QUALITATIVE TEMPORAL ANALYSIS: TOWARDS A FULL IMPLEMENTATION OF THE FAULT TREE HANDBOOK

Martin Walker, Yiannis Papadopoulos

Department of Computer Science, University of Hull, UK
{m.d.walker, y.i.papadopoulos}@dcs.hull.ac.uk

Abstract: The *Fault Tree Handbook* has become the de facto standard for Fault Tree Analysis (FTA), defining the notation and mathematical foundation of this widely used safety analysis technique. The *Handbook* recognises that classical combinatorial fault trees employing only Boolean gates cannot capture the potentially critical significance of the temporal ordering of failure events in a system. Although the *Handbook* proposes two dynamic gates that could remedy this, a Priority-AND and an Exclusive-OR gate, these gates were never accurately defined. In this paper, we propose extensions to the logical foundation of fault trees that enable use of these dynamic gates in an extended and more powerful FTA. The benefits of this approach are demonstrated on a generic triple-module standby redundant system exhibiting dynamic behaviour. *Copyright © 2007 IFAC*

Keywords: fault trees, temporal logic, safety critical, safety analysis, reliability

1. INTRODUCTION & BACKGROUND

Fault Tree Analysis, or FTA, is a well-established system analysis technique widely used in reliability engineering & system safety. It was created in the 1960s and since then has been used in a variety of fields, including the automotive, aerospace, and nuclear industries, where numerous applications have been reported on safety critical control systems. FTA is a deductive analysis method, meaning that the analysis starts with a system failure and works backwards to try to determine its root causes. The result is a fault tree, which represents this knowledge graphically by showing the logical connections between different faults and their causes. The system failure is the *top event* of the fault tree, and it is connected to a set of *basic events*, or root causes, by a system of logical gates such as AND and OR and intermediate events. From the resultant fault tree, it is then possible to see how combinations of smaller basic faults, such as failures of individual components or certain environmental conditions, can cause major failures in the system as a whole.

The fault tree itself is then analysed to produce more useful information about the failure behaviour of the system. This further analysis comes in two forms: *qualitative* (logical) analysis and *quantitative* (probabilistic) analysis. Qualitative analysis is performed by reducing the fault tree to its *minimal cut sets*, which are the smallest combinations of failures necessary to cause the top events; these minimal cut sets are easier to understand than the fault tree as a whole. Quantitative analysis assigns probabilities to each of the basic events based on failure and repair rates, and from these, the probability of the top event occurring can also be estimated. These processes are best described in the *Fault Tree Handbook* (Vesely, Goldberg, Roberts, Haasl, 1981), which sets out the fundamentals of fault trees and explains the principles of fault tree construction and analysis.

Unfortunately, there is a notable absence in FTA: it cannot express time- or sequence-dependent dynamic behaviour. Fault trees rely upon classical Boolean logical gates to make connections between the basic events and the system failure, and these simple gates struggle to model the dynamic behaviour of systems in which time plays a major role. Such systems include those with different modes of operation, in which the failure behaviour changes depending on which mode is in operation; systems with redundant or standby components, which can partly adapt to the presence of failures; and more generally, systems in which the failure behaviour is dependent on the sequence of events. For example, consider the triple-module standby redundant system in Fig 1.

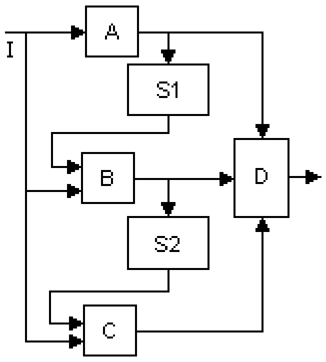

Fig 1: Simple example of a triple-module redundant system

The system is generic in the sense that component A, B and C could be any input, control or actuating device. A is the primary component that takes input from I, performs some function on it, and feeds the result to D. S1 is a monitoring sensor that detects an omission of output from A and activates the first standby, B. Similarly, S2 is another sensor that detects a failure of B and actives the second standby, C. In theory, all three components A, B, and C must fail in order for the whole system to fail – the failure of one or two of the three can be tolerated. The results of a classical fault tree analysis would seem to confirm this, as the following minimal cut sets show:

```
1. No input at I
2. Failure of all of A, B, and C
3. Failure of A and S1
4. Failure of A, B, and S2
5. Failure of D
```

Each of these five minimal cut sets is sufficient to cause the whole system to fail. However, the situation is not so simple; consider cut sets 3 and 4: what if one of the sensors fails *last*? If S1 fails *after* A has failed, then it does not lead to failure because S1 has already performed its function and activated standby component B. Similarly, if S2 fails *after* B has failed, then C will have been activated, and there is no system failure. Both of these are therefore unnecessarily pessimistic results. Conversely, since the sensors work by detecting an omission, if B fails before A, then S2 would never detect a cessation in output from B, because it was never activated; C therefore remains unused. Cut set A.B.C is then a very optimistic result, since B failing before A is sufficient to cause the system to fail. Clearly, classical fault tree analysis fails to capture the full failure behaviour of this system, because it cannot adequately model the effect of time and sequences of events on the failure behaviour. Instead, the results obtained are either unnecessarily pessimistic or excessively optimistic; and if a system with only five cut sets can have such pronounced inaccuracies, how far can we trust an analysis of a system with hundreds of thousands of cut sets, like a nuclear plant?

This is not a new problem, and there have been several different solutions proposed. In recent times, the most prominent of these is the Dynamic Fault Tree (or DFT) methodology and the related Galileo FTA tool (Sullivan, Dugan, & Coppit, 1999; Manian, Dugan, Coppit & Sullivan, 1998). DFTs date back to the early 1990s, and their goal is to extend fault trees to be able to work with Markov chains so that fault tolerant computer systems can be better analysed; DFTs are also included in the updated *Fault Tree Handbook* (Vesely et al, 2002) as a method of analysing dynamic systems. DFTs allow for the analysis of systems with more complex interrelationships between events, including functional dependencies, standby components, and event sequences. DFTs can analyse both normal and dynamic systems; the Markov chains are generated only for those parts of the fault tree that require it. Although they are primarily meant for quantitative analysis, there has also been a recent suggestion on using DFTs for qualitative analysis (Tang & Dugan, 2004); however, this approach has its drawbacks, since it does not account for the possibility of contradictions or redundancies arising as a result of the temporal information in the fault tree and instead deals with the temporal and static qualitative analyses separately.

Another approach is the Temporal Fault Tree (TFT) methodology (Palshikar, 2001). TFTs are based on a new temporal logic called PLTLP (Propositional Linear Temporal Logic) and introduce a set of new gates to represent the various temporal relationships that this logic allows, such as UNTIL-PAST, PREV, and WITHIN. However, although the TFT approach is sufficiently expressive, it is meant to be employed *after* a system has failed, as a diagnostic aid. It therefore relies upon a log of the system behaviour and is designed to work with specific timings (e.g. WITHIN 5 seconds); as a result, it is less useful when applied during the design phase of the system. Its gates and logic are also relatively complicated, based as they are on a formal style of temporal logic, which makes it less intuitive to use.

One other attempt to represent the temporal information as part of the fault tree structure is the AND-THEN or TAND gate (Wijayarathna and Maekawa, 2000). The TAND specifies that one event must immediately follow the cessation of another, and is designed to be used as a building block to construct more complex expressions. In combination with a temporal version of the AND gate, it is able to precisely represent many temporal relations between two intervals. However, it suffers from a number of problems, mainly stemming from its reliance upon *states*. The *Fault Tree Handbook* states that "From the standpoint of constructing a fault tree we need only concern ourselves with the phenomenon of occurrence. This is tantamount to considering all systems as nonrepairable." (Vesely et al, 1981, p V-1) The idea is that an event can only occur once, and once it occurs, it remains; by contrast, the TAND gate assumes that faults are temporary and can disappear, and possibly even occur again later. In other words, fault trees model the *occurrence* of faults, whilst the TAND gate is designed to model the *existence* of faults. A further problem is that to be of any practical use, the TAND gate must used to build up larger expressions, and these can rapidly become cumbersome and over-complex. However, the TAND gate does have some advantages, too – it can model repeated events, for example, which traditionally the other methods have struggled with. It is also very simple and, in principle, easy to understand.

There is another solution, older and simpler than any of those mentioned so far: the Priority-AND (or PAND) gate. The Priority-AND gate is designed to impose a sequence on a set of events, so that it is true only if they occur in a certain order. The PAND gate is presented by the *Fault Tree Handbook* as a means of analysing dynamic systems, and as such it is also used in DFTs as well. According to the *Handbook*, "The PRIORITY AND-gate is a special case of the AND-gate in which the output event occurs only if all input events occur in a specified ordered sequence. The sequence is usually shown inside an ellipsis drawn to the right of the gate." (Vesely et al, 1981, p IV-11) The PAND gate has the advantage of being much simpler than any of the solutions presented thus far, and its pedigree is ensured by its inclusion in the *Handbook*.

Unfortunately for the PAND gate, this definition can be ambiguous. For instance, it gives no indication of what sequence should be used if none is specified in an ellipsis (or *conditioning event*), and it fails to address the possibility of events occurring simultaneously or overlapping. Does the PAND gate mean that events have a duration, like the TAND gate requires? Or are they instantaneous? If a PAND gate represents events occurring in a specific sequence, does that mean that an AND gate represents events occurring in any sequence? Furthermore, there is no mention in the *Handbook* of the potential ramifications of incorporating the PAND gate into a fault tree – namely, *contradictions*. What happens if the same event occurs more than once in a PAND gate? Can an event occur 'before' itself? What about an expression like (X PAND Y) AND (Y PAND X)? Presumably this is impossible, since X cannot occur before Y if Y occurs before X, assuming events can only occur once.

With all this confusion, it is no wonder that many cut set analysis algorithms, ranging from older tools such as SETS (Worrell & Stack, 1978) to some versions of more modern software packages like Isograph's FaultTree+, simply treat the PAND gate as a normal AND gate for the purposes of logical reduction. It is often argued that the replacement of a PAND by an AND simply leads only to a conservative prediction of the failure behaviour of the system, but this claim is not necessarily true, as already explained; as seen in Fig 1, a failure of both A and S1 will only cause a failure of the system if S1 does not fail *after* A has failed. Using an AND gate in this instance is not only pessimistic, it is logically incorrect and potentially misleading; in other cases, the results can even be overly optimistic.

Despite this lack of clarity in terms of the qualitative analysis of PAND gates, there are techniques of analysing PAND gates quantitatively. The quantification of sequential failures in a fault tree is known as Sequential Failure Logic, or SFL. It is often necessary to use SFL to aid the quantitative analysis of dynamic systems and has been applied to space satellites, human-robot systems, product liability prevention and more (Long, Sato & Horigome, 1999). There are at least two methods of quantifying SFL: Markov models, such as those used by DFTs, can be used; in addition, the PAQ (Priority-AND Quantification) method, originally proposed by Fussel et al (Fussel, Aber & Rahl, 1976), can be used. There are advantages and disadvantages to both; Markov chains are more complicated, but more accurate, whilst the PAQ method is applicable to a wider range of systems, but is less precise. However, performing quantitative analysis in these cases without first performing a true qualitative analysis can be problematic; a probabilistic analysis of (X PAND Y) AND (Y PAND X) would not result in an probability of 0, despite the fact that it is logically impossible. This can result in an unnecessarily pessimistic reliability estimation.

Before the Priority-AND can be properly used in FTA, it needs to be better defined so that it can be analysed qualitatively. That is the goal of this paper. In Chapter 2, we describe the Pandora methodology, which is aimed at providing a more thorough definition for the Priority-AND gate, allowing qualitative temporal analysis to take place. In Chapter 3, we describe the laws that make this analysis possible, and in Chapter 4, we describe the qualitative analysis process itself. Chapter 5 demonstrates the advantages of the Pandora approach on the example system given above, and finally in Chapter 6 we present our conclusions.

2. INTRODUCING PANDORA

Pandora[1] is a new extension to FTA designed to enable fault trees to model event sequences and relative temporal ordering, which we hope will aid in the analysis of dynamic systems (Walker & Papadopoulos, 2006). To that end, Pandora is built around a redefinition of the Priority-AND (PAND) gate. Many of the problems with the PAND stem from its ambiguous definition in the *Fault Tree Handbook*, which means it is difficult to use successfully in qualitative analysis. The definitions used in Pandora are intended to remove these ambiguities and provide a framework for the qualitative analysis of both PAND gates and other 'temporal' gates, as will be described shortly.

2.1. Events

To address the problems with the *Handbook*'s definition of the PAND gate, it is important to start at the beginning, with the events themselves. The *Handbook* does not define events with time in mind; it says only that a basic event is "a basic initiating fault event that requires no further development." (Vesely et al, 1981, p IV-2) However, we also know that events represent occurrences of faults rather than the existence of faults, and the *Handbook* further states that "under conditions of no repair, a fault that occurs will continue to exist." (Vesely et al, 1981, p V-1) Pandora takes these statements and defines fault tree events to be true at the first moment of occurrence of that event; any subsequent occurrences (i.e. after a fault has been repaired and it fails again) are ignored, since the event is already true. We also assume that events occur instantaneously. Therefore:

- An event in Pandora represents the occurrence of a fault or expected system process, or the transition of the system into a certain state.
- If the fault/state transition has not yet occurred, the event is false.
- When the fault/transition occurs, the event becomes true and will remain true thereafter.
- An event cannot go from being true to false, only from false to true – thus systems are considered unrepairable for the purposes of qualitative analysis.
- An event is instantaneous – it goes instantly from being false to being true.
- To model the cessation of a fault or a state, a second event must be used.

[1] PAND-ORA: Hour or "time" (ORA [*ώρα*] *in Greek*) of PAND gates.

This simplifies the situation and avoids the difficulties that plague the TAND gate; since events do not "un-occur" or cease, it is not possible in Pandora for one event to occur immediately after another event ceases, and nor is it possible for them to partially overlap. Therefore unlike with the TAND gate, which models a number of different temporal relations between two event intervals, in Pandora there are only three possible temporal relations between two events X and Y, and only one can be true at once:

- *Before* - X occurs before Y.
- *Simultaneous* - X occurs at the same time as Y.
- *After* - X occurs after Y, or equivalently, Y occurs before X.

Note that this is strictly a *relative* system of time: we do not know, or need to know, the exact moment at which an event occurs; instead, we only know when an event occurred relative to other events.

2.2. The Conjunctive Temporal Gates

Pandora defines the Priority-AND gate to represent the first of these three relations, the *Before* relation, so that X PAND Y means "X occurs before Y". The Priority-AND gate can also represent the third relation, *After*, by reversing its inputs; thus Y PAND X can also mean "X occurs after Y". To represent the second temporal relation, another gate is needed. The authors of the TAND gate redefine the normal logical AND gate to represent this situation, so that X AND Y means that X and Y both begin and end at the same time (Wijayarathna, Kawat, Santosa & Isogai, 1997). Rather than change the meaning of a well-understood gate like the AND, Pandora introduces a new gate instead, the 'Simultaneous-AND' (or SAND) gate. This gate represents the second temporal relation only, so that it is true only if all of its inputs occur simultaneously; the AND gate retains its usual meaning of being true if all of its inputs occur, regardless of when. Thus, with both the PAND and SAND gate, we can represent all the temporal relations possible between two events in Pandora.

By using these two gates in this way, we avoid the ambiguities of whether or not events can overlap or occur simultaneously in a PAND gate, which was one of the problems with the original definition; it is not possible for a SAND gate and a PAND gate to both be true if they have the same inputs. Further, we say that the Priority-AND gate is true only if its input events all occur in order from left to right, i.e. its leftmost event must occur before its rightmost event. This also avoids the problem of what ordering to use if none is explicitly specified by a conditioning event.

2.3. The Disjunctive Temporal Gate

However, both the PAND and SAND represent temporal conjunctions of events – what about disjunctions? The *Fault Tree Handbook* also includes a strange illustration depicting an Exclusive-OR (or XOR) gate as being equivalent to an OR gate with a conditioning event attached specifying an order, i.e. A XOR B is equal to A OR B assuming A occurs before B (Vesely et al, 1981, p IV-10). This conflicts with the traditional definition of the XOR gate, which states that only one input event must be true – a definition which is also given in the *Handbook* on the same page. However, the traditional Exclusive-OR gate implicitly uses NOT gates to achieve this, as A XOR B is traditionally equivalent to A.¬B + ¬B.A (where +, . and ¬ represent OR, AND and NOT respectively). The *Handbook* does not include NOT gates, and using this definition would result in a more complex non-coherent fault tree, because the occurrence of A and then B would result in the XOR first becoming true and then becoming false again. This also conflicts with the definitions in Pandora, which states that an event (or gate) can only go from being false to being true. Therefore, the *Handbook*'s illustration avoids this by specifying a sequence, i.e. it allows both events to occur, as long as they occur in a certain order. This definition is more like a 'Priority-OR' gate than the traditional XOR gate.

Pandora takes this idea and creates a new gate, the Priority-OR or 'POR' gate, to represent this sort of situation (Walker & Papadopoulos, 2007). The OR gate is true if at least one of its input events is true, and becomes true at that moment. The POR is similar, but gives its first input *priority* (hence its name). For the POR to be true, this leftmost input must also be true. Furthermore, this input must precede all of the other inputs, if they occur. In this it is very similar to the PAND gate. However, the PAND gate is not true until the *last* of its input events is true – and all inputs must occur. The POR becomes true as soon as its *first* input (i.e. its leftmost input) becomes true, as long as none of the other inputs have occurred yet.

In this respect, like the OR, it does not matter whether any of the other input events occur or not once it is true.

2.4. Summary of Temporal Gates

The three gates used in Pandora are summarised below. The fault tree symbols for each gate are given in Fig 2. Note that the POR gate uses the same symbol as the XOR gate is given in the *Handbook*; the upwards arrow inside can be read to signify "priority", as in the PAND gate.

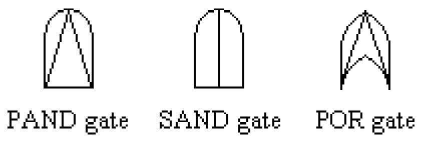

Fig 2: The symbology of the temporal gates

Priority-AND gate (PAND)
Symbol: <
Sequence Value: *S*(A < B) = *S*(B)
Meaning: A occurs before B occurs. Both A and B must occur.

Simultaneous-AND gate (SAND)
Symbol: &
Sequence Value: *S*(A & B) = *S*(A) = *S*(B)
Meaning: B occurs at the same time as A occurs. Both A and B must occur.

Priority-OR gate (POR)
Symbol: |
Sequence Value: *S*(A | B) = *S*(A)
Meaning: Either A occurs and B does not, or both occur and A occurs first.

The meanings given here are only for two input events, but can be extended for any number of events by "nesting" the gates, i.e. A<B<C is equivalent to (A<B)<C, A&B&C is equivalent to (A&B)&C, and A|B|C is equivalent to (A|B)|C.

As stated earlier, Pandora is concerned only with the order in which events occur, not with the exact moment at which they occur. To model this order, we define a property called the *sequence value* of an event, a positive integer that specifies the position of the event in a sequence of other events. This is indicated by *S*, e.g. *S*(X) is the sequence value of X. Given a set of events, S(X)=1 means the event occurs first in the set, S(X)=2 means it occurs second, and so forth. If two events have the same sequence value, then they occur simultaneously; if an event has the sequence value 0, then it means it has not occurred.

These sequence values can be used to construct a *Temporal Truth Table* (TTT), with which the values of temporal expressions can be computed. TTTs are similar to classical Boolean truth tables, except the number of values is extended to account for the sequence values of the events. 0 continues to represent 'false', i.e. an event that has not occurred, but 'true' is now represented by the positive integers, and the value of the integer is the sequence value. We can then use a TTT to define the behaviour of the three temporal gates and two existing logical gates, as shown in Table 1.

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

Table 1: A TTT describing the five gates in Pandora

As can be seen from the table, X.Y is true whenever both X and Y occur, and takes on the maximum sequence value of its inputs in that case (since it is only true once all of its inputs have occurred). X+Y is true whenever at least one of its inputs occurs, and takes on the minimum input sequence value (i.e. the value of the first input to occur). X<Y is true only if X precedes Y, and takes on the sequence value of its right-most input; X&Y is true only if all inputs occur simultaneously and so also has that same sequence value; and X|Y is true if either X is true and Y is not or X occurs before Y, taking on the sequence value of X in either case.

By enabling the computation of the sequence value of an expression, Temporal Truth Tables can also be used to establish the equivalence of two expressions under all possible conditions and therefore prove a new set of temporal laws, as will be seen next.

3. TEMPORAL LAWS

Qualitative analysis of normal fault trees consists of transforming a tree into a group of *cut sets*, which are sets of events that can cause the top event, and then reducing these cut sets by removing any redundancies. This transformation and reduction is done with the aid of Boolean laws of logic, and the result is a group of *minimal* cut sets (MCS); all of the events in a MCS are necessary to cause the top event, and so they contain no redundant events. The key to this process is the fact that a fault tree can also be represented as a logical expression, e.g. (A+C).(B+A), and the Boolean laws can then be used to manipulate and reduce that expression (and thus the corresponding fault tree).

The primary laws used for reduction in normal fault trees are the Absorption and Idempotent Laws. The Absorption Law states that X + (X.Y) ⇔ X, which allows one cut set that contains all the events in another to be eliminated, and the Idempotent Law states that X.X ⇔ X, which means that duplicate events within a cut set can be reduced to just a single occurrence. Thus an expression like (A+C).(B+A) can be reduced to its minimal cut sets like so:

1.	Convert to cut sets (Distributive Laws):	(A+C).(B+A) ⇔ A.B + C.B + A.A + C.A
2.	Minimise using the Idempotent Law:	A.B + C.B + A.A + C.A ⇔ A.B + C.B + A + C.A
3.	Apply the Absorption Law to the cut sets:	A.B + C.B + A + C.A ⇔ C.B + A

The results are the minimal cut sets of the tree. However, in Pandora, we have three new gates as well as the AND and OR gates, and the cut sets may include groups of events that must occur in a specific sequence or must occur simultaneously; effectively, part or all of a cut set might be in a certain sequence. Therefore, in Pandora, cut sets which include temporal gates are termed *cut sequences*, and equivalently a minimal cut set with temporal gates is known as a *minimal cut sequence* (MCSQ). The *Handbook* lists the usual Boolean laws, but gives no information on how these laws should be applied to any gates other than the AND and OR gates. The problem is that these existing Boolean laws, such as the Absorption and Idempotent laws, do not necessarily apply to cut sequences; instead, we need new laws that *do* apply to the temporal gates to allow us to reduce them. These are known as the *temporal laws*.

3.1. Modified Boolean Laws

The first class of temporal laws are Boolean laws which have been modified to work with the three temporal gates. The semantics of the temporal gates often means that these laws do not always work as they did originally; for example, the Boolean Commutative Law states that X.Y ⇔ Y.X; this is still true of the SAND gate, but when applied to the PAND or POR, this is not true, since reversing the order of inputs also means reversing the sequence in which they occur.

Instead, we can use TTTs to determine new equivalences between temporal expressions, and so create temporal versions of the Boolean laws. For example, the two Priority gates are not left distributive over AND; e.g. X < (Y . Z) is not equivalent to (X<Y) . (X<Z) in the same way that X + (Y.Z) ⇔ (X+Y).(X+Z). Instead, it is equivalent to Y.(X<Z) + Z.(X<Y), which can be easily demonstrated using

TTTs. Similarly, X | (Y . Z) ⇔ X|Y + X|Z rather than (X|Y).(X|Z). These are therefore new temporal versions of the Distributive Law. However, the Priority gates are right distributive over AND, e.g. (Y.Z) < X ⇔ (Y<X) . (Z<X), and so the normal Distributive Law still applies in those cases. By contrast, PAND and POR are both left distributive over OR, but not right distributive; for example, X | (Y+Z) ⇔ (X|Y).(X|Y). The SAND gate is not distributive, whether over AND or OR.

This class of temporal laws is very useful for the manipulation and reduction of temporal fault trees in Pandora, since they allow events to be rearranged and duplicates and other redundancies to be removed according to the Absorption and Idempotent Laws, but the introduction of the temporal gates also introduces a new type of possible reduction: *contradiction*. Because the sequence of events is now taken into account, it is possible to have combinations of contradictory sequences such as (X<Y) . (Y<X), which are impossible. The Boolean laws are insufficient to detect this type of situation, because these contradictions arise from the definitions of the temporal gates. Instead, we need another set of new temporal laws. We also need some laws to link the temporal gates with the normal Boolean gates.

3.2. The Completion Laws

This second class of temporal laws come from observations about the definitions of the temporal gates and temporal relations, rather than from existing Boolean laws. The first and most fundamental set of these laws are the Completion Laws, which relate the new temporal gates to the existing AND and OR:

Conjunctive Completion Law

X . Y ⇔ X<Y + X&Y + Y<X

Disjunctive Completion Law

X + Y ⇔ X|Y + X&Y + Y|X

The Conjunctive Completion Law relates PAND and SAND gates to the AND gate, and the Disjunctive Completion Law relates POR and SAND gates to an OR gate. These relationships are perhaps best understood with the aid of some diagrams. Fig 3 illustrates how X.Y consists of three independent parts, each representing one of the three possible temporal relations:

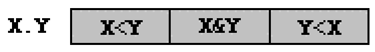

Fig 3: The Conjunctive Completion Law

If two events X and Y are both true (i.e. X.Y is true), then one of either X<Y (*Before*), X&Y (*Simultaneous*), or Y<X (*After*) is true. Intuitively, this is clear: if two events are both true, then they must either occur simultaneously, or one must occur before the other. If all three relations are connected by OR gates, then the combination is equivalent to the AND gate – the exact order in which the two events occur no longer matters, because all three relations are included.

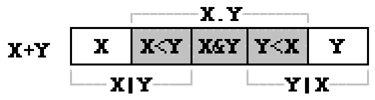

Fig 4: The Disjunctive Completion Law

Fig 4 shows the Disjunctive Completion Law. The shaded area in both figures represents X.Y, and it is possible to see how in Fig 4, X + Y consists of five mutually exclusive areas: either X alone is true, Y alone is true, or one of the three components of X.Y is true. However, four of these areas are covered by two POR gates, X|Y and Y|X; the disjunction of these two gates and a SAND gate is therefore equivalent to X + Y as a whole. The Completion Laws can be proved by using TTTs, as seen in Table 2; if all the values in two columns in a TTT are the same, then the expressions represented by those columns (marked in bold) are equivalent.

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

Table 2: A TTT showing the Completion Laws

There is also a third Completion Law in the table, shown in the rightmost column of Table 2. It can be used to reduce complex temporal expressions when the occurrence of a second event is not important:

X ⇔ X|Y + X&Y + Y<X

This Completion Law uses all three temporal gates, and is particularly useful for eliminating a given factor from a number of different cut sequences (Y in this case). It can also be extended to three events:

```
X ⇔ X|Y|Z + (X&Y)|Z + (X&Z)|Y + X&Y&Z + Y<(X&Z) + Z<(X&Y)
   + (Y<Z)<X + (Y&Z)<X + (Z<Y)<X + (Y<X)|Z + (Z<X)|Y
```

3.3. Further 'Temporal' Laws

The Completion Laws are useful because they allow us to expand Boolean gates into disjunctions of temporal gates and vice versa; they link the two types of gates together. But it is important to remember that each of the three temporal relations are mutually exclusive: only one can be true at once. This is exemplified by the Laws of Mutual Exclusion, a set of laws which indicate possible contradictions stemming from conjunctions of different temporal relations. The three basic ones are:

X<Y . Y<X ⇔ 0
X<Y . X&Y ⇔ 0
Y<X . X&Y ⇔ 0

It is not possible for X to occur both before and after Y, or for it to occur before Y and at the same time as Y. These situations lead to contradictions – they are impossible, and can never be true. Similarly, Mutual Exclusion extends to the POR gate too, e.g. X|Y . Y<X ⇔ 0. Another form of contradiction is also possible, and this happens when the same event occurs more than once as an input to a priority gate (i.e. a PAND or POR gate):

X<X ⇔ 0
X|X ⇔ 0
X&X ⇔ X

These are the Laws of Simultaneity, and are evident from the definitions of the gates: X cannot occur before X. However, it is perhaps best understood by looking at the sequence values; both priority gates are false if any of their inputs have the same sequence value. Since an event appearing twice as an input to a gate will have the same sequence value in each case, the priority gates are always false in these cases, no matter what other inputs they have. By contrast, the SAND gate is not affected: X & X ⇔ X; in other words, it obeys the Idempotent Law as normal.

There is also another set of laws, the Laws of Extension. These laws are important when more than two events are being used, because they explicitly show any temporal relationships which are implied by the temporal gates. For example, if X occurs before Y and Y occurs before Z, then *by extension*, X must also occur before Z. In this case, it is equivalent to saying X<Y<Z. The Laws also apply to the other two gates, like so:

$$X<Y \,.\, Y<Z \Leftrightarrow X<Y \,.\, Y<Z \,.\, X<Z$$
$$X\&Y \,.\, Y\&Z \Leftrightarrow X\&Y \,.\, Y\&Z \,.\, X\&Z$$
$$X|Y \,.\, Y|Z \Leftrightarrow X|Y \,.\, Y|Z \,.\, X|Z$$

The Extension Laws are also used to identify cyclic contradictions, which are expressions that initially appear sound but are in fact contradictory. For example, X<Y . Y<Z . Z<X. This expression contains no immediate contradictions, but if followed through, each event must occur before and after the other two. This can be shown explicitly by first applying the Law of Extension (additions are in bold):

X<Y . Y<Z . Z<X **. X<Z . Y<X . Z<Y**

And now there are contradictions, because we have a conjunction of both X<Y and Y<X for instance.

There are two other laws which can prove useful during reduction: the Laws of POR Transformation and the Laws of Priority. The first are a variant of the Absorption Laws, and deal with the Absorption of a POR gate:

$$X|Y \,.\, Y \Leftrightarrow X<Y$$
$$X|Y + Y \Leftrightarrow X + Y$$

This behaviour contrasts with the usual behaviour of the temporal gates under Absorption, i.e.:

$X<Y \,.\, X \Leftrightarrow X<Y$	$X<Y + X \Leftrightarrow X$
$X\&Y \,.\, X \Leftrightarrow X\&Y$	$X\&Y + X \Leftrightarrow X$
$X\mid Y \,.\, X \Leftrightarrow X\mid Y$	$X\mid Y + X \Leftrightarrow X$

The difference comes from the fact that the left-most input to a POR gate has priority (i.e. it is the only one which must occur in a specific sequence) and all of the others do not; Absorption works differently when these other events are involved, rather than the priority event.

The Priority Laws are related, and describe how combinations of certain pairs of gates mean that one of the gates is redundant, because the other "overrules" it. There are two types: a disjunctive type and a conjunctive type.

Disjunctive	*Conjunctive*
$X<Y + X.Y \Leftrightarrow X.Y$	$X<Y \,.\, X.Y \Leftrightarrow X<Y$
$X\&Y + X.Y \Leftrightarrow X.Y$	$X\&Y \,.\, X.Y \Leftrightarrow X\&Y$
$X\mid Y + X.Y \Leftrightarrow X$	$X\mid Y \,.\, X.Y \Leftrightarrow X<Y$
$X<Y + X\mid Y \Leftrightarrow X\mid Y$	$X<Y \,.\, X\mid Y \Leftrightarrow X<Y$

For the most part, temporal gates overrule ANDs in conjunctions, and AND gates overrule temporal gates in disjunctions. This is because in conjunctions, the AND is implicitly included by the temporal gate (e.g. PAND requires all inputs to occur), whereas in disjunctions, an AND gate implies that any sequence of its inputs is possible, thereby including any temporal gates. However, in the case of the POR, the situation is slightly different, especially when combined with a PAND; in a disjunction, the POR overrules the PAND (because it implicitly includes the PAND), but in a conjunction, the PAND overrules the POR (because the PAND dictates that all events must occur).

These temporal laws, taken together with the definitions used in Pandora, allow for a great deal of manipulation, simplification, and reduction to take place on temporal fault trees, including the facility to detect contradictions when they arise. The temporal laws can be proven using temporal truth tables, which demonstrate equivalence between two (or more) temporal expressions. If the number of events involved renders the use of TTTs impractical, it is possible to use the temporal laws to prove other temporal expressions to be true or false; if it is possible to reach one expression from the other by applying a succession of temporal laws that *have* been proved using TTTs, then the expressions are equivalent.

X	**Y**	**Z**	**X<Y**	**Y<Z**	**X<Z**	**Y<X**	**X&Y**	**X|Y**	**X<Y . Y<X**	**X<Y . Y<Z**	**X<Y.Y<Z .X<Z**	**X|Y . Y**	**X<Y + X|Y**
0	0	0	0	0	0	0	0	0	*0*	**0**	**0**	0	0
0	0	1	0	0	0	0	0	0	*0*	**0**	**0**	0	0
0	1	0	0	0	0	0	0	0	*0*	**0**	**0**	0	0
0	1	1	0	0	0	0	0	0	*0*	**0**	**0**	0	0
0	1	2	0	2	0	0	0	0	*0*	**0**	**0**	0	0
0	2	1	0	0	0	0	0	0	*0*	**0**	**0**	0	0
1	0	0	0	0	0	0	0	1	*0*	**0**	**0**	0	1
1	0	1	0	0	0	0	0	1	*0*	**0**	**0**	0	1
1	0	2	0	0	2	0	0	1	*0*	**0**	**0**	0	1
2	0	1	0	0	0	0	0	2	*0*	**0**	**0**	0	2
1	1	0	0	0	0	0	1	0	*0*	**0**	**0**	0	0
1	2	0	2	0	0	0	0	1	*0*	**0**	**0**	2	1
2	1	0	0	0	0	2	0	0	*0*	**0**	**0**	0	0
1	1	1	0	0	0	0	1	0	*0*	**0**	**0**	0	0
1	1	2	0	2	2	0	1	0	*0*	**0**	**0**	0	0
1	2	1	2	0	0	0	0	1	*0*	**0**	**0**	2	1
2	1	1	0	0	0	2	0	0	*0*	**0**	**0**	0	0
1	2	2	2	0	2	0	0	1	*0*	**0**	**0**	2	1
2	1	2	0	2	0	2	0	0	*0*	**0**	**0**	0	0
2	2	1	0	0	0	0	2	0	*0*	**0**	**0**	0	0
1	2	3	2	3	3	0	0	1	*0*	**3**	**3**	2	1
1	3	2	3	0	2	0	0	1	*0*	**0**	**0**	3	1
2	1	3	0	3	3	2	0	0	*0*	**0**	**0**	0	0
2	3	1	3	0	0	0	0	2	*0*	**0**	**0**	3	2
3	1	2	0	2	0	3	0	0	*0*	**0**	**0**	0	0
3	2	1	0	0	0	3	0	0	*0*	**0**	**0**	0	0

Table 3: A TTT showing some of the other Temporal Laws, with the Extension Law highlighted in bold and Mutual Exclusion in italics

Next, we shall consider how to use these laws in practice for the purposes of qualitative analysis.

4. QUALITATIVE TEMPORAL ANALYSIS

Although these new temporal laws allow us to manipulate and reduce Boolean expressions, applying them is not always simple, especially if the qualitative analysis is to be carried out automatically. While it is possible to manually apply the laws directly, it is easier to first transform the expression into a simplified form which can be more readily manipulated by a smaller number of temporal laws. This form consists of *doublets* and is known as *base temporal form* or BTF (Walker, Bottaci & Papadopoulos, 2007). To transform an expression into minimal cut sequences, the general process is as follows:

1. Convert the expression into a disjunctive normal form (to obtain cut sequences) using Distributive laws.
2. Rearrange expressions so that any temporal gates are beneath AND gates using appropriate temporal/Boolean laws, e.g. (X.Y)<Z ⇔ (X<Z) . (Y<Z).
3. Expand multiple input temporal gates so that each temporal gate has only two children (to obtain the doublets), applying Extension in the process. E.g. (X<Y)<Z ⇔ X<Y . X<Z. Y<Z.
4. Check for any redundancies or contradictions within a cut sequence (such as Mutual Exclusion, Simultaneity, Idempotence), e.g. X<Y.Y ⇔ X<Y and X<Y.Y<X ⇔ 0.
5. Check for any redundant cut sequences by comparing them against each other (using Absorption laws and other temporal laws, e.g. Completion), e.g. X<Y + X|Y ⇔ X|Y.

An expression in base temporal form contains only AND gates, OR gates, and doublets[2]. Doublets are a way of encapsulating the temporal information in an expression so that the expression can be manipulated like a normal non-temporal expression, thus avoiding the problem of having cut sets that contain more than just AND gates. A doublet itself is simply a pair of events connected by a single temporal gate, and is indicated by square brackets, e.g. `[X<Y]`. Any expression containing temporal gates can be converted into a set of doublets connected by either AND or OR gates – cut sequences. In so doing, the doubletisation of an expression also means that complex hierarchies of temporal gates can be flattened so that they only contain doublets and normal events, with no temporal gates below other temporal gates, e.g. `X|(Y&Z)` becomes `[X|Y] + [X|Z] + X.[Y|Z] + X.[Z|Y]`. By simplifying and then abstracting the temporal information out of the cut sequence in this way, it is possible to treat doublets as basic events and reduce them accordingly. It is also easier to see the real temporal relations between events in base temporal form, because every temporal relation between each pair of events is represented by a separate doublet.

To convert an expression into doublets, it must be transformed into a cut set-like format with any OR gates highest in the expression, then AND gates, and then the three temporal gates (preferably in the order POR, PAND, and SAND). This rearranging is done by applying the Distributive, Commutative, and Associative laws as necessary. There are too many of these laws to list them all, but as an example, consider a fault tree represented by the following expression:

```
TOP = (A.((((X<Y)<Z)+B)+((C&D)&E))).(F|(G+H))
```

We can first apply the Associative Laws to remove some of the unnecessary brackets, e.g. (X+Y)+Z ⇔ X+Y+Z. We can also apply two temporal laws, (X<Y)<Z ⇔ X<Y<Z and (X&Y)&Z ⇔ X&Y&Z:

```
TOP = A.((X<Y<Z)+B+(C&D&E)).(F|(G+H))
```

However, this is still not in cut set format, since the top level gate is an AND gate. To be in cut set form, any OR gates must be on top, and all temporal gates at the bottom. To achieve this, we must apply some Distributive laws, such as X . (Y+Z) ⇔ X.Y + X.Z:

```
TOP = A.(X<Y<Z).(F|(G+H)) + A.B.(F|(G+H)) + A.(C&D&E).(F|(G+H))
```

This is almost in cut set form, but we still have OR gates underneath temporal gates. To rearrange this, we need to apply another temporal Distributive law: X|(Y+Z) ⇔ (X|Y).(X|Z):

```
TOP = A.(X<Y<Z).(F|G).(F|H) + A.B.(F|G).(F|H) + A.(C&D&E).(F|G).(F|H)
```

This expression is now in the correct format for doubletisation. The POR gates already only have two children each, so these can be converted directly to doublets; the SAND and PAND gates have three children, and need to be expanded according to the Law of Extension first. The end result is three doubletised cut sequences as follows:

```
TOP = A.[X<Y].[Y<Z].[X<Z].[F|G].[F|H]
    + A.B.[F|G].[F|H]
    + A.[C&D].[D&E].[C&E].[F|G].[F|H]
```

Once an expression is in BTF like this, it is a simple matter to reduce it. First, all events not part of doublets are compared against all the doublets; if any are contained within a doublet, they are redundant and can be removed according to the temporal Absorption Laws (e.g. X.[X<Y] ⇔ [X<Y]). The only exception is the POR Transformation law – if an event occurs on the right of a POR, it is absorbed and the doublet changed to a PAND (e.g. Y.[X|Y] ⇔ [X<Y]).

Next, all the doublets are compared with each other. Because the Extension Law is applied during the conversion of the expression into doublets, all temporal relations are explicitly represented by doublets.

[2] The name "Doublets" is from a word puzzle invented by Lewis Carroll, which first appeared in 1879 in *Vanity*. The aim is to transform one word into another word in as few steps as possible, changing one letter at a time.

Any PAND or POR doublet containing two identical events is an immediate contradiction according to the Laws of Simultaneity; any SAND doublet reduces to just a single event by the same Law (so that [X&X] ⇔ X, for instance). Any two doublets each containing the same pair of events may lead to a reduction of some kind, so we also check for Idempotence, Mutual Exclusion, and Priority, as indicated by Table 3.

If there are no contradictions in the cut sequence, then we can move on; however, if there is a contradiction, either from Simultaneity or Mutual Exclusion, then the whole cut sequence is also contradictory according to the basic Boolean law X.0 ⇔ 0. Then, according to the law X+0 ⇔ X, the entire cut sequence can be removed, effectively eliminating any contradictions.

.	**[X|Y]**	**[Y|X]**	**[X<Y]**	**[Y<X]**	**[X&Y]**	**[Y&X]**
[X|Y]	Idempotent: **[X|Y]**	Mut. Excl.: **0**	Priority: **[X<Y]**	Mut. Excl.: **0**	Mut. Excl.: **0**	Mut. Excl.: **0**
[Y|X]	Mut. Excl.: **0**	Idempotent: **[Y|X]**	Mut. Excl.: **0**	Priority: **[Y<X]**	Mut. Excl.: **0**	Mut. Excl.: **0**
[X<Y]	Priority: **[X<Y]**	Mut. Excl.: **0**	Idempotent: **[X<Y]**	Mut. Excl.: **0**	Mut. Excl.: **0**	Mut. Excl.: **0**
[Y<X]	Mut. Excl.: **0**	Priority: **[Y<X]**	Mut. Excl.: **0**	Idempotent: **[Y<X]**	Mut. Excl.: **0**	Mut. Excl.: **0**
[X&Y]	Mut. Excl.: **0**	Mut. Excl.: **0**	Mut. Excl.: **0**	Mut. Excl.: **0**	Idempotent: **[X&Y]**	Idempotent: **[X&Y]**
[Y&X]	Mut. Excl.: **0**	Mut. Excl.: **0**	Mut. Excl.: **0**	Mut. Excl.: **0**	Idempotent: **[X&Y]**	Idempotent: **[Y&X]**

Table 3: Reduction of two doublets with the same events

Once each cut sequence has been minimised, the final step is to compare the cut sequences against each other. Just as with normal Boolean cut sets, if any cut sequence wholly contains another, then the largest one is redundant, e.g. X.[Y<Z] + [Y<Z] ⇔ [Y<Z]. Extra care must be taken, however, to ensure that the doublets are compared correctly; doublets usually have to contain the same events in the same order to be redundant, so X.[Y<Z] + [Z<Y], for instance, cannot be reduced further. The exceptions to this are SAND gates (in which order is irrelevant, since [X&Y] ⇔ [Y&X]) and instances of the Priority Law, e.g. [X|Y] + [X<Y] ⇔ [X|Y]; if two cut sequence are otherwise the same, but one contains a PAND where the other contains a POR, the PAND cut sequence can be removed; for example:

```
A.[B&C].[C<D] + A.[B&C].[C|D] ⇔ A.[B&C].[C|D]
```

This process will result in the set of minimal cut sequences (MCSQs) for the fault tree.

5. EXAMPLE

To see how Pandora works in practice, consider the example triple-module standby redundant system shown earlier in Fig 1. This system provides a useful illustration of the benefits of Pandora because the standby redundancy modelled in this system is a classic way of achieving fault tolerance. The system is also generic, an example of a design pattern rather than a specific system; the components are abstract and therefore independent of technology, and could be replaced by specific sensors, controllers, actuators and other types of component. Finally, the results obtained from this example can easily be generalised to cover the case of N-module standby redundancy.

In the introduction, we have seen that the order in which components fail in this system can have a large impact on the failure behaviour of the system, an impact which is ignored in traditional FTA. In this section, we show how it is possible to capture the consequences of this behaviour by using Pandora.

For the purposes of failure modelling, we describe the failure behaviour of components in the system using a simplified form of the HiP-HOPS notation (Walker & Papadopoulos, 2006). In HiP-HOPS, faults at component outputs (*output deviations*) are linked to a combination of faults at component inputs (*input deviations*) or internal failures (basic events). Deviations are indicated by a letter describing the failure class, e.g. O for Omission, C for Commission, and then the component, so that `O-S1` means "Omission of signal from Sensor 1". Basic events are indicated in this paper by the component name, so that `A` is an internal failure of the primary component A (a short circuit, for instance). The failure behaviour of the system is therefore as follows:

```
Component A
       O-A         = O-I + A
```
-- Omission of output from A is caused by an omission of input or internal failure.
```
Component S1
       O-S1        = S1<O-A + S1&O-A
```
-- An omission from Sensor 1 is caused by an internal failure before or at the same time as an omission from A (i.e. the sensor is incapable of detecting O-A).
```
       StartB-S1 = O-A
```
-- This is an expected event, rather than a failure, and is the signal to activate standby B.
```
Component B
       Odet-B    = StartB-S1 < (B+O-I)
```
-- Detectable omission from B is caused by a failure of B or a lack of input as long as B has been activated.
```
       Ound-B    = O-S1 + (B+O-I)<StartB-S1 + (B+O-I)&StartB-S1
```
-- Undetectable omission from B is caused by an omission from Sensor 1 or a failure of B or a lack of input before or at the same time as B is activated.
```
Component S2
       O-S2        = S2<Odet-B + S2&Odet-B
```
-- An omission from Sensor 2 is caused by an internal failure before or at the same time as a detectable omission from B (i.e. the sensor is incapable of detecting Odet-B).
```
       StartC-S2 = Odet-B
```
-- This is an expected event, not a failure, and is the signal to activate standby C.
```
Component C
       O-C         = O-S2 + (C+O-I).StartC-S2
```
-- An omission from C is caused by an omission from Sensor 2 (in which case C will not be started) or a failure of C or a lack of input when C is active.
```
Component D
       O-D         = D + Ound-B + O-C
```
-- An omission from D is caused either by an internal failure, an undetectable omission from B, or an omission from C.

Note that the failure behaviour of seemingly identical components may differ depending on the role of these components in a model. Thus, even though components B and C are likely identical in terms of hardware and function, the effects of their failures are different because they have different roles within the system (and so an undetectable failure of B means that C will not be activated, whereas once C is activated, any failure of C will cause system failure, whether undetectable or not).

HiP-HOPS takes this information and constructs a fault tree from it for the top event "O-D", i.e. an omission from the system output, component D. This is done by substituting input deviations for corresponding output deviations, e.g. `O-A` in `StartB-S1` can be replaced by `(A + O-I)`. This fault tree is shown in Fig 5, and the expression represented by the tree is as follows:

```
  O-D = D
      + S1<(O-I + A)                      + S1&(O-I + A)
      + (B+O-I)<(O-I + A)                 + (B+O-I)&(O-I + A)
      + S2<((O-I + A) < (B+O-I))          + S2&((O-I + A) < (B+O-I))
      + (C+O-I).((O-I + A) < (B+O-I)))
```

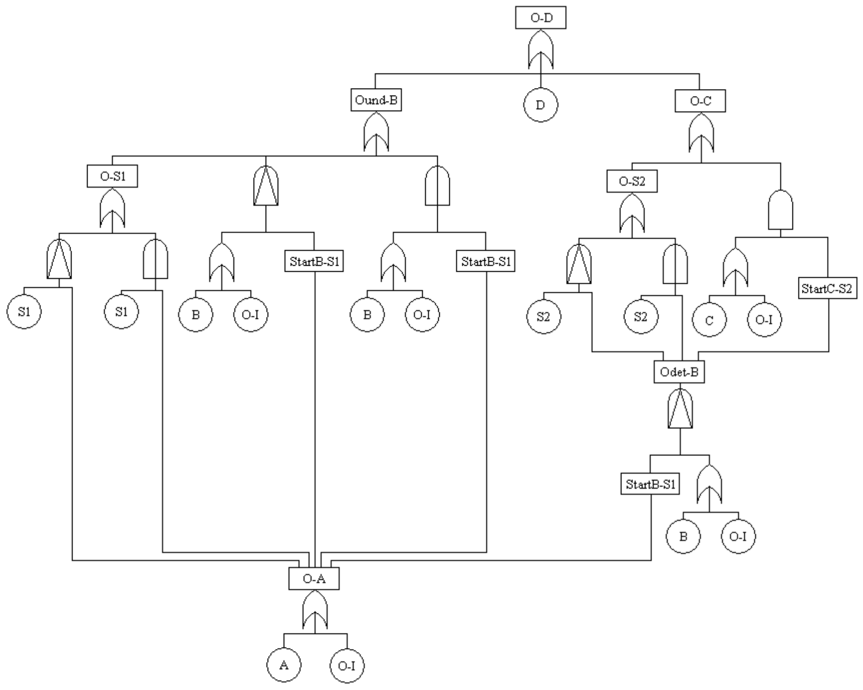

Fig 5: Fault tree for example system in Fig 1.

The first task, as described in the last section, is to rearrange this expression into cut set format, with ORs uppermost and temporal gates lowest. However, because the resulting expression would be too large to present here, we will only show part of this process. For example, taking only the first line of O-D, rearranging `S1<(O-I + A)` requires the temporal Distributive Law `X<(Y+Z) ⇔ (X|Y).(X|Z).(Y+Z)` and rearranging `S1&(O-I+A)` requires the temporal Distributive Law `X&(Y+Z) ⇔ (X&Y&Z)+(X&Y|Z)+(X&Z|Y)`. To rearrange `(B+O-I)<(O-I + A)`, we need to use another Distributive Law first, `(Y+Z)<X ⇔ Y<X + Z<X`. Sometimes we will also need to use the Boolean Distributive Laws too, i.e. `X.(Y+Z) ⇔ X.Y + X.Z`. The first two lines of expression O-D therefore become:

```
O-D = D + (S1|O-I).(S1|A).O-I + (S1|O-I).(S1|A).A
    + (S1&O-I&A) + (S1&O-I|A) + (S1&A|O-I) + (B|O-I).(B|A).O-I
    + (B|O-I).(B|A).A + (O-I|O-I).(O-I|A).O-I + (O-I|O-I).(O-I|A).A  + ...
```

The next step is to convert the expression into doublets, so that it is easier to deal with redundancies and contradictions. In most cases this is straightforward, but if we have temporal gates with more than two basic event inputs, we need to break them up: `(S1&O-I&A)⇔[S1&O-I].[S1&A].[O-I&A]`. This is also true of any multiple temporal gates, e.g. `(S1&O-I|A)` becomes (according to the Laws of Extension): `[S1&O-I].[S1|A].[O-I|A]`.

Once converted into doublets, we check for any possible reductions. First, we check for redundancies within each cut sequence: Idempotence, Simultaneity, POR Transformation and Absorption. Given the first two lines of the original expression, now transformed into doublets:

```
O-D = D + [S1|O-I].[S1|A].O-I + [S1|O-I].[S1|A].A
    + [S1&O-I].[S1&A].[O-I&A] + [S1&O-I].[S1|A].[O-I|A]
    + [S1&A].[S1|O-I].[A|O-I] + [B|O-I].[B|A].O-I
    + [B|O-I].[B|A].A + [O-I|O-I].[O-I|A].O-I + [O-I|O-I].[O-I|A].A
```

We can see two places where the Law of Simultaneity is being violated (`[O-I|O-I] ⇔ 0`) and instances of POR Transformation, e.g.: `[S1|O-I].[S1|A].O-I ⇔ [S1<O-I].[S1|A]`. Once this phase is complete, the result is a set of doubletised cut sequences that each contain no redundancies:

```
O-D = D
    + [S1<O-I] . [S1|A]
    + [S1|O-I] . [S1<A]
    + [S1&O-I] . [S1&A] . [O-I&A]
    + [S1&O-I] . [S1|A] . [O-I|A]
    + [S1&A] . [S1|O-I] . [A|O-I]
    + [B<O-I] . [B|A]
    + [B|O-I] . [B<A]
    + [B&O-I] . [B&A] . [O-I&A]
    + [B&O-I] . [B|A] . [O-I|A]
    + [B&A] . [B|O-I] . [A|O-I]
    + [O-I&A] . [O-I|B] . [A|B]
    + [O-I|A] . [O-I|B]
    + S2<(A<B . A|O-I) . S2|(A|B . A<O-I) . (A<B . A|O-I)
    + S2|(A<B . A|O-I) . S2<(A|B . A<O-I) . (A|B . A<O-I)
    + S2 & (A<B . A|O-I) & (A|B . A<O-I)
    + S2 & (A<B . A|O-I) | (A|B . A<O-I)
    + S2 & (A|B . A<O-I) | (A<B . A|O-I)
    + C . [A<B] . [A|O-I]
    + C . [A|B] . [A<O-I]
    + [A|B] . [A<O-I]
```

The next step is to compare cut sequences against each other, to look for any redundancies or possible places to apply the Completion Law. Both are possible; for example, we can reduce seven of these cut sequences to just one by applying the third Completion Law, taking `O-I` as `X`, `A` as `Y`, and `B` as `Z`:

`O-I` ⇔ `[B<O-I].[B|A]`	*contains* `Z<X|Y + Z<Y<X + Z<X&Y`
`+ [B&O-I].[B&A].[O-I&A]`	*contains* `Z&X&Y`
`+ [B&O-I].[B|A].[O-I|A]`	*contains* `Z&X|Y`
`+ [B&A].[B|O-I].[A|O-I]`	*contains* `Y&Z<X`
`+ [O-I&A].[O-I|B].[A|B]`	*contains* `X&Y|Z`
`+ [O-I|A].[O-I|B]`	*contains* `X|Y|Z` (*since* `X|Y|Z` ⇔ `X|Y.X|Z`)
`+ [A|B].[A<O-I]`	*contains* `Y<X|Z + Y<X&Z + Y<Z<X`

This results in `O-I`, which we can now use to reduce any cut sequence containing it according to the Absorption or POR Transformation laws, and sometimes both, e.g.:

`O-I + C.[A<B].[A|O-I]` ⇔ `O-I + C.[A<B]`	*(POR Transformation, Absorption)*
`O-I + [S1<O-I] . [S1|A]` ⇔ `O-I`	*(Absorption –* `X + X<Y` ⇔ `X`)

Ultimately, we arrive at just nine minimal cut sequences:

1. `O-I`	*Omission of input to system*
2. `D`	*Failure of component D (output component)*
3. `[S1<A]`	*Failure of sensor S1 before it can detect failure of A*
4. `[S1&A]`	*Failure of sensor S1 at the same time as A fails*
5. `[B<A]`	*Failure of standby component B before primary component A*
6. `[B&A]`	*Failure of component B at the same time as failure of A*
7. `[A<B].[A<S2].[S2&B]`	*Failure of A before a simultaneous failure of B and S2*
8. `[A<B].[A<S2].[S2<B]`	*Failure of A before sensor S2 fails, which in turn fails before B*
9. `C.[A<B]`	*Failure of A before B in conjunction with a failure of C*

This can be contrasted with the results of a classical FTA:

1. `O-I`	*Omission of input to system*
2. `D`	*Failure of component D*
3. `A.B.C`	*Failure of primary component A and both standbys, B and C*
4. `A.S1`	*Failure of primary component A and sensor S1*
5. `A.B.S2`	*Failure of A and standby component B as well as sensor S2*

These classical results overlook many of the subtleties of the example system. The results provided by Pandora show us that the order in which A and B fail is critical, since those two events appear together in five of the MCSQs; if B fails before or at the same time as A, then the whole system will fail, but according to the classical analysis, either C or S2 must also fail for this to occur. In fact, Pandora tells us that a failure of all three of A, B and C will result in system failure only if A failed before B. Similarly, a failure of all of A, B and S2 is necessary only if A failed before B, and B fails after or at the same time as S2; a failure of B before S2 will not cause the system to fail, since C has already been activated by then. Lastly, we can now see that a failure of S1 after A will not cause the system to fail on its own, which is contrary to the classical results. The Pandora results are thus significantly more accurate.

The results obtained by Pandora differ in several ways from those possible using other approaches. DFTs are primarily a quantitative analysis technique and so would not be able to produce minimal cut sequences in this way, although if probabilistic failure data was available, it would presumably be possible to arrive at an estimate for the system reliability. The TFT approach is meant for a system in operation, not a model of a system under design; similarly, using the TAND gate would mean representing events as having a finite duration, which means making different assumptions about the system behaviour and this would likely produce different results.

To further evaluate the potential of Pandora, we currently apply the technique on a model of a multi-modal vehicle steer-by-wire system, which we have previously analysed using traditional combinatorial FTA techniques (Parker, Walker, Papadopoulos & Grante, 2006). Early results from a partial analysis of this model and comparisons with results gained via classical FTA show that similar benefits can be expected to those displayed in the example, in terms of improving the clarity, correctness and accuracy of the analysis. However, because the analysis in Pandora is of a much finer grain, issues of computational complexity must be addressed in the future with the development of efficient analysis algorithms, so that an automated analysis process becomes feasible for large and complex systems.

6. CONCLUSION

It has long been recognised in the *Fault Tree Handbook* that classical combinatorial fault trees cannot capture the potentially critical significance of the temporal ordering of failures in a system. Although the *Handbook* proposed dynamic gates to address the problem, these gates were rarely used due to the lack of any information in the *Handbook* on how to perform qualitative or quantitative analysis on them.

In this paper, we presented Pandora, an approach that enables modelling and analysis of dynamic failure behaviour in systems via extensions to Boolean logic, as was originally envisaged in the *Fault Tree Handbook*. Pandora uses only 3 gates to describe temporal relationships between events, a PAND, a POR and a SAND gate; it also defines the simple notion of a relative ordering of events, in which it is only necessary to determine whether one event came before, after, or at the same time as another. Furthermore, attempts at a formal underpinning of these gates in Pandora can be found in (Merle & Roussel, 2007) and (Walker, Bottaci, Papadopoulos, 2007).

Temporal order is represented using a sequence value, which is an integer indicating the order in which an event occurs within a set. We have shown that sequence values can be used in Temporal Truth Tables (TTTs) to compute the truth value of expressions that connect events with classical Boolean and temporal operators. TTTs have been used to prove a set of 'temporal' laws in Pandora logic, including extended versions of traditional Boolean laws like the Distributive and Absorption laws as well as new laws specific to the PAND, SAND and POR gates. Qualitative analysis of temporal fault trees in Pandora is possible by using these temporal laws. The result is a set of minimal cut sequences (MCSQs), which are analogous to traditional minimal cut sets, except for the fact that they contain temporal gates. MCSQs therefore represent the smallest *sequence* of events necessary to cause the top event, and are more accurate and more informative than traditional minimal cut sets. This is important not in only itself but also as a precursor for later quantitative analysis, as it should result in a less pessimistic estimate of system reliability.

In the future, we hope to improve the performance of the qualitative analysis algorithm in an attempt to match the efficiency of existing non-temporal FTA techniques, which will allow Pandora to be applied to larger and more complex systems. In addition, we plan to extend Pandora with quantitative analysis capabilities. The goal is to produce a comprehensive tool for fault tree analysis of dynamic systems.

REFERENCES

Fussel JB, Aber EF, Rahl RG. On the quantitative analysis of Priority-AND failure logic. IEEE Transactions on Reliability; 1976; Vol R-25/5:324-326.

Long W, Sato Y, Horigome M. Quantification of sequential failure logic for fault tree analysis. Reliability Engineering & System Safety; 2000; Vol 67: 269-274.

Manian R, Dugan JB, Coppit D, Sullivan KJ. Combining various solution techniques for dynamic fault tree analysis of computer systems. Third IEEE International High-Assurance Systems Engineering Symposium, Washington DC; Nov 13-14 1998; p21.

Merle G, Roussel J-M. Algebraic Modelling of Temporal Fault Trees. 1st IFAC workshop on Dependable Control of Discrete Systems (DCDS'07), Paris, France. June 2007; 175-180.

Palshikar GK. Temporal Fault Trees. Information and Software Technology; 2002; Vol 44: 137-150.

Parker D, Walker M, Papadopoulos Y, Grante C. Component-based, automated FMEA of advanced active safety systems. FISITA’06, 31st World Automotive Congress, Yokohama, 2006.

Sullivan K, Dugan J, Coppit D. The Galileo Fault Tree Analysis Tool. Proceedings of IEEE International Symposium of Fault Tolerant Computing, FTC-29; June 1999; 232-235.

Tang Z, Dugan JB. Minimal cut set/sequence generation for dynamic fault trees. Annual Reliability and Maintainability Symposium Proceedings; 2004.

Vesely WE, Goldberg FF, Roberts NH, Haasl DF. Fault Tree Handbook. Washington D.C., USA. US Nuclear Regulatory Commission; 1981.

Vesely WE., Stamatelatos M, Dugan JB, Fragola J, Minarick J, Railsback J Fault Tree Handbook with Aerospace Applications. NASA Office of Safety and Mission Assurance, USA; 2002

Walker M, Bottaci L, Papadopoulos YI. Compositional Temporal Fault Tree Analysis. SAFECOMP 2007, in Computer Safety, Reliability, and Security; Springer-Berlin/Heidelberg, 2007; 106-119.

Walker M, Papadopoulos YI. Pandora 2: The Time of Priority-OR gates. 1st IFAC workshop on Dependable Control of Discrete Systems (DCDS'07), Paris, France. June 2007; 169-174.

Walker M, Papadopoulos YI. Pandora: The Time of Priority-AND gates. INCOM 2006, France; 2006; 237-242.

Wijayarathna PG, Kawat Y, Santosa A, Isogai K. Representing relative temporal knowledge with the tand connective. In ‘Eight Ireland Conference on Artificial Intelligence (AI-97); Sept 1997; Vol 2: 80-87.

Wijayarathna PG, Maekawa M. Extending Fault Trees with an AND-THEN gate. Proceedings of the 11th International Symposium on Software Reliability Engineering (ISSRE’00); 2000.

Worrell RB, Stack DW. A SETS User Manual for the Fault Tree Analyst. NUREG CR-04651; 1978.