Reasoning About Temporal Faults Using an Activation Logic

André Didier(✉) and Alexandre Mota

Cidade Universitária, Av. Jornalista Anibal Fernandes,
s/n, Recife, PE 50740-560, Brazil
alrd@cin.ufpe.br

Abstract. Faults modelling is essential to anticipate failures in critical systems. Traditionally, Static Fault Trees (SFTs) are employed to this end, but Temporal and Dynamic Fault Trees (TFTs and DFTs) are gaining evidence due to their enriched power to model and detect intricate propagation of faults that lead to a failure. SFTs structure can be abstracted to Boolean expressions. An algebra with an operator to express order is needed to abstract TFT and DFT structures. These expressions for SFT, TFT, and DFT are called structure expressions.

Architectural modelling languages, such as Architecture and Analysis Design Language (AADL), have been used to model components and systems relations, including modelling of faults, errors, failures, and fault propagation. AADL tools can perform Static Fault Tree Analysis, for the faults modelled using AADL's Error Model Annex.

In previous work, we showed an Algebra of Temporal Faults to analyse the order of occurrence of faults extending Boolean algebra to perform analysis for Temporal and Dynamic fault trees. In this work, we show a parametrized logic to express nominal and erroneous behaviours, including faults modelling, provided an algebra and a set of operational modes. We show how to use this logic together with the Algebra of Temporal Faults to analyse the occurrence of faults as well as their order and propagation. The logic created in this work is intended to help analysts to consider all possible situations in complex expressions with order-related operators, avoiding to miss some subtle (but relevant) combination.

Keywords: Activation logic · Algebra of temporal faults · Dynamic fault trees · Boolean algebra

1 Introduction

The development process of critical control systems is based essentially on the rigorous execution of guides and regulations [1,4,10,11]. Specialised agencies (like FAA, EASA and ANAC in the aviation field) use these guides and regulations to certify such systems.

Safety plays a crucial role on critical systems and it is the responsibility of the safety assessment process. ARP-4761 [1] defines several techniques to

© Springer International Publishing AG 2018
S.H. Rubin and T. Bouabana-Tebibel (eds.), *Quality Software Through Reuse and Integration*, Advances in Intelligent Systems and Computing 561,
DOI 10.1007/978-3-319-56157-8_13

perform safety assessment. One of them is Fault Tree Analysis (FTA). It is a deductive process that uses trees to model faults and their dependencies and propagation. In such trees, the premises are the leaves (basic events) and the conclusions are the roots (top events). Intermediary events use gates to combine basic events and each kind of gate has its own combination semantics definition. For example, the most traditional gates are OR and AND. They combine the events as *at least one shall occur* and *all shall occur*, respectively. To analyse fault trees, their structures are then abstracted as Boolean expressions called *structure expressions*. The analysis with these two traditional gates uses a well-defined algorithm based on Shannon's method—which originated the Binary Decision Diagrams (BDDs) [3,6]—to obtain minimal cut sets from the structure expressions and a general formula to calculate the probability of top events.

Besides the traditional OR and AND gates, the Fault Tree Handbook defines other gates. For example the Priority-AND gate, which considers the order of occurrence of events. Although the work reported in [24] defines these new gates, there is no algorithm to perform the analysis of trees that contain such new gates. This motivated the introduction of two new kinds of fault trees: Dynamic Fault Trees [9] (DFTs) and Temporal Fault Trees [26,27] (TFTs). These variant trees can capture sequence dependencies of fault events in a system. The difference from Temporal Fault Tree [26,27] (TFT) to Dynamic Fault Tree [9] (DFT) is that Temporal Fault Trees [26,27] (TFTs) use temporal gates directly, while Dynamic Fault Trees [9] (DFTs) do not—Dynamic Fault Trees [9] (DFTs) gates are an abstraction of temporal gates. To differentiate traditional fault trees from the other two, we will call traditional fault trees as Static Fault Trees (SFTs).

Both TFT and DFT also use structure expressions ([19,27], respectively) to abstract the tree to enable their analyses. Despite some limitations related to spare gates [19], the structure expressions used in TFTs and DFTs can be formulated in terms of a generic order-based operator.

The NOT operator is absent in the algebras showed in [16,18,25,27]. They conceptually remove such an operator to avoid incorrect analysis, as there is no consensus about the relevance of its use: (i) it can be misleading, generating non-coherent analysis [22], or (ii) it can be essential in practical use [5]. The algebra created in our previous work [8] defines the NOT operator and allows its use.

In structure expressions, the variables represent fault events and the expressions represent a top event, an operational mode of a system. The combination of all operational modes is expected to describe the complete behaviour of a system. For example, if no fault occurs, then the system is in a nominal state; if all faults occur, then the system is definitely in a faulty state. Possibilities in between vary accordingly to the fault tolerance strategies employed in a system. The analysis of all possibilities is what we call *completeness analysis*. For Boolean algebra, it is equivalent to verify if all rows in a truth table (in which the variables are fault events) are considered in at least one structure expression.

Architecture Analysis and Design Language [12] (AADL) is a standard language to model (among other features) system structure and component interaction. AADL has several tools to perform different analyses to obtain SFT to perform FTA. But AADL's assertions framework does not express order explicitly as needed for TFT and DFT analyses.

On the analyses of systems and its constituents, there is a distinction of operational modes and error events. Operational modes refer to the behaviour that is perceived on the boundaries of a system. Error events, on the other hand, represent the behaviour detected in a constituent of a system. Such error events may relate to an operational mode, but not necessarily. We abstract these differences and leave the distinction as a parameter. In this article, we refer to such a set as *operational modes*.

Another concern, left untreated in the literature, is the undesirable possibility of non-determinism in structure expressions. What guarantees can we provide to detect non-determinism in erroneous behaviour? For example, if a commission is observed when fault A is active and an omission is observed when faults A and B are active, then the system may behave non-deterministically with a commission or omission when both A and B are active (A and B implies A). In this work we show three different approaches to check the non-determinism: (i) verify its existence, (ii) indicate which set of operational modes are active for a combination of faults, or (iii) what is the combination of faults that activates a set of operational modes.

Writing and analysing expressions with order-related operators is more complex than analysing expressions with Boolean operators only. In this work, we define a formal Activation Logic (AL) that works together with an inner algebra to perform analysis of system structure and component interaction with a focus on fault modelling and fault propagation, tackling the complexity introduced by order-related operators. AL receives an algebra and the set of operational modes of a system as parameters. The choice of algebra defines which structure expressions can be obtained: if Boolean algebra is passed as a parameter, the AL can generate structure expressions with Boolean operators (SFT); if the Algebra of Temporal Faults [8] (ATF) is passed as a parameter, the AL can generate structure expressions with order-related operators (TFT and DFT). The AL requires that the inner algebras provide a set of properties (tautology and contradiction) and semantic values. The use of the NOT is essential: besides its use in expressions, we use the complement to normalise the expressions to provide *healthy* expressions. To obtain critical event expressions used in FTs and to denote faults propagation, the AL provides a predicates notation and verification of non-determinism.

This paper is organised as follows: in Sect. 2 we show the concepts used as the basis for this work. Section 3 presents the proposed AL, and Sect. 4 the case study and the application of the proposed AL. Finally, we present our conclusions and future work in Sect. 5.

2 Background

Faults modelling depends on which analyses we want to perform. For instance, in fault trees, even if a fault can be repaired, it is considered as a nonrepairable fault. A fault tree is a snapshot[1] of a system's, subsystem's or component's topology of faults. The time relations on fault events in TFTs and DFTs allows the analysis of different configurations (or snapshots) of a system, subsystem or component. We discuss these time relations in Sect. 2.1.

Structure expressions are used to analyse fault trees. In general, a structure expression is obtained from gates semantics and basic events. Basic events become variables and gates become operators (a gate may become one or more operators). In Sect. 2.2 we explain these structure expressions for SFTs, TFTs and DFTs.

The AL proposed in this work depends on algebraic rules and relies on a *complement* operator. Our previous work showed the ATF that extends the Boolean algebra, thus providing the NOT operator and some properties and rules to use the algebra. In Sect. 2.3 we show these properties and rules used in this work.

2.1 Time Relation of Fault Events

The most general case for time relations is to consider that each fault event has a continuous time duration. They are the basis on how fault events discretization are defined. The point of view in this work is the analysis of the effects caused by a combination of faults in a snapshot of a system state. In Fig. 1 we show all possibilities of events relations in a continuous timeline (from A to B; the converse relation is similar):

a. A starts before and ends after B has started, but before B has ended;
b. A starts before B and ends after B has ended (A contains B);
c. B starts after A, but they end at the same time;
d. A and B start at the same time, but A ends before B;
e. A and B start and end at the same time;
f. A starts before B and ends when B starts.
g. A starts and ends before B starts;

Considering that fault occurrence corresponds to the start of a fault event and its duration, from Fig. 1 we clearly identify which event comes first: A comes before B, except in the cases (d) and (e), where they start exactly at the same time. If fault events are independent (they are not susceptible to have a common cause) then the probability of their occurrences starting at the same time is very low. The relations (f) and (g) shows the case that the system was repaired, thus A is not active when B starts. In Sect. 2.3 we show that the ATF abstracts the relation of events in continuous time as an *exclusive before* relation, based on fault *occurrence* (it is similar—at least implicitly—to what is reported in [17,27]).

[1] Whether a top event indeed causes a catastrophic or major failure is out of the scope of this paper; we consider that, if it is possible that such failure occurs, then it will.

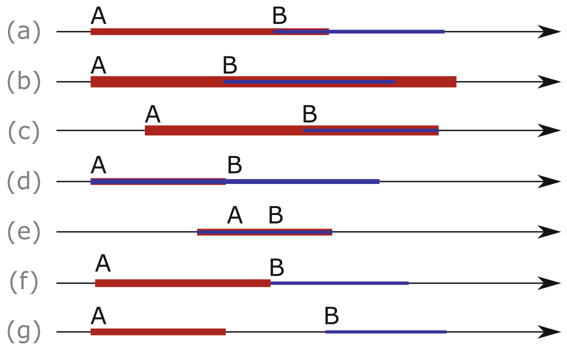

Fig. 1. Relation of two events with duration

2.2 Structure Expressions

Structure expressions in FTA are defined in terms of set theory, using symbols for fault events occurrence. If a fault event symbol is in a set, then it means that fault has occurred. A set is a combination of fault events that causes the top-level event of a tree. A structure expression of a tree is denoted by a set of sets of fault event combinations. The OR gate becomes the union operator between sets and the AND gate, the intersection. For example, if a system contains fault events a, b, and c, fault trees for this system contain at most all these three events. The occurrence of the fault event a is denoted by a set of sets A, which contains the following sets:

1. $\{a\}$: only a occurs;
2. $\{a, b\}$: a and b occur;
3. $\{a, c\}$: a and c occur;
4. $\{a, b, c\}$: all three events occur.

The fault tree in Fig. 2 contains only two events and the resulting structure expression for this tree is the expression $A \cap B$ (TOP), where A and B are the sets of sets that contain a and b, respectively. The resulting combinations for TOP are $\{a, b\}$ and $\{a, b, c\}$ (fault events a and b occur in all possibilities).

After obtaining structure expressions, the next step is to reduce the expressions to a canonical form to obtain the *minimal cut sets*. Minimal cut sets are the sets that contain the minimum and sufficient events to activate the top-level failure. That is, minimal cut sets are the smallest sets of fault events that, if all occur, cause the top-level failure to occur. Probabilistic analysis is then performed on these events to obtain the overall probability of occurrence of the top-level event. The Fault Tree Handbook shows an algorithm based on Shannon's method to reduce structure expressions to obtain minimal cut sets. The Boolean expression of the tree shown in Fig. 2 is $TOP = A \wedge B$.

Structure expressions are also present in TFTs [25,27,28], through the Pandora[2] methodology. These expressions use the FTA operators OR and AND,

[2] Pandora stands for: P-AND-ORA, which translates to Priority AND, Time.

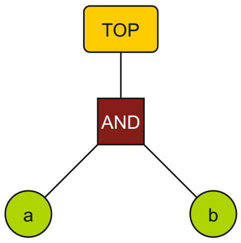

Fig. 2. Very simple example of a fault tree

and three new operators related to events ordering: Priority-AND (PAND), Priority-OR (POR), and Simultaneous-AND (SAND). The semantics of the PAND in TFTs is similar to the semantics of the Priority-AND described in the Fault Tree Handbook. To avoid ambiguous expressions, the semantics in TFTs is stated in terms of natural numbers (instead of Boolean values), using a *sequence value* function. For every possibility it assigns a sequence value to each fault event. For example, if event A occurs before event B, then the sequence value of A is lower than the sequence value of B, and one formula to express this is A PAND B.

In TFTs, an invariant on sequence values is that there are no gaps for assigned values higher than zero. For example, if faults A and B occur at the same time and there are only these two events, then they should both be assigned value 1. On the other hand, if A occurs before B, then the assigned values are 1 and 2, respectively. Value zero means that the event is not active in the combination. Table 1 shows the semantics of all TFT operators with sequence values.

Table 1. TFT operators and sequence value numbers

A	B	AND	OR	PAND	POR	SAND
0	0	0	0	0	0	0
0	1	0	1	0	0	0
1	0	0	1	0	1	0
1	1	1	1	0	0	1
1	2	2	1	2	1	0
2	1	2	1	0	0	0

The reduction of TFT expressions is achieved using dependency trees. In a dependency tree, if all children of a tree node are true, then the node is also true. Conversely, if a node is true, then all its children are also true. An issue with dependency trees is that they grow exponentially. Accordingly to the work reported in [28], it is already infeasible to deal with seven fault events in TFTs. They use an alternative solution based on modularisation and algebraic laws [27] to tackle this.

Structure expressions are also used in DFTs. In [16,17,20] fault events occur in a specific time and are instantaneous, stated through a *date-of-occurrence* function. As the date-of-occurrence function is stated in continuous time, the probability of two events occurring at the same time is negligible. In fact, useful information is obtained from the possibilities of relation in time of the occurrence of the events.

The work reported in [16,17,20] describe an algebra with operators OR and AND, and three new operators to express events ordering: (i) non-inclusive-before, (ii) simultaneous, and (iii) inclusive-before. The non-inclusive-before and the simultaneous operators are similar to TFT's POR and SAND operators, respectively (although in [16,17,20] the only *true* result with the simultaneous operator happens if the operands are the same). The inclusive-before is a composition of the non-inclusive-before and the simultaneous operators.

The work reported in [23,29] shows the top-level events probability calculation for DFTs by converting them to a simplified version, using only order-based operators. Such a simplified version, which is based on a modified BDD that includes an order-based operator, creates Sequential BDDs that are used to perform the probabilistic analysis.

From the previous explanation, we can conclude that an order-based operator is present on the analyses of TFTs and DFTs. Each approach describes a new algebra (without the NOT operator) based on different representations of events ordering with similar theorems to reduce expressions to a canonical form.

2.3 The Algebra of Temporal Faults

Recall from Sects. 2.1 and 2.2 that fault events are independent on one another if the events are not susceptible to a common cause. Also, the simultaneity of events is probabilistically impossible, so one event occurs exclusively before or after another one, inducing an order of occurrence of events. Moreover, the analysis of fault events considers that they have started and are active, as a snapshot of a system (faulty) state. Thus, the ATF is not used to analyse the effects of a repairable fault. For example, the cases that are possible to analyse with the theory shown here are (a), (b) and (c) of Fig. 1, in Sect. 2.1.

The set-theoretical abstraction of structure expressions for SFTs [24, pp. VI-11] is very close to a Free Boolean Algebra [13, pp. 256–266] (FBA), where each generator in FBA corresponds to a fault event symbol in fault trees. In FBAs, as generators are "free", they are independent on one another and Boolean formulas are written as a set of sets of possibilities, which are similar to the structure expressions of SFTs.

The set of sets for FBAs is the denotational semantics for Boolean algebras. We use the concept of generators to define the denotational semantics of ATF using a set of lists without repetition (distinct lists). The choice of lists is because this structure inherently associates a generator to an index, making implicit the representation of order. These lists are composed by non-repeated elements because the events in fault trees are non-repairable, thus they do not occur more than once.

In the following, we show the definitions and laws of the ATF used in Sect. 4. To avoid repetition, let S, T and U be sets of distinct lists. A list xs is distinct if it has no repeated element. So, if x is in xs, then it has a unique associated index i and we denote it as $x = xs_i$.

The ATF form a free algebra, similarly to FBAs. *Infimum* and *Supremum* are defined as set intersection ($\cap$) and union ($\cup$) respectively. The order within the algebra is defined with set inclusion ($\subseteq$).

To distinguish the permutations that are not defined in FBA, we need a new operator. The definition of XBefore ($\rightarrow$) is given in terms of list concatenation:

$$\begin{aligned} S \rightarrow T \ &= \{zs | \exists xs, ys \bullet (\mathbf{set}\, xs) \cap (\mathbf{set}\, ys) = \{\} \\ &\qquad \wedge xs \in S \wedge ys \in T \wedge zs = xs@ys\} \end{aligned} \tag{1}$$

where the **set** function returns the set of the elements of a list, and @ concatenates two lists.

In some cases it is more intuitive to use the XBefore definition in terms of lists slicing because it uses indexes explicitly. Lists slicing is the operation of taking or dropping elements, obtaining a sublist. In slicing, the starting index is inclusive, and the ending is exclusive. Thus the first index is 0 and the last index is the list length. For example, the list $xs_{[i..|xs|]}$ is equal to the xs list, where $|xs|$ is the list length. We use the following notation for list slicing:

$$xs_{[i..j]} = \text{starts at } i \text{ and ends at } j-1 \tag{2a}$$

$$xs_{[..j]} = xs_{[0..j]} \tag{2b}$$

$$xs_{[i..]} = xs_{[i..|xs|]} \tag{2c}$$

List slicing and concatenation are complementary: concatenating two consecutive slices results in the original list:

$$\forall i \bullet xs_{[..i]}@xs_{[i..]} = xs \tag{3}$$

There is an equivalent definition of XBefore with concatenation using lists slicing:

$$S \rightarrow T = \{zs | \exists i \bullet zs_{[..i]} \in S \wedge zs_{[i..]} \in T\} \tag{4}$$

A variable in ATF is defined by one generator, and denotes its occurrence:

$$\mathbf{var}\, x = \{zs | x \in zs\} \tag{5}$$

The following expressions are sufficient to define the ATF in terms of an inductively defined set (**atf**):

$$\mathbf{var}\, x \in \mathbf{atf} \qquad \text{Variable} \tag{6a}$$

$$S \in \mathbf{atf} \implies -S \in \mathbf{atf} \qquad \text{Complement, Negation} \tag{6b}$$

$$S \in \mathbf{atf} \wedge T \in \mathbf{atf} \implies S \cap T \in \mathbf{atf} \qquad \text{Intersection, } \textit{Infimum} \tag{6c}$$

$$S \in \mathbf{atf} \wedge T \in \mathbf{atf} \implies S \rightarrow T \in \mathbf{atf} \qquad \text{XBefore} \tag{6d}$$

Following the definitions, the expressions below are also valid for **atf** (using DeMorgan laws):

$$UNIV \in \textbf{atf} \qquad \text{Universal set, True} \tag{6e}$$

$$\{\} \in \textbf{atf} \qquad \text{Empty set, False} \tag{6f}$$

$$S \in \textbf{atf} \land T \in \textbf{atf} \implies S \cup T \in \textbf{atf} \qquad \text{Union, } \textit{Supremum} \tag{6g}$$

The following expressions are valid for generators a and b and are sufficient to show that the generators are independent:

$$\textbf{var}\, a = \textbf{var}\, b \iff a = b \tag{7a}$$

$$\textbf{var}\, a \not\subseteq -\textbf{var}\, b \tag{7b}$$

$$\textbf{var}\, a \neq -\textbf{var}\, b \tag{7c}$$

$$-\textbf{var}\, a \not\subseteq \textbf{var}\, b \tag{7d}$$

$$-\textbf{var}\, a \neq \textbf{var}\, b \tag{7e}$$

Expressions (6a) to (6g) and (7a) to (7e) imply that the ATF without the XBefore operator (1) forms a Boolean algebra based on sets of lists. And this is also equivalent to an FBA with the same generators.

Note that, as the ATF is a conservative extension of a Boolean algebra, the NOT operator is defined here, so expressions in the ATF can use it.

In the following section, we show properties as a generalisation of the preconditions of laws related to XBefore.

Temporal properties (*tempo*). Temporal properties give a more abstract and less restrictive shape on the XBefore laws. These properties avoid the requirement that every operand of XBefore should be a variable (5).

The first temporal property is about disjoint split. If the first part of a list is in a given set, then every remainder part is not. So, if a generator is in the beginning of a list, it must not be at the ending (and vice-versa).

$$\textbf{tempo}_1 S = \forall i, j, zs \bullet i \leq j \implies \neg \left(zs_{[..i]} \in S \land zs_{[j..]} \in S\right) \tag{8a}$$

$$\textbf{tempo}_2 S = \forall i, zs \bullet zs \in S \iff zs_{[..i]} \in S \lor zs_{[i..]} \in S \tag{8b}$$

$$\textbf{tempo}_3 S = \forall i, j, zs \bullet j < i \implies \left(zs_{[j..i]} \in S \iff zs_{[..i]} \in S \land zs_{[j..]} \in S\right) \tag{8c}$$

$$\textbf{tempo}_4 S = \forall zs \bullet zs \in S \iff (\exists i \bullet zs_{[i..(i+1)]} \in S) \tag{8d}$$

The second temporal property is about belonging to one sublist in the beginning or in the end. If a generator is in a list, then it must be at the beginning or at the ending.

The third temporal property is about belonging to one sublist in the middle. If a generator belongs to a sublist between i and j, then it belongs to the sublist that starts at first position and ends in j and to the sublist that starts at i and ends at the last position (both sublists contain the sublist in the middle).

Finally, if a generator belongs to a list, then there is a sublist of size one that contains the generator.

Variables have all four temporal properties. For a generator x, the following is valid:

$$\mathbf{tempo}_1\,(\mathbf{var}\,x) \wedge \mathbf{tempo}_2\,(\mathbf{var}\,x) \wedge \mathbf{tempo}_3\,(\mathbf{var}\,x) \wedge \mathbf{tempo}_4\,(\mathbf{var}\,x)$$

Other expressions also meet one or more temporal properties:

$$\mathbf{tempo}_1 S \wedge \mathbf{tempo}_1 T \implies \mathbf{tempo}_1\,(S \cap T) \tag{9a}$$
$$\mathbf{tempo}_3 S \wedge \mathbf{tempo}_3 T \implies \mathbf{tempo}_3\,(S \cap T) \tag{9b}$$
$$\mathbf{tempo}_2 S \wedge \mathbf{tempo}_2 T \implies \mathbf{tempo}_2\,(S \cup T) \tag{9c}$$
$$\mathbf{tempo}_4 S \wedge \mathbf{tempo}_4 T \implies \mathbf{tempo}_4\,(S \cup T) \tag{9d}$$

XBefore Laws. We now show some laws to be used in the algebraic reduction of ATF formulas. The laws follow from the definition of XBefore, from events independence, and from the temporal properties.

We define events independence ($\lhd\rhd$) as the property that one operand does not imply the other. For example, we need to avoid that the operands of XBefore are $\mathbf{var}\,a$ and $\mathbf{var}\,a \cup \mathbf{var}\,b$ (it results in $\{\}$, see (11e)).

$$S \lhd\rhd T = \forall i, zs \bullet \neg \left(zs_{[i..(i+1)]} \in S \wedge zs_{[i..(i+1)]} \in T \right) \tag{10}$$

The absence of occurrences ($\{\}$, the empty set of **atf**) is a “0” for the XBefore operator.

$$\{\} \rightarrow S = \{\} \qquad \text{left-false-absorb} \tag{11a}$$
$$S \rightarrow \{\} = \{\} \qquad \text{right-false-absorb} \tag{11b}$$
$$(S \rightarrow T) \cup S = S \qquad \text{left-union-absorb} \tag{11c}$$
$$(T \rightarrow S) \cup S = S \qquad \text{right-union-absorb} \tag{11d}$$
$$\mathbf{tempo}_1 S \implies S \rightarrow S = \{\} \qquad \text{non-idempotent} \tag{11e}$$
$$\begin{aligned} &\mathbf{tempo}_1 S \wedge \mathbf{tempo}_1 T \wedge \\ &\mathbf{tempo}_1 U \implies \\ &S \rightarrow (T \rightarrow U) = (S \rightarrow T) \rightarrow U \end{aligned} \qquad \text{associativity} \tag{11f}$$

The XBefore is absorbed by one of the operands: if one of the operands may happen alone, thus the order with any other operand is irrelevant. However, an event cannot come before itself, thus XBefore is not idempotent Finally, the XBefore is associative.

To allow formula reduction we need the relation of XBefore to the other Boolean operators. We use the XBefore as operands of union and intersection.

$$\begin{aligned} &\mathbf{tempo}_1 S \wedge \mathbf{tempo}_1 T \implies \\ &\quad (S \rightarrow T) \cap (T \rightarrow S) = \{\} \end{aligned} \qquad \text{inter-equiv-false} \tag{12a}$$
$$\begin{aligned} &\mathbf{tempo}_{1-4} S \wedge \mathbf{tempo}_{1-4} T \wedge S \lhd\rhd T \implies \\ &\quad (S \rightarrow T) \cup (T \rightarrow S) = S \cap T \end{aligned} \qquad \text{union-equiv-inter} \tag{12b}$$

As the XBefore is not symmetric, the intersection of symmetrical sets is empty. The union of the symmetric is a partition of the intersection of the operands.

There are other laws shown in [8]. We are still working on a syntactical reduction for tautology and contradiction. Such an analysis for Boolean algebra uses binary trees for formula reduction. Our initial studies show that the ATF relies on a ternary tree. Besides such a syntactical analysis, we only need those laws shown in this section.

3 The Activation Logic

The Activation Logic (AL) proposed in this work emerges from the need to analyse the behaviour of a system when a subset of the faults is active during the same time period, and to provide completeness analysis of system behaviour. There are at least two strategies to use AL to obtain structure expressions of SFT, TFT, or DFT: (i) model systems directly in AL, and (ii) obtaining operational mode expressions extracted from failure traces, as shown in the work reported in [8]. In approaches as those reported in [16,27], behavioural completeness is left for the analyst. Using tautology and the indication of undefined nominal values, we ensure that no situation is left forgotten.

The AL associates: (i) an operational mode, and (ii) the expression of fault events that *activates* the operational mode or error event. The expressions of fault events can be written in any algebra that provides tautology and contradiction properties. Thus, AL is parametrized by: (i) an algebra that provides at least tautology and contradiction, and (ii) operational modes. Figure 3 depicts an overview of AL.

We summarise the properties of the AL as follows:

1. No expression predicate is a contradiction: there are no *false* predicates in activation expressions;
2. The predicates in the terms of an expression consider all possible situations: expression tautology;
3. There are no two terms with exactly the same operational mode: all expression terms are related to a unique operational mode.

These properties form the *healthiness conditions* [14] of an expression in the AL.

We show the general form of the AL to model faults in Sect. 3.1, the healthiness conditions to normalize expressions in Sect. 3.2, how to identify non-determinism in an expression in Sect. 3.3, and the predicates notation to analyse systems and model fault propagation in Sect. 3.4.

3.1 The Activation Logic Grammar

Each term in an expression is a pair of a predicate and an operational mode. The predicate is written in either Boolean algebra, ATF, or any algebra that provides these properties: tautology and contradiction. We assume that the set

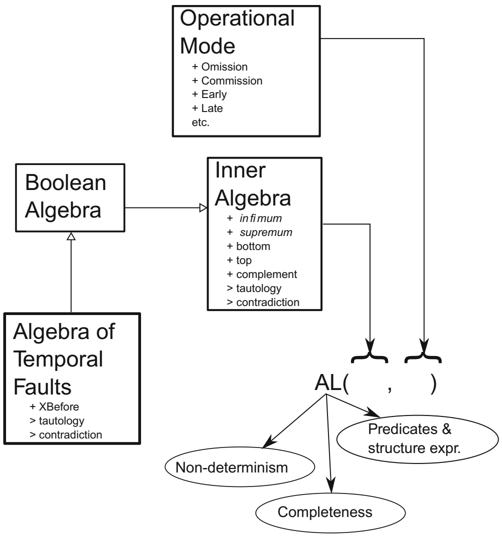

Fig. 3. Activation Logic (AL) overview

of possible faults on a system is finite and that each variable declared in a predicate represents a fault event.

The operational mode has two generic values: (i) Nominal, and (ii) Failure. Nominal values either determine value, or an undefined value (in this case, the constant value "$undefined$" is assumed). Failure values denote an erroneous behaviour, which can be a total failure (for example, signal omission) or a failure that causes degradation (for example, a signal below or above its nominal range). The choice of the operational modes depends on the system being analysed and its definition is generic and is left for the analyst. For the AL, it is sufficient to specify that it is an erroneous behaviour.

The grammar is parametrized by the syntax of an algebra (`Algebra`) and a set of operational modes (`OperModes`). The initial rules of the grammar are defined as follows:

```
AL(Algebra, OperModes)      = TERM(Algebra, OperModes)
                              | TERM(Algebra, OperModes)
                                '|' AL(Algebra, OperModes)
TERM(Algebra, OperModes)    = '(' Algebra ',' OM(OperModes) ')'
OM(OperModes)               = 'Nominal' NominalValue
                              | 'Failure' OperModes
NominalValue                = 'undefined' | Number
Number                      = Integer | Bool | Decimal
```

The denotational semantics of the expressions in AL is a set of pairs. The predicate in each term of an expression depends on the semantics of the inner algebra. Thus the predicate evaluates to either *true* ($\top$) or *false* ($\bot$) depending on the valuation in the algebra. In what follows we show a sketch of the denotational semantics of AL.

$$\begin{aligned}
(\mathtt{P}_1, \mathtt{O}_1) &\mapsto \{(P_1, O_1)\} \\
(\mathtt{P}_1, \mathtt{O}_1) \mid (\mathtt{P}_2, \mathtt{O}_2) &\mapsto \{(P_1, O_1), (P_2, O_2)\} \\
\mathtt{Nominal\ 100} &\mapsto \mathrm{Nominal}\, 100 \\
\mathtt{Nominal\ undefined} &\mapsto \mathrm{Nominal}\, undefined \\
\mathtt{Failure\ Omission} &\mapsto \mathrm{Failure}\, Omission
\end{aligned}$$

In an expression, if the ith predicate evaluates to *true* ($\top$), we say that the ith operational mode is *activated*. To simplify the presentation of the expressions and to ease the understanding, we use the denotational semantics in the remainder of this article (the right-hand side of the sketch above).

In this section, to illustrate the properties and possible analyses, we use an example of a system with faults A and B and the following outputs:

O_1**:** when A is active;
O_2**:** when B is active;
O_3**:** when A is active, but B is not;
O_4**:** when A or B are active.

The expression for this example in AL is:

$$S = \{(A, O_1), (B, O_2), (A \wedge \neg B, O_3), (A \vee B, O_4)\} \tag{13}$$

In this example we see that one of the healthiness conditions is not satisfied: when for instance, A and B are both inactive ($\neg(A \wedge B)$), there is no explicit output defined. In Sect. 4 we show a more detailed case study to illustrate the reasoning about temporal faults. In the next section, we show how to normalise the expression, so that the three healthiness conditions are satisfied.

3.2 Healthiness Conditions

The healthiness conditions are fix points of a language. The property is defined as a function of an expression and returns another expression. For example, if a healthiness condition H is satisfied for an expression Exp, thus $\mathrm{H}(Exp) = Exp$.

In what follows we show the three healthiness conditions for the AL. All definitions in this section refer to an algebra that has the following properties:

contradiction: the expression always evaluates to *false*;
tautology: the expression always evaluates to *true*.

$\mathbf{H_1}$**: No predicate is a contradiction.** This property is very simple and it is used to eliminate any term that has a predicate that always evaluates to false.

Definition 1. *Let exp be an expression in the AL, then:*

$$\mathrm{H}_1(exp) = \{(P, O) \mid (P, O) \in exp \bullet \neg \mathrm{contradiction}(P)\} \tag{14}$$

where the operator $\in$ indicates that a term is present in the expression.

Applying the first healthiness condition to our example results in:

$$\mathrm{H}_1(S) = S$$

Thus, we conclude that S *is* H_1-healthy.

$\mathbf{H_2}$: All possibilities are covered. This property is used to make explicit that there are uncovered operational modes. In this case, there is a combination of variables in the inner algebra that was not declared in the expression. Very often the focus when modelling faults is the erroneous behaviour, so we assume that such an uncovered operational mode is nominal, but has an undefined value.

Definition 2. *Let exp be an expression in the AL, and τ is:*

$$\tau = \neg\left(\bigvee_{(P,O)\in exp} P\right)$$

then:

$$\mathrm{H}_2(exp) = \begin{cases} exp, & \textit{if}\ \text{contradiction}(\tau) \\ exp \cup \{(\tau, \text{Nominal}\, undefined)\}, & \textit{otherwise} \end{cases} \tag{15}$$

This property states that if the expression is already complete, so all possibilities are already covered, thus the expression is healthy.

Applying the second healthiness condition to our example results in the following expression after simplification:

$$\mathrm{H}_2(S) = S \cup \{(\neg A \land \neg B, \text{Nominal}\, undefined)\}$$

Thus, we conclude that S *is not* H_2-healthy.

$\mathbf{H_3}$: There are no two terms with exactly the same operational mode. This property merges terms that contain the same operational mode. It avoids unnecessary formulas and may reduce the expression.

Definition 3. *Let exp be an expression in the AL. Then:*

$$\begin{aligned} \mathrm{H}_3(exp) = \{\ &(P_1, O_1) \mid (P_1, O_1) \in exp \land \\ &\forall (P_2, O_2) \in exp \bullet (P_1, O_1) = (P_2, O_2) \lor O_1 \neq O_2\ \} \cup \\ &\{(P_1 \lor P_2, O_1) \mid (P_1, O_1), (P_2, O_2) \in exp \land O_1 = O_2\} \end{aligned} \tag{16}$$

Applying H_3 in the example in the beginning of the section, we conclude that S *is* H_3-healthy. On the other hand, if we consider an S' system being a copy of S, but making $O_1 = O_2$, then:

$$\mathrm{H}_3(S') = \{(A \lor B, O_1), (A \land \neg B, O_3), (A \lor B, O_4)\}$$

Thus, we conclude that S' *is not* H_3-healthy.

Healthy Expression. To obtain a healthy expression, we apply all three healthiness conditions. The order of application of each healthiness condition does not change the resulting expression. The healthiness function is written as composition of functions as follows:

$$\mathrm{H} = \mathrm{H}_1 \circ \mathrm{H}_2 \circ \mathrm{H}_3 \tag{17}$$

After applying the three healthiness conditions to S, the resulting expression is:

$$\begin{aligned} \mathrm{H}(S) = \{\ & (A, O_1), (B, O_2), \\ & (A \wedge \neg B, O_3), (A \vee B, O_4), \\ & (\neg A \wedge \neg B, \mathrm{Nominal}\, undefined)\ \} \end{aligned}$$

The healthiness conditions are useful to faults modelling, aiding the faults analyst to check contradictions and completeness. Also, obtaining safe predicates is only possible in healthy expressions. In the next section, we show how to verify non-determinism in AL expressions.

3.3 Non-determinism

The non-determinism is usually an undesirable property. It causes an unexpected behaviour, so the analysis shall consider the activation of fault even if the fault might or not be active.

To identify a non-determinism, we can check for the negation of a contradiction in a pair of predicates in the inner algebra.

Definition 4 (Non-determinism). *Let exp be an expression in AL.*

$$\begin{aligned} \mathrm{nondeterministic}(exp) = & \exists (P_1, O_1), (P_2, O_2) \in exp \bullet \\ & \neg \mathrm{contradiction}(P_1 \wedge P_2) \end{aligned} \tag{18}$$

If there is at least one combination that evaluates $P_1 \wedge P_2$ to true (it is not a contradiction), then *exp* is non-deterministic. Our example is clearly non-deterministic as at least $A \wedge (A \vee B)$ is not a contradiction.

To analyse components and systems, and to model faults propagation, a predicates notation is shown in the next section. The predicates notation offers more two ways to check non-determinism.

3.4 Predicates Notation

The AL needs a special notation to enable the analysis of: (i) a particular faults expression, or (ii) a propagation in components. Such a special notation extracts predicates in the inner algebra given an output of interest.

Definition 5 (Predicate). *Let exp be an expression in AL, and O_x an operational mode. A predicate over exp that matches O_x is then:*

$$\langle \mathrm{out}(exp) = O_x \rangle \iff \exists (P, O) \in \mathrm{H}(exp) \mid O = O_x \bullet P \tag{19}$$

The predicate notation function returns a predicate in the inner algebra. For the example in the beginning of this section, the predicate for O_2 is obtained as follows:

$$\langle \text{out}\,(S) = O_2 \rangle = B$$

To allow fault propagation of components we need another special notation that expands the modes of an expression with a predicate in the inner algebra.

Definition 6 (Modes). *Let exp be an expression in AL, and P a predicate in the inner algebra, then:*

$$\text{modes}\,(exp, P) = \{(P_i \wedge P, O_i) \ \mid \ (P_i, O_i) \in \text{H}\,(exp)\} \tag{20}$$

Finally, to check the possible outputs, we need a function to obtain a set of outputs given an expression.

Definition 7 (Activation). *Let exp be an expression in AL, and P_x a predicate in the inner algebra, then:*

$$\text{activation}\,(exp, P_x) = \{O | (P, O) \in \text{H}\,(exp) \wedge \text{tautology}\,(P_x \implies P)\} \tag{21}$$

The non-determinism can also be checked using the predicates notation and the activation property:

$$\text{activation}\,(S, A \wedge \neg B) = \{O_1, O_3\} \tag{22a}$$
$$\langle \text{out}\,(S) = O_1 \rangle \wedge \langle \text{out}\,(S) = O_3 \rangle = \ A \wedge \neg B \tag{22b}$$

Equation (22a) shows that both O_1 and O_3 can be observed if $A \wedge \neg B$ is *true*. Equation (22b) states that if the possible operational modes of healthy S are O_1 and O_3, then the predicate is $A \wedge \neg B$. In the next section, we show a practical case study using these properties and notations.

4 Case Study

EMBRAER provided us with the Simulink model of an Actuator Control System (depicted in Fig. 4). The failure logic of this system (that is, for each of its constituent components) was also provided by EMBRAER (we show some of them in Table 2). In what follows we illustrate our strategy using the Monitor component.

A monitor component is a system commonly used for fault tolerance [15,21]. Initially, the monitor connects the main input (power source on input port 1) with its output. It observes the value of this input port and compares it to a threshold. If the value is below the threshold, the monitor disconnects the output from the main input and connects to the secondary input. We present the Simulink model for this monitor in Fig. 5.

Now we show two contributions: (i) using only Boolean operators, thus ignoring ordering, we can obtain the same results obtained in [7], and (ii) using the order-related operator reported in [8] obtaining an expression in ATF with the

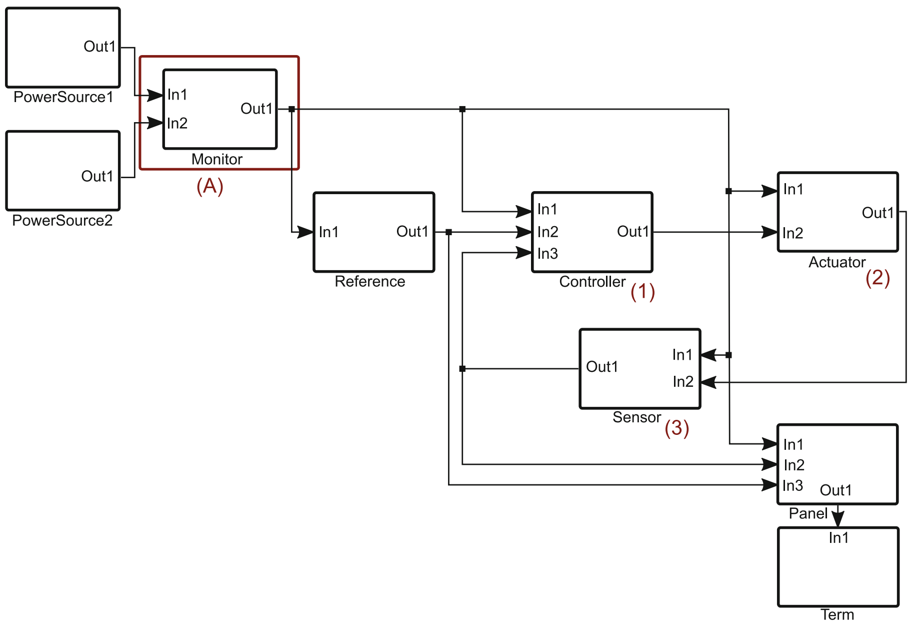

Fig. 4. Block diagram of the ACS provided by EMBRAER (nominal model)

Table 2. Annotations table of the ACS provided by EMBRAER

Component	Deviation	Port	Annotation
PowerSource	LowPower	Out1	PowerSourceFailure
Monitor	LowPower	Out1	(SwitchFailure AND (LowPower-In1 OR LowPower-In2)) OR (LowPower-In1 AND LowPower-In2)

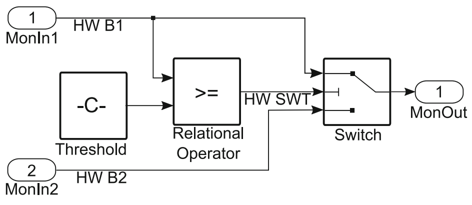

Fig. 5. Internal diagram of the monitor component (Fig. 4(A)).

same results as shown in [8]. To simplify formulas writing, we associate the fault events as:

$$B_1 = \text{LowPower-In1}$$
$$B_2 = \text{LowPower-In2}$$
$$F = \text{SwitchFailure}$$

The power source has only two possible operational modes: (i) the power source works as expected, providing a nominal value of $12V$, and (ii) is has an internal failure B_i, and its operational mode is “low power”. In AL it is modelled as:

$$PowerSource_i = \{(B_i, LP), (\neg B_i, \text{Nominal}\, 12V)\} \tag{23}$$

where LP is the LowPower failure. The expression $PowerSource_i$ is healthy.

The monitor is a bit different because its behaviour depends not only on internal faults, but also on its inputs. We will now use the predicates notation defined in Sect. 3.4 to express fault propagation. As the monitor has two inputs and its behaviour is described in Fig. 5, then it is a function of the expressions of both inputs:

$$\begin{aligned} Monitor_{bool}(in_1, in_2) = \\ & \text{modes}(in_1, \langle \text{out}(in_1) = \text{Nominal}\, X \rangle \wedge \neg F) \cup \\ & \text{modes}(in_2, \neg \langle \text{out}(in_1) = \text{Nominal}\, X \rangle \wedge \neg F) \cup \\ & \text{modes}(in_2, \langle \text{out}(in_1) = \text{Nominal}\, X \rangle \wedge F) \cup \\ & \text{modes}(in_1, \neg \langle \text{out}(in_1) = \text{Nominal}\, X \rangle \wedge F) \end{aligned} \tag{24}$$

where X is an unbound variable and assumes any value. The expression states the following:

- The monitor output is the same as in_1 if the output of in_1 *is* nominal and *there is no* internal failure in the monitor:

$$\text{modes}(in_1, \langle \text{out}(in_1) = \text{Nominal}\, X \rangle \wedge \neg F)$$

- The monitor output is the same as in_2 if the output of in_1 *is not* nominal and *there is no* internal failure in the monitor:

$$\text{modes}(in_2, \neg \langle \text{out}(in_1) = \text{Nominal}\, X \rangle \wedge \neg F)$$

- The monitor output is the converse of the previous two conditions if the internal failure F is active:

$$\begin{aligned} & \text{modes}(in_2, \langle \text{out}(in_1) = \text{Nominal}\, X \rangle \wedge F) \cup \\ & \text{modes}(in_1, \neg \langle \text{out}(in_1) = \text{Nominal}\, X \rangle \wedge F) \end{aligned}$$

The operational modes (observed behaviour) of the monitor depend on: (i) its internal fault, and (ii) propagated errors from its inputs. Composing the monitor with the two power sources, we obtain the AL expression of a power supply subsystem $System_{\text{bool}}$:

$$\begin{aligned}
&= Monitor_{bool}\left(PowerSource_1, PowerSource_2\right) \\
&= \text{modes}\left(in_1, \neg B_1 \wedge \neg F\right) \cup \text{modes}\left(in_2, \neg\neg B_1 \wedge \neg F\right) \cup \\
&\quad \text{modes}\left(in_2, \neg B_1 \wedge F\right) \cup \text{modes}\left(in_1, \neg\neg B_1 \wedge F\right) && \text{by Eq. (19)} \\
&= \text{modes}\left(in_1, \neg B_1 \wedge \neg F\right) \cup \text{modes}\left(in_2, B_1 \wedge \neg F\right) \cup \\
&\quad \text{modes}\left(in_2, \neg B_1 \wedge F\right) \cup \text{modes}\left(in_1, B_1 \wedge F\right) && \text{by simplification} \\
&= \left\{\left(P_i \wedge \neg B_1 \wedge \neg F, O_i\right) \mid \left(P_i, O_i\right) \in in_1\right\} \cup \\
&\quad \left\{\left(P_i \wedge B_1 \wedge \neg F, O_i\right) \mid \left(P_i, O_i\right) \in in_2\right\} \cup \\
&\quad \left\{\left(P_i \wedge \neg B_1 \wedge F, O_i\right) \mid \left(P_i, O_i\right) \in in_2\right\} \cup \\
&\quad \left\{\left(P_i \wedge B_1 \wedge F, O_i\right) \mid \left(P_i, O_i\right) \in in_1\right\} && \text{by Eq. (20)} \\
&= \left\{\left(B_1 \wedge \neg B_1 \wedge \neg F, LP\right),\right. \\
&\quad \left(\neg B_1 \wedge \neg B_1 \wedge \neg F, \text{Nominal } 12V\right), \\
&\quad \left(B_2 \wedge B_1 \wedge \neg F, LP\right), \\
&\quad \left(\neg B_2 \wedge B_1 \wedge \neg F, \text{Nominal } 12V\right), \\
&\quad \left(B_2 \wedge \neg B_1 \wedge F, LP\right), \\
&\quad \left(\neg B_2 \wedge \neg B_1 \wedge F, \text{Nominal } 12V\right), \\
&\quad \left(B_1 \wedge B_1 \wedge F, LP\right), \\
&\quad \left.\left(\neg B_1 \wedge B_1 \wedge F, \text{Nominal } 12V\right)\right\} && \text{replacing vars}
\end{aligned}$$

Simplifying and applying H_1, we obtain:

$$\begin{aligned}
&H_1\left(System_{\text{bool}}\right) = \\
&\qquad \left\{\left(\neg B_1 \wedge \neg F, \text{Nominal } 12V\right), \left(B_2 \wedge B_1 \wedge \neg F, LP\right),\right. \\
&\qquad \left(\neg B_2 \wedge B_1 \wedge \neg F, \text{Nominal } 12V\right), \left(B_2 \wedge \neg B_1 \wedge F, LP\right), \\
&\qquad \left.\left(\neg B_2 \wedge \neg B_1 \wedge F, \text{Nominal } 12V\right), \left(B_1 \wedge F, LP\right)\right\}
\end{aligned}$$

Applying, H_3, we simplify to:

$$\begin{aligned}
&H_3 \circ H_1\left(System_{\text{bool}}\right) \\
&\quad = \left\{\left(\begin{matrix}\left(\neg B_1 \wedge \neg F\right) \vee \\ \left(B_1 \wedge \neg B_2 \wedge \neg F\right) \vee \\ \left(\neg B_1 \wedge \neg B_2 \wedge F\right)\end{matrix}, \text{Nominal } 12V\right),\right. \\
&\qquad \left.\left(\begin{matrix}\left(B_1 \wedge B_2 \wedge \neg F\right) \vee \\ \left(\neg B_1 \wedge B_2 \wedge F\right) \vee \\ \left(B_1 \wedge F\right)\end{matrix}, LP\right)\right\} \\
&\quad = \left\{\left(\left(\neg B_1 \wedge \neg B_2\right) \vee \neg F \wedge \left(\neg B_1 \vee \neg B_2\right), \text{Nominal } 12V\right),\right. \\
&\qquad \left.\left(F \wedge \left(B_1 \vee B_2\right) \vee \left(B_1 \wedge B_2\right), LP\right)\right\}
\end{aligned}$$

The monitor expression is H_2-healthy (the predicates are complete), thus:

$$H_2 \circ H_3 \circ H_1 \left(System_{\text{bool}}\right) = H_3 \circ H_1 \left(System_{\text{bool}}\right)$$

The resulting expression for the monitor after applying all healthiness conditions is:

$$\begin{aligned} \mathrm{H}\left(System_{\text{bool}}\right) = \{&\left(\left(\neg B_1 \wedge \neg B_2\right) \vee \neg F \wedge \left(\neg B_1 \vee \neg B_2\right), \text{Nominal}\, 12V\right), \\ &\left(F \wedge \left(B_1 \vee B_2\right) \vee \left(B_1 \wedge B_2\right), LP\right)\} \end{aligned} \tag{25}$$

The operational modes of $System_{\text{bool}}$ is either Nominal $12V$ or LP (low power).

Finally, we obtain the *low power* structure expression (see Table 2) using the predicates notation:

$$\langle \text{out}\left(System_{\text{bool}}\right) = LP \rangle \iff F \wedge \left(B_1 \vee B_2\right) \vee \left(B_1 \wedge B_2\right)$$

The monitor expression also indicates that if the switch is operational ($\neg F$) and at least one PowerSource is operational ($\neg B_1 \vee \neg B_2$), the monitor output is nominal. But if at least one PowerSource is faulty ($B_1 \vee B_2$) and the monitor has an internal failure (F) the system is not operational. These two sentences written in AL using predicates notation are:

$$\begin{aligned} &\text{activation}\left(System_{\text{bool}}, \neg F \wedge \left(\neg B_1 \vee \neg B_2\right)\right) \\ &\quad = \{O \mid (P, O) \in \mathrm{H}\left(System_{\text{bool}}\right) \wedge \\ &\qquad \text{tautology}\left(\neg F \wedge \left(\neg B_1 \vee \neg B_2\right) \implies P\right)\} && \text{[by Eq. (21)]} \\ &\quad = \{\text{Nominal}\, 12V\} && \text{[by simplification]} \quad (26a) \\ &\text{activation}\left(System_{\text{bool}}, F \wedge \left(B_1 \vee B_2\right)\right) \\ &\quad = \{O \mid (P, O) \in \mathrm{H}\left(System_{\text{bool}}\right) \wedge \\ &\qquad \text{tautology}\left(F \wedge \left(B_1 \vee B_2\right) \implies P\right)\} && \text{[by Eq. (21)]} \\ &\quad = \{LP\} && \text{[by simplification]} \quad (26b) \end{aligned}$$

Now, let's consider the same system but with a subtle modification. As shown in [8], the order of the occurrence of faults may be relevant, and the qualitative and quantitative analyses results may be different than those results without considering the order of the occurrence of faults. Observing Fig. 5, we see that if F activates before a failure in the first input of the monitor, then it would display a nominal behaviour, because the internal failure F anticipates switching to the second input. On the other hand, if the first input fails before F, then the monitor would switch to the second input, then switch back, due to the internal failure. We obtain the following expression for the monitor, now using the ATF:

$$\begin{aligned} &Monitor_{ATF}\left(in_1, in_2\right) = \\ &\qquad \text{modes}\left(in_1, \langle \text{out}\left(in_1\right) = \text{Nominal}\, X \rangle \wedge \neg F\right) \cup \\ &\qquad \text{modes}\left(in_2, \neg \langle \text{out}\left(in_1\right) = \text{Nominal}\, X \rangle \wedge \neg F\right) \cup \\ &\qquad \text{modes}\left(in_2, \langle \text{out}\left(in_1\right) = \text{Nominal}\, X \rangle \wedge F\right) \cup \\ &\qquad \text{modes}\left(in_1, \neg \langle \text{out}\left(in_1\right) = \text{Nominal}\, X \rangle \rightarrow F\right) \cup \\ &\qquad \text{modes}\left(in_2, F \rightarrow \neg \langle \text{out}\left(in_1\right) = \text{Nominal}\, X \rangle\right) \end{aligned} \tag{27}$$

where X is an unbound variable and assumes any value.

The difference to $System_{\text{bool}}$ (Eq. (24)) is only the finer analysis of the cases of erroneous behaviour of the first input and an internal failure. Note that the finer analysis splits the predicate

$$\neg \langle \text{out}\,(in_1) = \text{Nominal}\,12V\rangle \wedge F \qquad\qquad (\text{activates } in_1)$$

into:

$$\neg \langle \text{out}\,(in_1) = \text{Nominal}\,12V\rangle \rightarrow F \qquad\qquad (\text{activates } in_1)$$

and

$$F \rightarrow \neg \langle \text{out}\,(in_1) = \text{Nominal}\,12V\rangle \qquad\qquad (\text{activates } in_2)$$

We can assure that such a split is complete because the predicate notation evaluates to B_1. Thus the operands satisfy all temporal properties (Eqs. (8a) to (8d)) and events independence (Eq. (10)), thus the law shown in Eq. (12b) is valid. For case (i), the expected behaviour is the same as in_1 because the system switches to in_2, but then an internal failure occurs, and it switches back to in_1. For case (ii), it switches to in_2 due to an internal failure, then the first input fails, so the behaviour is similar to the nominal behaviour (see the second *modes* in Eq. (27)).

Following the similar expansions of Eq. (24), we obtain:

$$\begin{aligned}
System_{ATF} =& Monitor_{ATF}\,(PowerSource_1, PowerSource_2)\\
=& \{(B_1 \wedge \neg B_1 \wedge \neg F, LP)\,,\\
& (\neg B_1 \wedge \neg B_1 \wedge \neg F, \text{Nominal}\,12V)\,,\\
& (B_2 \wedge B_1 \wedge \neg F, LP)\,,\\
& (\neg B_2 \wedge B_1 \wedge \neg F, \text{Nominal}\,12V)\,,\\
& (B_2 \wedge \neg B_1 \wedge F, LP)\,,\\
& (\neg B_2 \wedge \neg B_1 \wedge F, \text{Nominal}\,12V)\,,\\
& (B_1 \wedge B_1 \rightarrow F, LP)\,,\\
& (\neg B_1 \wedge B_1 \rightarrow F, \text{Nominal}\,12V)\}\,,\\
& (B_2 \wedge F \rightarrow B_1, LP)\,,\\
& (\neg B_2 \wedge F \rightarrow B_1, \text{Nominal}\,12V)\}
\end{aligned}$$

Simplifying and applying H_1 to remove contradictions, we obtain:

$$\begin{aligned}
&\text{H}_1\,(System_{ATF}) =\\
&\quad \{(\neg B_1 \wedge \neg F, \text{Nominal}\,12V)\,, (B_2 \wedge B_1 \wedge \neg F, LP)\,,\\
&\quad (\neg B_2 \wedge B_1 \wedge \neg F, \text{Nominal}\,12V)\,, (B_2 \wedge \neg B_1 \wedge F, LP)\,,\\
&\quad (\neg B_2 \wedge \neg B_1 \wedge F, \text{Nominal}\,12V)\,, (B_1 \rightarrow F, LP)\,,\\
&\quad (B_2 \wedge F \rightarrow B_1, LP)\,, (\neg B_2 \wedge F \rightarrow B_1, \text{Nominal}\,12V)\}
\end{aligned}$$

Applying H_3 to remove redundant terms with identical operational modes and using the rules shown in Sect. 2.3, we simplify to:

$$\begin{aligned}
&\mathrm{H}_3 \circ \mathrm{H}_1 \left(System_{\mathrm{ATF}}\right) \\
&= \left\{ \left(\begin{matrix} (\neg B_1 \wedge \neg F) \vee \\ (B_1 \wedge \neg B_2 \wedge \neg F) \vee \\ (\neg B_1 \wedge \neg B_2 \wedge F) \vee \\ (\neg B_2 \wedge F \rightarrow B_1) \end{matrix}, \mathrm{Nominal}\, 12V \right), \right. \\
&\quad \left. \left(\begin{matrix} (B_1 \wedge B_2 \wedge \neg F) \vee \\ (\neg B_1 \wedge B_2 \wedge F) \vee \\ (B_1 \rightarrow F) \vee \\ (B_2 \wedge F \rightarrow B_1) \end{matrix}, LP \right) \right\} \\
&= \{((\neg B_1 \wedge \neg B_2) \vee \neg F \wedge (\neg B_1 \vee \neg B_2) \vee \\
&\quad \neg B_2 \wedge F \rightarrow B_1, \mathrm{Nominal}\, 12V), \\
&\quad ((B_1 \wedge B_2) \vee (\neg B_1 \wedge B_2 \wedge F) \vee (\neg B_2 \wedge B_1 \rightarrow F), LP)\}
\end{aligned}$$

The monitor expression is H_2-healthy. Simplifying Boolean operators as usual, the XBefore expression:

$$\neg B_2 \wedge F \rightarrow B_1 \vee \neg B_2 \wedge B_1 \rightarrow F$$

simplifies to

$$\neg B_2 \wedge F \wedge B_1 \qquad \text{by Eq. (12b)}$$

Thus:

$$\mathrm{H}_2 \circ \mathrm{H}_3 \circ \mathrm{H}_1 \left(System_{\mathrm{ATF}}\right) = \mathrm{H}_3 \circ \mathrm{H}_1 \left(System_{\mathrm{ATF}}\right)$$

The resulting expression for the monitor after applying all healthiness conditions is:

$$\begin{aligned}
\mathrm{H}\left(System_{ATF}\right) = \{&((\neg B_1 \wedge \neg B_2) \vee \neg F \wedge (\neg B_1 \vee \neg B_2) \vee \\
&\neg B_2 \wedge F \rightarrow B_1, \mathrm{Nominal}\, 12V), \\
&((B_1 \wedge B_2) \vee (\neg B_1 \wedge B_2 \wedge F) \vee \\
&(\neg B_2 \wedge B_1 \rightarrow F), LP)\}
\end{aligned} \tag{28}$$

Finally, we obtain the *low power* structure expression of the monitor using the predicates notation:

$$\langle \mathrm{out}\left(System_{\mathrm{ATF}}\right) = LP \rangle \iff (B_1 \wedge B_2) \vee (\neg B_1 \wedge B_2 \wedge F) \vee (\neg B_2 \wedge B_1 \rightarrow F)$$

Thus, $System_{\mathrm{ATF}}$ fails with LP if:

- Both power sources fail;
- The monitor fails to detect the nominal state of the first power source and the second power source is in a failure state;

– The monitor fails to detect the failure state of the first power source (the monitor fails after the failure of the first power source).

Note that if the monitor fails before the failure of the first power source, it fails to detect the operational mode of the first power source and switches to the second power source, which is in a nominal state (see expression $\neg B_2 \wedge F \rightarrow B_1$ in Eq. (28)).

5 Conclusion

In this work we proposed a parametrized logic that enables the analysis of systems depending on the expressiveness of a given algebra and a given set of operational modes. If ATF is used as a parameter, then the order of occurrence of faults can be considered. Although the logic is not as detailed as AADL, the predicates notation in conjunction with the ATF provides a richer assertion framework. Also, it is possible to verify non-determinism on the model, by: (i) verifying its existence with the nondeterministic function, (ii) providing an expression and obtaining the possible operational modes with the activation function, or (iii) using the predicates notation to obtain a predicate that enables two or more operational modes.

The AADL is extensible. The work reported in [2] shows an extension to perform dependability analysis through state machines and expressions on fault events and operational modes. Although such an extension captures system behaviour, operational mode activation conditions are expressed in state transitions in combination with an extension of Boolean expressions (not related to order). Our work relates operational modes and fault occurrences order explicitly.

As presented in [8], TFTs and DFTs structure expressions can be written as formulas in ATF. As the root events of TFTs and DFTs represent operational modes of a system, the ATF can be used to associate root events with operational modes, thus allowing the combination of two or more fault trees.

Although the properties of AL require that the inner algebra provides tautology and contradiction, and we used ATF in the case study, we did not show tautology and contradiction for ATF. Instead, we used a law to reduce the ATF expression to a Boolean expression. The methodology to check tautology and contradiction in ATF is a future work.

The original expression shown in the case study was already H_2-healthy. The second healthiness condition about completeness uses the concept of undefined value to make any expression H_2-healthy. Algebraically it is fine, but in practice, the property should be met originally, thus the initial expression is already H_2-healthy. This property should be used as an alert to the analyst if it not met originally.

Acknowledgements. We would like to thank **Alexander Romanovsky**, **Zoe Andrews** and **Richard Payne** for all discussions about fault modelling and dependability. This work was funded by CNPq, grants 476821/2011-8, 442859/2014-7, and 246956/2012-7, and by FACEPE grant IBPG-0408-1.03/11. This work was partially supported by the National Institute of Science and Technology for Software Engineering (INES, http://www.ines.org.br), funded by CNPq and FACEPE, grants 573964/2008-4 and APQ-1037-1.03/08.

References

1. SAE ARP4761 Guidelines and Methods for Conducting the Safety Assessment Process on Civil Airborne Systems and Equipment, December 1996
2. SAE Architecture Analysis and Design Language (AADL) Annex Volume 1: Annex A: ARINC653 Annex, Annex C: Code Generation Annex, Annex E: Error Model Annex. Technical report, SAE International (2015)
3. Akers, S.B.: Binary decision diagrams. IEEE Trans. Comput. **C–27**(6), 509–516 (1978)
4. ANAC. Aeronautical Product Certification. DOU No. 230, Seção 1, p. 28, 01 December 2011, (2011)
5. Andrews, J.D.: The use of not logic in fault tree analysis. Qual. Reliab. Eng. Int. **17**(3), 143–150 (2001)
6. Boute, R.T.: The binary decision machine as programmable controller. Euromicro Newslett. **2**(1), 16–22 (1976)
7. Didier, A.L.R., Mota, A.: Identifying hardware failures systematically. In: Gheyi, R., Naumann, D. (eds.) Formal Methods: Foundations and Applications. Lecture Notes in Computer Science, vol. 7498, pp. 115–130. Springer, Heidelberg (2012)
8. Didier, A.L.R., Mota, A.: An algebra of temporal faults. Inf. Syst. Front. **18**, 967–980 (2016)
9. Dugan, J.B., Bavuso, S.J., Boyd, M.A.: Dynamic fault-tree models for fault-tolerant computer systems. IEEE Trans. Reliab. **41**(3), 363–377 (1992)
10. FAA. RTCA, Inc., Document RTCA/DO-178B. U.S. Dept. of Transportation, Federal Aviation Administration, Washington, D.C. (1993)
11. FAA. Part 25 - Airworthiness Standards: Transport Category Airplanes. report, Federal Aviation Administration (FAA), USA (2007)
12. Feiler, P.H., Gluch, D.P., Hudak, J.J.: The Architecture Analysis & Design Language (AADL): An Introduction. CMU/SEI–2006–TN–011, February 2006
13. Givant, S., Halmos, P.: Introduction to Boolean Algebras. Undergraduate Texts in Mathematics, vol. XIV. Springer, New York (2009)
14. Hoare, C.A.R., He, J.: Unifying Theories of Programming, vol. 14. Prentice Hall, Englewood Cliffs (1998)
15. Koren, I., Krishna, C.M.: Fault Tolerant Systems. Morgan Kaufmann Publishers Inc., San Francisco (2007)
16. Merle, G.: Algebraic modelling of Dynamic Fault Trees, contribution to qualitative and quantitative analysis. Theses, École normale supérieure de Cachan - ENS Cachan (2010)
17. Merle, G., Roussel, J.-M., Lesage, J.-J.: Algebraic determination of the structure function of Dynamic Fault Trees. Reliab. Eng. Syst. Saf. **96**(2), 267–277 (2011)
18. Merle, G., Roussel, J.-M., Lesage, J.-J.: Dynamic fault tree analysis based on the structure function. In: 2011 Proceedings - Annual Reliability and Maintainability Symposium, January 2011

19. Merle, G., Roussel, J.-M., Lesage, J.-J.: Quantitative analysis of dynamic fault trees based on the structure function. Qual. Reliab. Eng. Int. **30**(1), 143–156 (2014)
20. Merle, G., Roussel, J.-M., Lesage, J.-J., Bobbio, A.: Probabilistic algebraic analysis of fault trees with priority dynamic gates and repeated events. IEEE Trans. Reliab. **59**(1), 250–261 (2010)
21. O'Connor, P.D.T., Newton, D., Bromley, R.: Practical Reliability Engineering. Wiley, Hoboken (2002)
22. Oliva, S.: Non-coherent fault trees can be misleading. e-J. Syst. Saf. **42**(3), 1–5 (2006)
23. Tannous, O., Xing, L., Dugan, J.B.: Reliability analysis of warm standby systems using sequential BDD. In: 2011 Proceedings - Annual Reliability and Maintainability Symposium, January 2011
24. Vesely, W., Goldberg, F.F., Roberts, N.H., Haasl, D.F.: Fault Tree Handbook. Number NUREG-0492. US Independent Agencies and Commissions (1981)
25. Walker, M.D.: Pandora: a logic for the qualitative analysis of temporal fault trees. Ph.D. thesis, University of Hull (2009)
26. Walker, M.D., Papadopoulos, Y.: Synthesis and analysis of temporal fault trees with PANDORA: the time of Priority AND gates. Nonlinear Anal. Hybrid Syst. **2**(2), 368–382 (2008)
27. Walker, M.D., Papadopoulos, Y.: Qualitative temporal analysis: towards a full implementation of the fault tree handbook. Control Eng. Pract. **17**(10), 1115–1125 (2009)
28. Walker, M.D., Papadopoulos, Y.: A hierarchical method for the reduction of temporal expressions in Pandora. In: Proceedings of the First Workshop on DYnamic Aspects in DEpendability Models for Fault-Tolerant Systems, DYADEM-FTS 2010, pp. 7–12. ACM, New York (2010)
29. Xing, L., Tannous, O., Dugan, J.B.: Reliability analysis of nonrepairable cold-standby systems using sequential binary decision diagrams. IEEE Trans. Syst. Man Cybern. A **42**(3), 715–726 (2012)