn from the causal sequences that are described in Section 3. It is the identification of these catastrophic failures that, traditionally, forms the heart of any accident investigation. The important point here is that the use of a formal notation provides a precise means of representing the causal relationships between failures that are often buried within the prose of natural language reports. Section 5 then goes on to demonstrate an additional benefit of our approach. Formal proof techniques can be applied to TLA descriptions in order to prove that the causal relationships will hold given a known set of failures and a normative model of system operation. In other words, we can validate our model

of system failure by comparing the causal links in our specification to the known outcome of system failure and operator error. Section 6 summarises the conclusions from this work and suggests areas for future research.

2 A High Level Model For Accident Investigations

There are a number of key aims for any accident investigation [11]. In particular, analysts must identify:

- *the starting point or initial circumstances for any accident*;
- *the normative or anticipated behaviour of the system*;
- *the abnormal events that led to system failures and operator 'error'.*

This section shows how a model developed to support the software engineering of concurrent programs can also be used to provide a generic framework for these different activities. Subsequent sections will show how detail can gradually be added to this abstract model in order to identify the ways in which critical events combine during major accidents.

2.1 The Elements of Lamport's Model

Many similarities can be drawn between the development of concurrent programs and the generation of accident reports. Both activities force engineers to analyse asynchronous interaction between complex, autonomous processes. In software engineering, designers must account for the behaviour of programs interacting through tasking mechanisms or remote procedure calls. In the case of accident investigation, analysts must examine the ways in which multiple operators interact with a range of hardware and software systems. These similarities are also apparent if one compares the aims of accident investigations, illustrated by the previous list, to the formal components of Lamport's model for the software engineering of concurrent programs [13]. This model describes the operation of a system *Opconds* in terms of some initial conditions, *Init*, some changes to the state of the system, $\Box[\mathcal{N}]_f$, and a fairness condition, F:

$$Opconds \stackrel{\triangle}{=} Init \wedge \Box[\mathcal{N}]_f \wedge F \tag{1}$$

For our purposes, an accident report, *Opconds*, must represent the initial conditions that hold before an accident, *Init*. It must also consider the behaviour of the application during the accident, $\Box[\mathcal{N}]_f$. Finally, the report must contain a precise analysis of the catastrophic events, F, which ensured that the accident progressed.

2.2 Initialisation (*Init*)

A key stage in any investigation is the identification of a starting point for an accident. This may seem to be a trivial issue but an analysis of the reports that were produced in the aftermath of Three Mile Island indicates that this is not the case. For example, our account of the accident in Section 1.3 followed [7] in stating that the original cause was a demineraliser failure. Other reports blaim the accident upon the failure of the air supply to an air operated valve [3]. It is clearly important to reduce such ambiguity of software engineers, hardware developers and interface designers are to coordinate the development of future systems. Formal methods might

be used to provide an unambiguous representation of the initial circumstances that surround an accident. Unfortunately, *Init* is at too high a level of abstraction to achieve this. The elements of TLA can, however, provide additional detail to our formalisation of the accident. For example, *Init* might be defined to reflect the second interpretation based upon air supply failure rather than a mineraliser fault. The state of the mineraliser is omitted because it does not play a significant part in the subsequent analysis of the accident. It would be possible to explicitly include clauses to state that the mineraliser was working correctly. This has not been done in order to precisely reflect the lack of evidence about this:

$$
\begin{aligned}
& Init \triangleq (relief_valve = \mathsf{closed}) \wedge (primary_flow = \mathsf{normal}) \\
\wedge \quad & (steam_turbine = \mathsf{on}) \wedge (e_pumps_status = \mathsf{online}) \\
\wedge \quad & (e_pumps = \mathsf{off}) \wedge (air_circuit_status = \mathsf{clear}) \\
\wedge \quad & (secondary_flow = \mathsf{normal}) \wedge (ac_valves = \mathsf{open}) \\
\wedge \quad & (control_rods = \mathsf{up}) \wedge (ac_pumps = \mathsf{on}) \\
\wedge \quad & (water_runoff = \mathsf{false}) \wedge (water_temp = \mathsf{normal}) \\
\wedge \quad & (water_press = \mathsf{normal}) \wedge (feedwater_line = \mathsf{open}) \\
\wedge \quad & (relief_valve_status = \mathsf{working}) \wedge (hpp_enabled = \mathsf{online}) \\
\wedge \quad & (letdown_valve = \mathsf{closed}) \wedge (high_pressure_pumps = \mathsf{off}) \qquad (2)
\end{aligned}
$$

It should be stressed that this predicate is in itself an abstraction of the mass of more detailed information that may be presented to courts of enquiry. For example, *primary_flow* and *secondary_flow* indicate the flow of the water in the primary and secondary circuits. *steam_turbine* is an abstract representation of the state of the steam turbine. The *e_pumps* and *e_pumps_status* variables indicate whether the emergency pumps were on and whether they were enabled or not. The explicit representation of *Init* is an important stage in the development of accident reports just as it is for the engineering of concurrent programs. In both cases, it helps to frame the subsequent design and analysis by providing a starting point for further consideration.

2.3 Fairness Conditions (F

The second element of Lamport's model, is the fairness condition F. In software engineering terms, this is used to ensure that deadlock does not occur. In other words, F guarantees that computation must eventually progress. In terms of accident analysis, the fairness condition ensures that the plant will eventually run as planned, $\langle LOCA \rangle_f$. *LOCA* is the Loss of Coolant Accident. In other words, operator intervention or software and hardware protection mechanisms will intervene to avoid the accident. Or they will ensure that the system will eventually fail, $\neg Enabled \langle LOCA \rangle_f$. At first sight, it might seem strange to consider both the successful intervention of safety systems and their failure as fairness conditions. This is important because many accident reports describe 'near misses' [2]. In other words they report on the reasons why an accident was eventually avoided:

$$F \triangleq (\Box\Diamond\langle LOCA \rangle_f) \vee (\Box\Diamond\neg Enabled \langle LOCA \rangle_f) \qquad (3)$$

This predicate simplifies the potential course of interaction with the nuclear power plant so that either a loss of coolant accident must occur or it becomes impossible. This is a *strong* fairness condition (see [13, section 6.3]). It is important to emphasise that this disjunction precisely characterises the aim of accident reports. These documents are intended to describe how an accident eventually occurred so that it can never happen again.

2.4 State Transitions ($\mathcal{N}$)

In software engineering, designers must establish that certain output relations will eventually hold given a set of pre-conditions, *Init*, and some well-defined state transitions, $\mathcal{N}$ [13]. In accident analysis, investigators must show how an accident eventually occurred given the basic operating conditions of the nuclear plant, $\mathcal{N}$, and the set of initiating events described in their report, *Init*. This parallel between software engineering and accident analysis can be used to structure the formal analysis of our Three Mile Island example. For example, it is possible to introduce a causal relationship, $A \leadsto B$ (read as 'leads to'), that indicates the intended impact of a sequence of instructions in a concurrent program. These are the transitions described by $\mathcal{N}$. This same relation can also be used to describe the cause-consequence chains that lead to major accidents. The following formula can be read as it is always the case that if A occurs then eventually B will happen:

$$A \leadsto B \stackrel{\triangle}{=} \Box(A \Rightarrow \Diamond B) \tag{4}$$

The $\leadsto$ operator can be used to describe the way in which critical failures lead to an eventual accident. In the following, *Opconds* represents the operating conditions existing at the plant, prior to and during the accident. This was defined in (1). *Cat* represents the set of catastrophic conditions that occurred on 28th March 1979. These form the traditional focus of accident analysis:

$$Opconds \Rightarrow (Cat \leadsto LOCA) \tag{5}$$

This section has argued that the similarities between the software engineering of concurrent programs and accident analysis can be used to provide a formal structure for accident reports. Lamport's abstract model for the specification of parallel applications has been used to describe the events leading to major failures at an extremely high level of abstraction. This is important because it helps designers to strip away the clutter of low level details that can frequently obscure critical events during an accident. Unfortunately, such a high level of abstraction is inappropriate for many stages of accident analysis. It is, therefore, important that human factors analysts and systems engineers can gradually introduce additional detail as their investigations progress. This process is illustrated in the following section.

3 Causal Sequences ($\leadsto$)

One of the ways in which formal notations can add detail to high level accounts of accident reports is by representing the causal chains that lead to major accidents. These chains are built from the operating conditions, *Opconds*, and the catastrophic events, *Cat*, that led to the loss of coolant accident, *LOCA*. It is important to have a clear representation of these sequences because it can be difficult for readers to extract causal relationships from the natural language accounts of major accidents. For example, one report includes the following lines:

> ... they [the maintenance crew] allowed some water to enter an air circuit that opened and closed some valves ... The affected valves shut off ... ([3], p.12)

This quotation illustrates how the ordering and selection of material is used to imply a causal relationship between water entering the air circuit and the valves being shut off. The following

clause makes this relationship explicit. WAC denotes water being introduced into the air circuit and VSO stands for the valves switching off in the circuit:

$$WAC \leadsto VSO \tag{6}$$

This illustrates how causal chains are formed from the catastrophic events, *Cat*, such as the introduction of water into the air circuit, and from the operating conditions, *Opconds*, that determine the system's response to the failure. In the previous clause, these conditions ensure that the valves are switched off. Having constructed such a causal sequence, software and hardware engineers can trace the ways that subsequent events stemmed from these initial failures. For instance, the same report states that:

> The affected valves shut off and the pumps in the same circuit closed down in quick succession.

The same $\leadsto$ relation can be used to represent the consequences of the valve failure which in turn was caused by the water entering the air circuit. In the following clause, PCD is the action of the pumps closing down:

$$VSO \leadsto PCD \tag{7}$$

The important point here is that the $\leadsto$ relation forces analysts to consider the ways in which a wide range of failures contribute to major accidents. The initial events might simply relate to the hardware or systems engineering but the knock-on effects of those failures are quickly propagated throughout the application. In our example, the failure of the valves led to the pumps closing down. This led to the feedwater line being closed. The consequences of this was that the automated control systems intervened. They shut down the steam turbine and started up the emergency turbines. As software is increasingly being introduced into such safety-critical applications it is vital that designers understand how such knock-on failures affect the overall requirements for their systems. Unfortunately, it is not easy to extract these requirements from natural language accounts:

> One of the pumping circuits affected [by the valves switching off] was the feedwater line to the steam generator in the secondary circuit. ...so the safety system shut down the steam turbine and the electric power generator it drove. ...When the pumps supplying water to the steam generator stopped, three emergency pumps for the feedwater started.

As before, formal notations can be used to focus in upon the causal relationships that are embedded within such informal observations. FSD represents the feedwater line to the steam generator being closed. STS represents the shutdown of the steam turbine and EPS is the action of the emergency pumps starting up:

$$PCD \leadsto FSD \tag{8}$$

$$FSD \leadsto (STS \wedge EPS) \tag{9}$$

Thus far, we have shown how a hardware failure led to the intervention of control systems in order to start up emergency pumping equipment. We have not, however, considered the ways in which human intervention, or lack of it, can exacerbate system failures. Again such observations are embedded within natural language accounts:

> ... an operator ... noticed that these pumps were running but he did not notice two particular warning lights on a control panel. They signalled that valves were closed on each of the two emergency feedwater lines and so were preventing water from reaching the steam generators, which soon boiled dry ... Not until almost eight minutes later did the operators notice the closed valves and open them.

We use EVC to represent the fact that the emergency feedwater line valves were closed and SBD stands for the steam generators boiling dry.

$$(FSD \wedge EPS \wedge EVC) \rightsquigarrow SBD \tag{10}$$

It is important to note that this clause does not explicitly represent the absence of operator intervention. In previous work [9], we have represented this by the introduction of additional requirements. This approach is illustrated by the following clause:

$$(FSD \wedge EPS \wedge EVC \wedge \neg INT) \rightsquigarrow SBD \tag{11}$$

Such formulae require further refinement if they are to guide the future activities of interface designers, systems engineers and programmers. For instance, there must be some explanation as to why the operators did not intervene to open the valves, $\neg INT$. The subsequent sections address this issue in greater detail. In contrast, the remainder of this section continues to build the causal chain of events leading to the Three Mile Island accident:

> Deprived of an outlet for the heat still entering from the reactor core the water in the primary circuit increased in temperature and pressure ... A relief valve on top of the pressuriser opened and steam and water began to flow out to a drain tank ... To reduce the production of heat in the reactor core the control rods automatically descended into it ...

In the following, PCE stands for the emergency conditions which resulted in the primary circuit. In particular, there was a rise in the temperature and pressure in the circuit. CRD denotes the fact that the control rods dropped into the reactor core and RVO indicates the opening of the relief valve on the primary water circuit. RVO represents the relief valve being open and WRO denotes water run-off into the drain tank from the primary circuit:

$$SBD \rightsquigarrow PCE \tag{12}$$

$$PCE \rightsquigarrow CRD \wedge RVO \tag{13}$$

$$RVO \rightsquigarrow WRO \tag{14}$$

Formula (13) deserves particular attention because it illustrate how a causal chain of failure can have multiple consequences. An increase in temperature and pressure led to the introduction of the control rods and to the opening of relief valves. Such combined actions are often necessary in order to restore 'normative' states:

> The combined effects of lowering the control rods and opening the relief valve brought the temperature in the vessel down to normal ...

The use of a formal notation would provide few benefits if it could not trace the ways in which automated systems and safety applications successfully intervened:

$$CRD \wedge WRO \rightsquigarrow \neg PCE \tag{15}$$

This illustrates some of the complexities that can arise during accident investigations. It also emphasises the importance of tracing the full extent of $\leadsto$ chains. The fact that the temperature and pressure problems were being reduced, $\neg PCE$, did not mean that the accident was over. One of the relief valves was faulty and did not close as the pressure and temperature returned to normal:

> Through this valve more than one third of the contents of the primary circuit escaped. ... two high pressure pumps started automatically, triggered by a lowering of pressure in the primary circuit.

The following clause, therefore, states that if there is no pressure emergency in the primary circuit, $\neg PCE$, then water, WRO, run-off will lead to a loss of pressure, PLP. The second clause represents the observation that the high pressure pumps start, HPP, as a consequence of the loss of pressure if the pumps are enabled HPE:

$$\neg PCE \Rightarrow (WRO \leadsto PLP) \tag{16}$$

$$HPE \Rightarrow (PLP \leadsto HPP) \tag{17}$$

This final sequence provides the missing link between the initial ingress of water into the air circuit and the eventual loss of coolant accident (LOCA):

> The let-down valve was opened to release what was believed to be an excess of water. The original fall in pressure and the failure of the injection to bring down the temperature should have alerted the operators that LOCA was underway but it did not.

The following clause states that a loss of pressure in the primary circuit, together with opening the let-down valve, LDO, led to the Loss of Coolant Accident:

$$PLP \wedge LDO \leadsto LOCA \tag{18}$$

It is important to re-iterate the main point of this section which is that the complexity of natural language descriptions makes it difficult to reconstruct the causal sequences that lead to major failures. The reader is left to interpret the relationships that are built up from natural language statements such as 'and then', 'at which time', 'and so...', 'in quick succession' and 'they signalled that'. The use of the $\leadsto$ relation avoids such ambiguity. It also provides a concise means of representing the human errors and system failures that led to the loss of coolant accident.

4 'Catastrophic' Events (Cat)

The previous section has constructed causal sequences of events that characterise the course of a major accident. These sequences are built from a number of failures and from the responses of the system and its operators to those failures, *Opconds*. Having constructed these causal sequences, it is possible for analysts to extract those critical failures which were essential for the accident to progress, *Cat*. For the Three Mile Island example, these catastrophic events can be represented as follows:

$$Cat \triangleq WAC \vee RVF \vee EVC \vee LDO \vee ESO \tag{19}$$

The accident was caused by the introduction of water into the air circuit or the failure of the relief valve or by the operators' failing to close the relief valve or the opening of the let down valve in the primary circuit or the emergency pumps being disabled. These high level abstractions enable analysts to focus upon critical events. Hardware and software engineers are not forced to consider the mass of circumstantial detail that must be presented to accident inquiries. In the above, *ESO* refers to the emergency pumps being disabled and switched off:

$$ESO \triangleq \neg HPE \wedge \neg HPP \tag{20}$$

Clauses such as (19) and (20) are more detailed than the generic model for concurrent programs and accident analysis introduced in (1). Unfortunately, they are still not at a level that human factors analysts and programmers might use to guide the development of future applications. This problem can be addressed by representing the changes that critical events cause to process parameters. For example, the introduction of liquid into the air circuit caused its ducts to become waterlogged. The value of particular variables, such as *air_circuit_status*, can be directly related to the sensor readings that must be polled by control programs:

$$WAC1 \triangleq (air_circuit_status = \mathsf{clear}) \wedge (air_circuit_status' = \mathsf{water_filled})$$

It is important to emphasise that critical events will, typically, have no impact upon most of the variables in a system. This can be represented by introducing, $Unchanged\,(Var/S)$, to indicate that all variables in the system not in the set S are unchanged by a specified action:

$$\begin{aligned} WAC \triangleq\ & (air_circuit_status = \mathsf{clear}) \wedge (air_circuit_status' = \mathsf{water_filled}) \\ & \wedge Unchanged\,(Var/\{air_circuit_status\}) \end{aligned}$$

The failure of the relief valve was another catastrophic event that contributed to the accident. This event had no impact upon the rest of the system apart from causing the valve to fail. Again, this illustrates how the abstractions of formal notations can be used to scope or frame the impact of critical failures:

$$\begin{aligned} RVF \triangleq\ & relief_valve_status' = \mathsf{faulty} \\ & \wedge Unchanged\,(Var/\{relief_valve_status\}) \end{aligned}$$

The previous clause represents the impact of a hardware failure. It is also possible to represent the impact of operator 'error' in a similar manner. For example, the disabling of the high pressure coolant pumps played a major part in the events leading to the accident. The pumps were switched from a state in which they were enabled but off-line to one in which they had been turned off:

$$ESO \triangleq (hpp_enabled = \mathsf{offline}) \wedge (high_pressure_pumps' = \mathsf{off})$$

The fourth of the critical events was the closure of the emergency feedwater valves:

$$\begin{aligned} EVC \triangleq\ & ((feedwater_line = \mathsf{open}) \wedge (feedwater_line' = \mathsf{closed}) \\ & \wedge Unchanged\,(Var/\{feedwater_line\}) \end{aligned}$$

The final critical factor in the lead up to the accident was the opening of the let down valve in the primary circuit. This had the effect of cutting of the primary coolant flow to the reactor:

$$LDO \triangleq ((water_press = \mathsf{low}) \wedge (letdown_valve' = \mathsf{open}) \wedge primary_flow' = \mathsf{none}) \\ \wedge Unchanged(Var/\{primary_flow, letdown_valve\})$$

Similar definitions can be introduced for all of the events that have been introduced in our formal model of the accident. This is omitted for the sake of brevity and the interested reader is directed to [12]. In contrast, the following section argues that the formal proof techniques can be used to determine whether causal sequences can be reconstructed from the failures in *Cat* and the operating procedures in *Opconds*.

5 Proofs and properties

The formal modelling of major accidents provides a number of benefits for human factors and software engineers. It avoids the ambiguity that can be found in natural language descriptions of causal relationships. It provides a precise means of framing an accident. TLA clauses can be used to describe the initial conditions for failure, *Init*. A further advantage is that formal proof techniques can be used to determine whether the causal sequences, described by $\leadsto$, can actually be produced from the known operating conditions of the system, *Opconds*, and the catastrophic system failures and human errors in *Cat*. For example, accident investigators might want to prove that if the operating procedures at Three Mile Island were followed then the introduction of water into the air circuit would lead to the valves being switched off. If this were not the case then either our causal analysis is incorrect; something else triggered the valves. Or our understanding of the operating conditions is incorrect:

$$Opconds \Rightarrow WAC \leadsto VSO \tag{21}$$

As in previous sections of this paper, we can again recruit software engineering techniques to support accident investigation. In this case, we can use Lamport's (SF1) proof rule [13, page 22] to help us establish the previous relationship:

$$\frac{\begin{array}{l} P \wedge [\mathcal{M}] \Rightarrow (P' \vee Q') \\ P \wedge \langle \mathcal{M} \wedge \mathcal{A} \rangle_f \Rightarrow Q' \\ \Box P \wedge \Box[\mathcal{M}]_f \wedge \Box F \Rightarrow \Diamond Enabled \langle \mathcal{A} \rangle_f \end{array}}{\Box[\mathcal{M}]_f \wedge \mathrm{SF}_f(\mathcal{A}) \wedge \Box F \Rightarrow (P \leadsto Q)}$$

The following substitutions can be made between our model of the Three Mile Island accident and the tokens in Lamport's rule:

$$\begin{array}{lll} P \leftarrow WAC & \mathcal{M} \leftarrow \mathcal{N} & Q \leftarrow VSO \\ \mathcal{A} \leftarrow \mathcal{C} & f \leftarrow v & \Box F \leftarrow \mathrm{SF}_f(WAC) \end{array}$$

The first premise of SF1 can now be written as follows:

$$WAC \wedge [\mathcal{N}]_f \Rightarrow (WAC' \vee VSO') \tag{22}$$

This is satisfied trivially since, in this case, if WAC is true then WAC' must automatically be true. The second premise can be re-written as follows:

$$WAC \wedge \langle \mathcal{N} \wedge \mathcal{C} \rangle_f \Rightarrow VSO' \tag{23}$$

This is true since we are assuming that the initial conditions, *Init*, hold. Consequently, it must follow that *air_circuit_status* = water_filled and subsequently from *WCV* that *VSO′* is true. The final premise can be written as:

$$\Box\, WAC \wedge \Box[\mathcal{N}]_f \wedge SF_v(WAC) \Rightarrow \Diamond Enabled\mathcal{C}_v \tag{24}$$

The above results from the fact that *WCV* is part of the disjunction which comprises $\mathcal{C}$. Consequently, since we assume here that *WAC* is always true then *WCV* must be enabled at some point. By applying SF1 we then obtain the following:

$$Init \Rightarrow (\Box[\mathcal{N}]_v \wedge \mathrm{SF}_v(\mathcal{C}) \wedge \mathrm{SF}_v(Cat)) \Rightarrow (WAC \leadsto VSO) \tag{25}$$

Finally, we observe that (25) is isomorphic to

$$Opconds \Rightarrow (WAC \leadsto VSO) \tag{26}$$

Such proof techniques suggest further ways in which our application of Lamport's model might benefit from software engineering techniques. For example, the transitivity of the $\leadsto$ relation suggests that certain sub-chains in our causal sequences might be omitted. In other words, if $A \leadsto B$ and $B \leadsto C$ then it may be possible to omit B altogether. An example of this may be seen in the above formalisation where the action *VSO*, and the variable, *ac_valves*, could be omitted. The action of the air circuit pumps switching off would then follow directly from the water being introduced into the air circuit. Such formal analysis might then be used to remove unnecessary parts from an accident report. This is similar to the ways in which pieces of redundant or inefficient computer code may be identified and removed through mathematical analysis. However, this proposal also illustrates some of the dangers that arise when transferring software engineering techniques directly into accident analysis. The purpose of writing an accident report is not to produce a minimal account but rather to produce a document that effectively communicates the weaknesses of previous systems. Removing such transitive relations may reduce the volume of an accident report but it can also hide links that help readers to understand critical sequences of human 'error' and system failure.

6 Conclusions and Future Work

This paper has argued that Lamport's model for the software engineering of concurrent systems can be used to support the generation of accident reports. This provides a number of benefits. The elements of Lamport's model can be used to identify generic requirements for accident investigations. In particular, multi-disciplinary design teams must agree upon the starting point for an accident, *Init*, the anticipated behaviour of a system, *Opconds*, and the abnormal events that lead to failure, *Cat*. The elements of the model can also be applied to structure the detailed analysis of the events leading to major accidents. The Temporal Logic of Actions notation can be used to construct the causal sequences of human 'error' and system failure that lead to disasters. The $\leadsto$ relation can be used to explicitly represent the informal observations that are often buried within informal accounts. Finally, proof techniques can be applied to the elements of the model to determine whether the $\leadsto$ relation actually does explain the circumstances leading to an accident.

The work reported in this paper is the initial product of an Engineering and Physical Sciences Research Council project into the formal analysis of human factors and system failures. Much

remains to be done. In particular, a powerful application of our approach is that it can be used to detect inconsistencies not only within but also between accident reports. For example, our description of the Three Mile Island accident has been drawn from a number of sources, including [3], [7] and documents issued by the US President's Commission and the US Nuclear Regulatory Commission. These, in turn, were the distillation of evidence presented to the courts of enquiry. Our formalisation is, therefore, the result of a series of interpretations and simplifications. However, the techniques that are presented in this paper provide a precise and concise account of the events leading to major accidents. It is hypothesised that such formalisations might help to reduce the potential ambiguities that frequently arise in these different accounts.

A further benefit of formalisation is that it also provides engineers and analysts with proof techniques that can be used to verify *cross-viewpoints consistency*. This again illustrates the wider application of software engineering techniques for accident analysis. The need for such consistency has been recognised by a growing number of researchers in the field of software engineering [6, 4]. This work is an important starting point for our future research. It is vital that those who are affected by the aftermath of an accident are left with a consistent view upon the implications of a report. This will mean, for instance, that consistency has to be shown between nuclear and software engineers. It is also necessary to maintain consistency between human-computer interaction specialists and those who are interested in the computational aspects of systems development.

Acknowledgement

Thanks go to the members of the Glasgow Interactive Systems Group (GIST) and to the Formal Methods and Theory Group in Glasgow University. This work is supported by the UK Engineering and Physical Sciences Research Council, grants GR/JO7686, GR/K69148 and GR/K55040.

References

[1] Air Accidents Investigations Branch, Department of Transport. *Report On The Incident Involving BAC 1-11 G-AYWB and Boeing 737 EI-BTZ On 12 April 1988 At Gatwick Airport*, number 2/89, London, United Kingdom, 1989. Her Majesty's Stationery Office.

[2] Air Accidents Investigations Branch, Department of Transport. *Report On The Accident To Boeing 737-400 G-OBME Near Kegworth, Leicestershire on 8th January 1989*, number 4/90, London, United Kingdom, 1990. Her Majesty's Stationery Office.

[3] V. Bignell and J. Fortune. *Understanding Systems Failures*. Manchester University Press, 1984.

[4] H. Bowman, J. Derrick, P. Linington, and M. Steen. Cross-viewpoint consistency in Open Distributed Processing. *Software Engineering Journal*, pages 44–57, January 1996.

[5] A.J. Dix. *Formal Methods For Interactive Systems*. Academic Press, London, United Kingdom, 1991.

[6] A. Finkelstein and I. Sommerville. The viewpoints FAQ. *Software Engineering Journal: Special Issue on Viewpoints for Software Engineering*, pages 2–4, January 1996.

[7] J.H. Fremlin. *Power Production: What Are The Risks.* Adam Hilger Ltd., Boston, United States of America, 1985.

[8] M.D. Harrison and H.W. Thimbleby, editors. *Formal Methods In Human Computer Interaction.* Cambridge University Press, Cambridge, United Kingdom, 1989.

[9] C.W. Johnson. The formal analysis of human-computer interaction during accidents investigations. In G. Cockton, S.W. Draper, and G.R.S. Weir, editors, *People And Computers IX.* Cambridge University Press, Cambridge, United Kingdom, 1994.

[10] C.W. Johnson. The application of Petri Nets to represent and reason about human factors problems during accident analyses. In P. Palanque and R., editors, *The Second Eurographics Workshop On The Design, Specification And Verification of Interactive Systems*, pages 345–357. Springer Verlag, Berlin, Germany, 1995.

[11] C.W. Johnson, J.C. McCarthy, and P.C. Wright. Using a formal language to support natural language in accident reports. *Ergonomics*, 38(6):1265–1283, 1995.

[12] C.W. Johnson and Alastair Telford. The formal analysis of human factors errors and system failures. Technical report, Department of Computer Science, University of Glasgow, Glasgow, Scotland., 1996.

[13] L. Lamport. The temporal logic of actions. Research Report 79, DEC Systems Research Center, 130 Lytton Avenue, Palo Alto, California 94301, USA, December 1991.

[14] Ministry Of Defence. *Requirements for the Procurement of Safety Critical Software*, MOD DEF-STAN 00-55, London, United Kingdom, 1991.

[15] National Aeronautic and Space Administration. *Advanced Orbiting Systems - Architectural Specification For The CCSDS Secretariat*, Washington DC, United States of America, 1989.

[16] B. Sufrin and J. He. Specification, refinement and analysis of interactive processes. In M. D. Harrison and H. W. Thimbleby, editors, *Formal methods in Human Computer Interaction*, pages 153–200. Cambridge University Press, Cambridge, United Kingdom, 1990.