Dynamic Fault Trees Analysis using an Integration of Theorem Proving and Model Checking

Yassmeen Elderhalli[1], Osman Hasan[1,2], Waqar Ahmad[2]
and Sofiène Tahar[1]

[1]Department of Electrical and Computer Engineering,
Concordia University, Montréal, Canada
{y_elderh,o_hasan,tahar}@ece.concordia.ca

[2]Electrical Engineering and Computer Science,
National University of Science and Technology, Islamabad, Pakistan
waqar.ahmad@seecs.nust.edu.pk

TECHNICAL REPORT

December 2017

Abstract

Dynamic fault trees (DFTs) have emerged as an important tool for capturing the dynamic behavior of system failure. These DFTs are then analyzed qualitatively and quantitatively using stochastic or algebraic methods to judge the failure characteristics of the given system in terms of the failures of its subcomponents. Model checking has been recently proposed to conduct the failure analysis of systems using DFTs with the motivation to provide a rigorous failure analysis of safety-critical systems. However, model checking has not been used for the DFT qualitative analysis and the reduction algorithms used in model checking are usually not formally verified. Moreover, the analysis time grows exponentially with the increase of the number of states. These issues limit the usefulness of model checking for analyzing complex systems used in safety-critical domains, where the accuracy and completeness of analysis matters the most. To overcome these limitations, we propose a comprehensive methodology to perform the qualitative and quantitative analysis of DFTs using an integration of theorem proving and model checking based approaches. For this purpose, we formalized all the basic dynamic fault tree gates using higher-order logic based on the algebraic approach and formally verified some of the simplification properties. This formalization allows us to formally verify the equivalence between the original and reduced DFTs using a theorem prover, and conduct the qualitative analysis. We then use model checking to perform the quantitative analysis of the formally verified reduced DFT. We applied our methodology to five benchmarks and the results show that the formally verified reduced DFT was analyzed using model checking with up to six times less states and up to 133000 times faster.

Keywords— Dynamic Fault Trees, Theorem Proving, Model Checking, HOL4, STORM

Contents

1 Introduction

A fault tree (FT) [1] is a graphical representation of the causes of failure of a system that is usually represented as the top event of the fault tree. FTs can be categorized as Static Fault trees (SFT) and Dynamic Fault trees (DFT) [1]. In SFT, the structure function (expression) of the top event describes the failure relationship between the basic events of the tree using FT gates, like AND and OR, without considering the sequence of failure of these events. DFTs, on the other hand, model the failure behavior of the system using dynamic FT gates, like the spare gate, which can capture the dependent behavior of the basic events along with the static gates. DFTs provide a more realistic representation of systems using the dynamic gates. For example, the spare DFT gate can model the failure of the car tires and their spares that cannot be modeled using the SFT gates.

Fault Tree Analysis (FTA) [1] has become an essential part of the safety-critical system design process, where the causes of failure and their probabilities should be considered at an early stage. There are two main phases for FTA, the qualitative analysis and the quantitative analysis [2]. In the *qualitative analysis*, the cut sets and cut sequences are determined, which, respectively, represent combinations and sequences of basic events of the DFT that cause a system failure [1]. The *quantitative analysis* provides numeric analysis results about the probability of failure of the top event and the mean time to failure among other metrics [1]. Dynamic FTA is commonly done algebraically [3] and using Markov chains [2]. In the algebraic approach, an algebra similar to the Boolean algebra is used to determine the structure function of the top event. Based on this algebra, the structure function can be reduced to determine a reduced form of the cut sets and sequences. The probabilistic analysis of the FT can then be performed based on the reduced form of the generated structure function by considering the probability of failure of the basic events. For the Markov chain based analysis, the FT is first converted to its equivalent Markov chain and then the probability of failure of the top event is determined by analyzing the generated Markov chain. The resultant Markov chain can be very large, while dealing with complex systems, which limits the usage of Markov chains in DFT analysis.

Traditionally, the dynamic FTA is performed using paper-and-pencil proof methods or computer simulation. While the former is error prone, specially for large systems, the latter provides a more scalable alternative. However, the results of simulation cannot be termed as accurate due to the involvement of several approximations in the underlying computation algorithms and the sampling based nature of this method. Given the dire need of accuracy in failure analysis of safety-critical systems, formal methods [1] have also been recently explored for DFT analysis. For example, the STORM probabilistic model checker [4] has been used to analyze DFTs based on Markov chain analysis [5]. Similarly, higher-order logic (HOL) theorem proving has been used to formalize and analyze SFTs [6]. However, probabilistic model checking has not been used in the formal qualitative analysis of DFTs. Moreover, it cannot support the analysis of large systems unless a reduction algorithm is invoked, and the implementation of such reduction is usually not formally verified. This means that one cannot ascertain that the analysis results after reduction are accurate or correspond to the original system.

On the other hand, the only support for FTs in HOL is limited to SFTs.

We propose to overcome the above-mentioned limitations of formal DFT analysis by using an integrated model checking and theorem proving based methodology. We propose to use theorem proving for verifying the equivalence between the original and the reduced form of the DFT. The formally verified reduced DFT can then by quantitatively analyzed using model checking. Thus, the proposed methodology tends to provide a more sound analysis than the sole model checking based analysis due to the involvement of a theorem prover in the verification of the reduced model. Moreover, it caters for the state-space based issues of model checking by providing it a reduced model for the quantitative analysis. The foremost components of the proposed methodology include the formalization of the dynamic gates and their formally verified reduction theorems, which in turn are used to verify the equivalence between the original DFTs and the reduced ones. Using this verified reduced DFT, a reduced form of the cut sets and sequences of the structure function of the DFT can be formally verified within a theorem prover. We then perform the quantitative analysis of the formally verified reduced DFT in model checking and thus reduce the generated state space and the analysis time. More importantly, we are confident that the analysis results of the reduced DFT correspond to the original DFT, as the reduction is verified using theorem proving. In order to illustrate the utilization and effectiveness of the proposed methodology, we analyzed five benchmark DFTs, i.e., a Hypothetical Example Computer System (HECS) [2], a Hypothetical Cardiac Assist System (HCAS) [3,7], a scaled cascaded PAND DFT [7, 8], a multiprocessor computing system [7, 9] and a variant of the Active Heat Rejection System (AHRS) [10].

The reduced DFTs and their reduced cut sequences are formally verified using HOL4 theorem prover. In addition, each DFT is analyzed twice using STORM model checker, one without any reduction and the other using the reduced DFTs. The analysis results show that using the verified reduced DFT for the quantitative analysis allows us to reduce the number of generated states by the model checker and the time required to perform the analysis.

The rest of the report is structured as follows: Section 2 presents some related work. Section 3 provides a detailed description of the proposed methodology. In Section 4, we present our HOL formalization of DFT gates. In Section 5, we provide the details of the verification of the simplification theorems. Section 6 describes a set of experimental results. Finally, we conclude the report in Section 7.

2 Related Work

DFT analysis has been conducted using various tools and techniques [1]. For example, Markov chains have been extensively used for the modeling and analysis of DFTs [2]. The scalability of Markov chains in analyzing large DFTs is achieved by using a modularization approach [11], where the DFT is divided into two parts: static and dynamic. The static subtree is analyzed using the ordinary SFT analysis methods, such as Binary Decision Diagrams (BDD) [1], and the dynamic subtree is analyzed using Markov chains. This kind of modularization approach is available in the Galileo tool [12]. In [7],

the authors use a compositional aggregation technique to develop Input-Output Interactive Markov Chains (I/O-IMC) to analyse DFTs. This approach is implemented in the DFTCalc tool [13]. The algebraic approach has also been extensively used in the analysis of DFTs [3], where the top event of the DFT can be expressed and reduced in a manner similar to the ordinary Boolean algebra. The reliability of the system expressed algebraically can be evaluated based on the algebraic expression of the top event [8]. The main problem with the Markov chain analysis is the large generated state space when analyzing complex systems, which requires high resources in terms of memory and time. Moreover, simulation is usually utilized in the analysis process, which does not provide accurate results. Although modularization tends to overcome the large state-space problem with Markov chains, we cannot obtain a verified reduced form of the cut sequences of the DFT. The algebraic approach provides an algebraic framework for performing both the reduction and the analysis of the DFT. However, the foundations of this approach have not been formalized, which implies that the results of the analysis should not be relied upon especially in safety-critical systems.

Formal methods can overcome the above-mentioned inaccuracy limitations of traditional DFT analysis techniques. Probabilistic model checkers, such as STORM [4], have been used for the analysis of DFTs. The main idea behind this approach is to automatically convert the DFT of a given system into its corresponding Markovian model and then analyze the safety characteristics quantitatively of the given system using the model checker [14]. The STORM model checker accepts the DFT to be analyzed in the Galileo format [12] and generates a failure automata of the tree. This approach allows us to verify failure properties, like probability of failure, in an automatic manner. However, the approach suffers from scalability issues due to the inherent state-space explosion problem of model checking. Moreover, the implementation of the reduction algorithms used in model checkers are generally not formally verified. Finally, model checkers have only been used in the context of probabilistic analysis of DFTs and not for the qualitative analysis, as the cut sequences in the qualitative analysis cannot be provided unless the state machine is traversed to the fail state, which is difficult to achieve for large state machines.

Exploiting the expressiveness of higher-order logic (HOL) and the soundness of theorem proving, Ahmad et.al [6, 15] formalized static fault trees in HOL4 and evaluated the probability of failure based on the Probabilistic Inclusion-Exclusion principle. However, the main problem in theorem proving lies in the fact that it is interactive, i.e., it needs user guidance in the proof process. Moreover, to the best of our knowledge, no higher-order-logic formalization of DFTs is available in the literature so far and thus it is not a straightforward task to conduct the DFT analysis using a theorem prover as of now.

It can be noted that both model checking and HOL theorem proving exhibit complementary characteristics, i.e., model checking is automatic but cannot deal with large systems and does not provide qualitative analysis of DFTs, while HOL theorem proving allows us to verify universally quantified generic mathematical expressions but at the cost of user interventions. In this work, we leverage upon the complementary nature of these approaches to present an integrated methodology that provides the expressiveness of higher-order logic and the existing support for automated probabilistic analysis of

DFTs using model checking. The main idea is to use theorem proving to formally verify the equivalence between the original and the reduced DFT and then use a probabilistic model checker to conduct quantitative analysis on the reduced DFT. As a result, both the generated state machine and the analysis time are reduced. In addition, a formally verified reduced form of the cut sequences is also obtained.

3 Proposed Methodology

The proposed methodology for the formal DFT analysis is depicted in Figure 1. It provides both formal DFT qualitative analysis using theorem proving and quantitative analysis using model checking. The DFT analysis starts by having a system description. The failure behavior of this system is then modeled as a DFT, which can be reduced based on the algebraic approach [3]. The idea of this algebraic approach is to deal with the events, which can represent the basic events or outputs, according to their time of failure (d). For example, $d(X)$ represents the time of failure of an event X. In the algebraic approach, temporal operators (Simultaneous (Δ), Before ($\lhd$) and Inclusive Before ($\trianglelefteq$)) are defined to model the dynamic gates. Based on these temporal operators, several simplification theorems exist to perform the required reduction. This reduction process can be erroneous if it is performed manually using paper-and-pencil. Moreover, reduction algorithms may also provide wrong results if they are not formally verified. In order to formally check the equivalence between the original model and the reduced one, we developed a library of formalized dynamic gates in HOL and verified their corresponding simplification theorems. These foundations allow us to develop a formal model for any DFT using the formal gate definitions. Based on the verified simplification theorems, we can then verify the equivalence between the formally specified original and the reduced DFT models using a theorem prover. The formally verified reduced structure function can then be utilized to perform the qualitative analysis of the reduced model in the theorem prover as well as its quantitative analysis by using a model checker.

The qualitative analysis represents an important and a crucial step in DFT analysis, since it allows to identify the sources of failure of the system without the availability of any information or actual numbers about the failure probabilities of the basic events. In static fault trees, the qualitative analysis is performed by finding the cut sets. Due to the temporal behavior of the dynamic gates, just finding the cut sets does not capture the sequence of failure of events that can cause the system failure. The cut sequences on the other hand capture not only the combination of basic events but also the sequence of events that can cause the system failure. In the proposed methodology, a theorem prover is used to verify a reduced expression of the structure function of the top event, which ensures that the reduction process is accurate. Using this reduced structure function, a formally verified reduced form of the cut sequences can also be determined.

The formally verified cut sets and sequences for the DFT and a reduced form of the structure function of the top event can now be used in a probabilistic model checker to do the quantitative analysis of the given system. Because of the reduced model, we

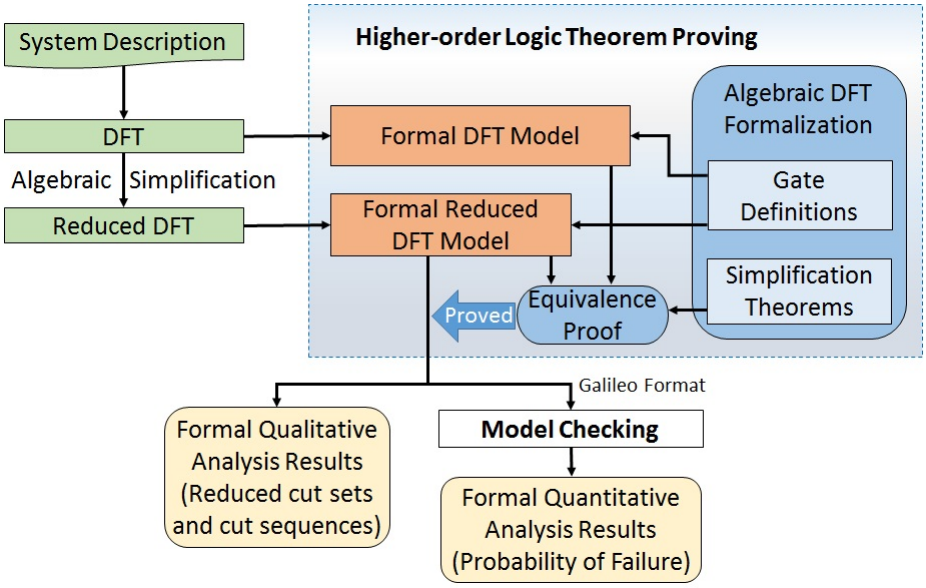


Figure 1: Overview of Proposed Methodology

get a reduction in the analysis time and number of states. In this work, the STORM model checker is used to perform the probabilistic analysis of the DFT. Several input languages are supported by this model checker including the Galileo format for DFT. Both the probability of failure of the top event as well as the mean time to failure can be computed using STORM. It is worth mentioning that since the analyzed model of the DFT is a Markov Automata (MA) (in case of non-deterministic behavior) or a Continuous Time Markov Chain (CTMC), only exponential failure distributions are supported by the proposed methodology.

4 Formalization of Dynamic Fault Trees in HOL

In this section, we present the formal definitions in HOL of the identity elements, the temporal operators and the dynamic gates. It is assumed that a fault is represented using an event. The occurrence of a fault indicates that the corresponding event is true. It is also assumed that the events are non-repairable.

4.1 Identity Elements

Two identity elements are defined, these are the *ALWAYS* and the *NEVER* elements. The *ALWAYS* identity element represents an event with a time of failure equals to 0. The *NEVER* element represents an event that never occurs. These two elements are defined based on their time of failure in HOL as follows:

Definition 1. *ALWAYS element*

```
⊢ ALWAYS = (0:extreal)
```

Definition 2. *NEVER element*

```
⊢ NEVER = PosInf
```

where `extreal` is the HOL data type for extended real numbers, which includes positive infinity ($+\infty$) and negative infinity ($-\infty$) and `PosInf` is the ($+\infty$) representation in HOL.

4.2 Temporal Operators

We formalize three temporal operators to model the dynamic behavior of the DFT: *Simultaneous* (Δ), *Before* ($\lhd$) and *Inclusive Before* ($\trianglelefteq$). The *Simultaneous* operator has two input events, which represent basic events or subtrees. The time of occurrence (failure) of the output event of this operator is equal to the time of occurrence of the first or the second input event considering that both input events occur at the same time:

$$d(A \Delta B) = \begin{cases} d(A), & d(A) = d(B) \\ +\infty, & d(A) \neq d(B) \end{cases} \tag{1}$$

It is assumed that for any two basic events, if the failure distribution of the random variables that represent these basic events is continuous then they cannot have the same time of failure, and hence the result of the *Simultaneous* operator between them is *NEVER*.

$$d(A \Delta B) = NEVER \tag{2}$$

where A and B are basic events with random variables that exhibit continuous failure distributions.

The *Before* operator accepts two input events, which can be basic events or two subtrees. The time of occurrence of the output event of this operator is equal to the time of occurrence of the first input event if the first input event (left) occurs before the second input event (right), otherwise the output never fails:

$$d(A \lhd B) = \begin{cases} d(A), & d(A) < d(B) \\ +\infty, & d(A) \geq d(B) \end{cases} \tag{3}$$

The *Inclusive Before* combines the behavior of both the *Simultaneous* and *Before* operators, i.e., if the first input event (left) occurs before or at the same time as the second input event (right), then the output event occurs with a time of occurrence equal to the time of occurrence of the first input event:

$$d(A \trianglelefteq B) = \begin{cases} d(A), & d(A) \leq d(B) \\ +\infty, & d(A) > d(B) \end{cases} \tag{4}$$

We formalize these temporal operators in HOL as follows:

Definition 3. *Simultaneous Operator*

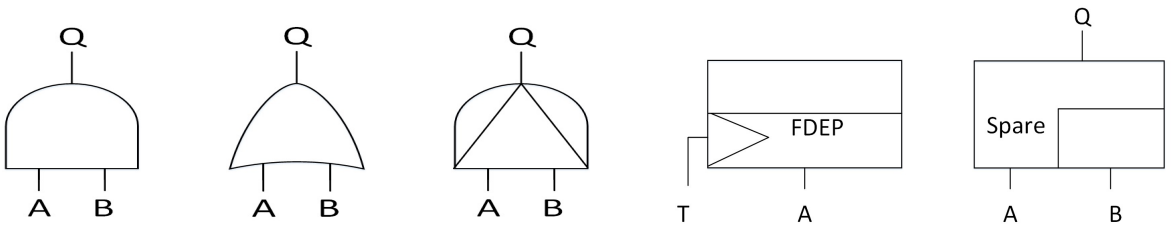
```
⊢ ∀ (A:extreal) B. D_SIMULT A B = if (A = B) then A else PosInf
```

Definition 4. *Before Operator*

```
⊢ ∀ (A:extreal) B. D_BEFORE A B = if (A < B) then A else PosInf
```

Definition 5. *Inclusive Before Operator*

```
⊢ ∀ (A:extreal) B. D_INCLUSIVE_BEFORE A B = if (A ≤ B) then A else PosInf
```



(a) AND (b) OR (c) PAND (d) FDEP (e) Spare

Figure 2: Fault Tree Gates

where A and B represent the time of failure of the events A and B, respectively.

4.3 Fault Tree Gates

Fig. 2 shows the main FT gates [2]; dynamic gates as well as the static ones.

Although, the AND ($\cdot$) and OR ($+$) gates are considered as static operators or gates, their behavior can be represented using the time of occurrence of the input events. For example, the output event of an AND gate occurs if and only if all its input events occur. This implies that the output of the AND gate occurs with the occurrence of the last input event, which means that the time of occurrence of the output event equals the maximum time of occurrence of the input events. The OR gate is defined in a similar manner with the only difference that the output event occurs with the occurrence of the first input event, i.e., the minimum time of occurrence of the inputs:

$$d(A \cdot B) = max(d(A)\ ,\ d(B)) \tag{5}$$

$$d(A + B) = min(d(A)\ ,\ d(B)) \tag{6}$$

We model the behavior of these gates in HOL as follows:

Definition 6. *AND gate (operator)*
```
⊢ ∀ (A:extreal) B. D_AND A B = max A B
```

Definition 7. *OR gate (operator)*
```
⊢ ∀ (A:extreal) B. D_OR A B = min A B
```

where `max` and `min` are functions that return the maximum and the minimum values of their arguments, respectively.

The Priority-AND (PAND) gate is a special case of the AND gate, where the output occurs when all the input events occur in sequence, conventionally from left to right. For the PAND gate, shown in Figure 2c, the output Q occurs if A and B occur and A occurs before or with B. The behavior of the PAND gate can be represented using the time of failure as:

$$d(Q) = \begin{cases} d(B), & d(A) \leq d(B) \\ +\infty, & d(A) > d(B) \end{cases} \tag{7}$$

The behavior of the PAND gate can be expressed using the temporal operators as:

$$Q = B \cdot (A \trianglelefteq B) \tag{8}$$

We define the PAND in HOL as:

Definition 8. *PAND gate*
```
⊢ ∀ (A:extreal) B. PAND A B = if (A ≤ B ) then B else PosInf
```

We verify in HOL that the PAND exhibits the behavior given in Equation 8:

Theorem 1. `⊢ ∀ A B. PAND A B = D_AND B (D_INCLUSIVE_BEFORE A B)`

The Functional Dependency (FDEP) gate , shown in Figure 2d, is used when there is a failure dependency between the input events or sub-trees, i.e., the occurrence of one input (or a sub-tree) can trigger the occurrence of other input events (or subtrees) in the fault tree. For example, in Figure 2d, the occurrence of T triggers the occurrence of A. This implies that A occurs in two different ways: firstly, when A occurs by itself, and secondly, when the trigger T occurs. This means that the time of failure of A_T (triggered A) equals the minimum time of occurrences of T and A:

$$d(A_T) = min(\ d(A), d(T)) \tag{9}$$

We define the FDEP in HOL as:

Definition 9. *FDEP gate*
```
⊢ ∀ (A:extreal) T. FDEP A T = min A T
```

where T is the occurrence time of the trigger. We also verify in HOL that the FDEP is equivalent to an OR gate as follows:

Theorem 2. `⊢ ∀ A T. FDEP A T = D_OR A T`

The spare gate, shown in Figure 2e, represents a dynamic behavior that occurs in many real world systems, where we usually have a main part and some spare parts. The spare parts are utilized when the main part fails. The spare gate, shown in Figure 2e, has a main input (A) and a spare input (B). After the failure of A, B is activated. The output of the spare gate fails if both the main input and the spare fail. The spare gate can have several spare inputs, and the output fails after the failure of the main input and all the spares. The spare gate has three variants depending on the failure behavior of the spare part: the hot spare gate (HSP), the cold spare gate (CSP) and the warm spare gate (WSP). In the HSP, the probability of failure for the spare is the same in both the dormant and the active states. For the CSP, the spare part cannot fail unless it is activated. The WSP is the general case, where the spare part can fail in the dormant state as well as in the active state, but the failure distribution of the spare in its dormant state is different from the one in the active mode, and it is usually attenuated by a dormancy factor. In order to be able to distinguish between the different states of the spare input, two different variables are assigned to each state. For example, for the spare gate, shown in Figure 2e, B is represented using two variables; B_a for the active state and B_d for the dormant state.

The input events of the spare gate cannot occur at the same time if they are basic events. However, if these events are subtrees then they can occur at the same time.

For a two input warm spare gate, with A as the primary input and B as the spare input, the output event occurs in two ways; firstly, if A fails first then B, i.e., the spare part, is activated and then B fails in its active state. The second way is when B fails in its dormant state (inactive) then A fails with no spare to replace it. For the general case where the input events can occur at the same time (if they are subtrees or depend on a common trigger), an additional option for the failure of the spare gate is added, where the two input events occur at the same time. This general warm spare gate can be described as:

$$Q = A.(B_d \lhd A) + B_a.(A \lhd B_a) + A \Delta B_a + A \Delta B_d \tag{10}$$

We formalize the WSP in HOL as:

Definition 10. *Warm Spare Gate*
```
⊢ ∀ A B_a B_d.  WSP A B_a B_d = D_OR (D_OR (D_OR (D_AND A(D_BEFORE B_d A))
(D_AND B_a(D_BEFORE A B_a)))(D_SIMULT A B_a))(D_SIMULT A B_d)
```

The time of failure of the CSP gate with primary input A and cold spare B can be defined as:

$$d(Q) = \begin{cases} d(B), & d(A) < d(B) \\ +\infty, & d(A) \geq d(B) \end{cases} \tag{11}$$

which means that the output event of the CSP occurs if the primary input fails and then the spare fails while in its active state. We define the CSP in HOL as:

Definition 11. *Cold Spare Gate*
```
⊢ ∀ (A:extreal) B. CSP A B = if (A < B) then B else PosInf
```

We verify in HOL that the CSP gate is a special case of WSP, where the spare part cannot fail in its dormant state.

Theorem 3. `⊢ ∀ A B_a B_d.`
`ALL_DISTINCT [A; B_a] ∧ COLD_SPARE B_d ⇒ (WSP A B_a B_d = CSP A B_a)`

where `ALL_DISTINCT` ensures that A and B_a are not equal, which means that they cannot fail at the same time, and `COLD_SPARE B_d` indicates that the spare B is a cold spare, i.e., it cannot fail in its dormant mode (B_d).

The spare part in the HSP has only one failure distribution, i.e., the dormant state and the active state are the same. The output of the HSP fails when both the primary and the spare fail, and the sequence of failure does not matter, as the spare part has only one failure distribution. The HSP is defined as:

$$d(Q) = max(d(A)\ ,\ d(B)) \tag{12}$$

where A is the primary input and B is the spare. We define this in HOL as:

Definition 12. *Hot Spare Gate*
```
⊢ ∀ (A:extreal) B. HSP A B = max A B
```

We verify in HOL that if both the dormant and the active states of the spare are equal, then the WSP is equivalent to the HSP:

Theorem 4. `⊢ ∀ A B_a B_d.  (B_a = B_d) ⇒ (WSP A B_a B_d = HSP A B_a)`

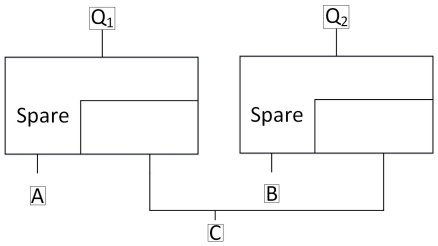


Figure 3: Spare Gates with Shared Spare

It is important to mention that more than one spare gate can share the same spare input. In this case, there is a possibility that one of the primary inputs is replaced by the spare, while the other input does not have a spare in case it fails. The outputs of the spare gates, shown in Figure 3, are represented as follows (assuming that A, B and C are basic events):

$$Q_1 = A.(C_d \lhd A) + C_a.(A \lhd C_a) + A.(B \lhd A) \tag{13}$$

$$Q_2 = B.(C_d \lhd B) + C_a.(B \lhd C_a) + B.(A \lhd B) \tag{14}$$

The last term in Q_1 indicates that if B occurs before A, then the spare part C is used by the second spare gate. This means that no spare is available for the first spare gate, which causes the failure of the output of the first spare gate if A occurs. We formalize the output Q_1 of the first spare gate in HOL as:

Definition 13. *Shared Spare*

```
⊢ ∀ A B C_a C_d.  shared_spare A B C_a C_d = D_OR (D_OR (D_AND A(D_BEFORE C_d A))(D_AND
C_a (D_BEFORE A C_a)))D_AND A(D_BEFORE B A))
```

We define a function in HOL called `Never_events`, which ensures that its operands are mutual exclusive, i.e., only one of them can occur. We formalize it in HOL as:

Definition 14. *Never events*

```
⊢ ∀ A B. NEVER_events A B = (D_AND A B = NEVER)
```

This function is useful when we want to make sure that two events cannot happen together. For example, for a CSP gate, the spare part can only fail in one of its two states and not in both.

5 Formal Verification of the Simplification Theorems

As with classical Boolean algebra, many simplification theorems also exist for DFT operators, which can be used to simplify the structure function of the DFT. We formally verified over 80 simplification theorems for the operators, defined in the previous section, including commutativity, associativity and idempotence of the AND, OR and Simultaneous operators, in addition to more complex theorems that include a combination of all operators. The verification process of these theorems was mainly based

on the properties of extended real numbers, since the DFT operators are defined based on the time of failure of the events, which we choose to model using the `extreal` data type in HOL. During the verification process, each theorem was divided into several sub-goals based on the definition of the operators. Most of these sub-goals were automatically verified using automated tactics that utilize theorems from the `extreal` HOL theory. These simplification theorems can be classified into four groups depending on the operators involved in the simplification.

5.1 Simplification Theorems using OR and AND

These simplification theorems are similar to the OR and AND related Boolean algebra theorems, such as commutativity and associativity. Based on the theorems presented in [3], Table 1 lists the formalization for these theorems, which we proved in HOL.

Table 1: Simplification Theorems for OR and AND

HOL Theorems	DFT Algebra Theorems
⊢ ∀ A B. D_OR A B = D_OR B A	$A + B = B + A$
⊢ ∀ A B. D_AND A B = D_AND B A	$A.B = B.A$
⊢ ∀ A B C. D_OR A (D_OR B C) = D_OR (D_OR A B)C	$A + (B + C) = (A + B) + C$
⊢ ∀ A B C. D_AND A (D_AND B C) = D_AND (D_AND A B) C	$A.(B.C) = (A.B).C$
⊢ ∀ A . D_OR A A = A	$A + A = A$
⊢ ∀ A . D_AND A A = A	$A.A = A$
⊢ ∀ A B C . D_AND A (D_OR B C) = D_OR (D_AND A B)(D_AND B C)	$A.(B + C) = A.B + A.C$
⊢ ∀ A. D_OR A NEVER = A	$A + NEVER = A$
⊢ ∀ A. D_AND A ALWAYS = A	$A.ALAWYS = A$
⊢ ∀ A. D_OR A ALWAYS = ALWAYS	$A + ALAWYS = ALWAYS$
⊢ ∀ A. D_AND A NEVER = NEVER	$A.NEVER = NEVER$
⊢ ∀ A B C. D_OR A (D_AND B C) = D_AND (D_OR A B)(D_OR A C)	$A + (B.C) = (A + B).(A + C)$
⊢ ∀ A B. D_OR A (D_AND A B) = A	$A + (A.B) = A$
⊢ ∀ A B. D_AND A (D_OR A B) = A	$A.(A + B) = A$

5.2 Simplification Theorems using Before Operator

As with the AND and OR, several simplification theorems were introduced in [3] to simplify expressions that include the Before operator. Our formalization of these theorems in HOL is given in Table 2.

Table 2: Simplification Theorems for Before Operator

HOL Theorems	DFT Algebra Theorems
⊢ ∀ A B. D_AND (D_BEFORE A B) D_BEFORE B A) = NEVER	$(A \lhd B).(B \lhd A) = NEVER$
⊢ ∀ A B C. D_BEFORE A (D_BEFORE B C) = D_OR (D_BEFORE A B)(D_AND (D_AND A B) (D_OR (D_BEFORE C B)(D_SIMULT C B)))	$A \lhd (B \lhd C) = (A \lhd B) + (A.B.$ $((C \lhd B) + (C \Delta B)))$
⊢ ∀ A B C. D_BEFORE A (D_BEFORE B C) = D_OR (D_BEFORE A B)(D_AND (D_AND A B) (D_INCLUSIVE_BEFORE C B))	$A \lhd (B \lhd C) = (A \lhd B) + (A.B.$ $((C \trianglelefteq B)))$
⊢ ∀ A B C. D_BEFORE (D_BEFORE A B) C = D_AND (D_BEFORE A B)(D_BEFORE A C)	$(A \lhd B) \lhd C = (A \lhd B).(A \lhd C)$
⊢ ∀ A. D_BEFORE NEVER A = NEVER	$NEVER \lhd A = NEVER$
⊢ ∀ A. D_BEFORE A NEVER = A	$A \lhd NEVER = A$
⊢ ∀ A. D_BEFORE A A = NEVER	$A \lhd A = NEVER$
⊢ ∀ A B C. D_BEFORE A (D_OR B C) = D_AND (D_BEFORE A B)(D_BEFORE A C)	$A \lhd (B + C) = (A \lhd B).(A \lhd C)$
⊢ ∀ A B C. D_BEFORE A (D_AND B C) = D_OR (D_BEFORE A B)(D_BEFORE A C)	$A \lhd (B.C) = (A \lhd B) + (A \lhd C)$
⊢ ∀ A B C. D_BEFORE A (D_SIMULT B C) = D_OR (D_OR (D_OR (D_AND A (D_BEFORE B C)) (D_AND A (D_BEFORE C B))) (D_BEFORE A B))(D_BEFORE A C)	$A \lhd (B \Delta C) = (A.(B \lhd C)) +$ $(A.(C \lhd B)) + (A \lhd B) +$ $(A \lhd C)$
⊢ ∀ A B C. D_BEFORE A (D_INCLUSIVE_BEFORE B C) = D_OR (D_BEFORE A B)(D_AND (D_AND A B)(D_BEFORE C B))	$A \lhd (B \trianglelefteq C) = (A \lhd B) +$ $(A.B.(C \lhd B))$
⊢ ∀ A B C. D_BEFORE (D_OR A B) C = D_OR (D_BEFORE A C)(D_BEFORE B C)	$(A + B) \lhd C = (A \lhd C) +$ $(B \lhd C)$
⊢ ∀ A B C. D_BEFORE (D_AND A B) C = D_AND (D_BEFORE A C)(D_BEFORE B C)	$(A.B) \lhd C = (A \lhd C).(B \lhd C)$

HOL Theorems	DFT Algebra Theorems
⊢ ∀ A B C. D_BEFORE (D_SIMULT A B) C = D_AND (D_SIMULT A B)(D_BEFORE A C)	$(A\Delta B) \lhd C = (A\Delta B).(A \lhd C)$
⊢ ∀ A B C. D_BEFORE (D_SIMULT A B) C = D_AND (D_SIMULT A B)(D_BEFORE B C)	$(A\Delta B) \lhd C = (A\Delta B).(B \lhd C)$
⊢ ∀ A B C. D_BEFORE (D_SIMULT A B) C = D_SIMULT (D_BEFORE A C)(D_BEFORE B C)	$(A\Delta B) \lhd C = (A \lhd C)\Delta$ $(B \lhd C)$
⊢ ∀ A B C. D_BEFORE (D_INCLUSIVE_BEFORE A B) C = D_AND (D_INCLUSIVE_BEFORE A B) (D_BEFORE A C)	$(A \trianglelefteq B) \lhd C = (A \trianglelefteq B).(A \lhd C)$
⊢ ∀ A B. D_OR A(D_BEFORE A B) = A	$A + (A \lhd B) = A$
⊢ ∀ A B. D_OR (D_BEFORE A B)B = D_OR A B	$(A \lhd B) + B = A + B$
⊢ ∀ A B. D_AND A(D_BEFORE A B) = D_BEFORE A B	$A.(A \lhd B) = A \lhd B$
⊢ ∀ A B C. D_AND (D_AND (D_BEFORE A B) (D_BEFORE B C))(D_BEFORE A C) = D_AND (D_BEFORE A B)(D_BEFORE B C)	$(A \lhd B).(B \lhd C).(A \lhd C) =$ $(A \lhd B).(B \lhd C)$

5.3 Simplification Theorems using Simultaneous Operator

Table 3 shows the simplification theorems which can be used with the Simultaneous operator along with their formalizations in HOL.

Table 3: Simplification Theorems for Simultaneous Operator

HOL Theorems	DFT Algebra Theorems
⊢ ∀ A B. D_SIMULT A B = D_SIMULT B A	$A\Delta B = B\Delta A$
⊢ ∀ A B C. D_SIMULT A (D_SIMULT B C) = D_SIMULT (D_SIMULT A B) C	$A\Delta(B\Delta C) = (A\Delta B)\Delta C$
⊢ ∀ A B C. D_SIMULT A (D_SIMULT B C) = D_AND (D_SIMULT A B)(D_SIMULT B C)	$A\Delta(B\Delta C) = (A\Delta B).(B\Delta C)$

HOL Theorems	DFT Algebra Theorems
⊢ ∀ A B C. D_SIMULT A (D_SIMULT B C) = D_AND (D_SIMULT A C)(D_SIMULT C B)	$A\Delta(B\Delta C) = (A\Delta C).(C\Delta B)$
⊢ ∀ A. D_SIMULT A NEVER = NEVER	$A\Delta NEVER = NEVER$
⊢ ∀ A. D_SIMULT A A = A	$A\Delta A = A$
⊢ ∀ A B C. D_SIMULT A (D_OR B C) = D_OR (D_OR (D_AND (D_SIMULT A B)(D_SIMULT B C)) (D_AND (D_SIMULT A B)(D_BEFORE B C))) (D_AND (D_SIMULT A C)(D_BEFORE C B))	$A\Delta(B + C) = (A\Delta B).(B\Delta C)$ $+(A\Delta B).(B \lhd C)$ $+(A\Delta C).(C \lhd B)$
⊢ ∀ A B C. D_SIMULT A (D_OR B C) = D_OR (D_AND (D_SIMULT A B) (D_INCLUSIVE_BEFORE B C))(D_AND (D_SIMULT A C)(D_INCLUSIVE_BEFORE C B))	$A\Delta(B + C) = (A\Delta B).(B \trianglelefteq C) +$ $(A\Delta C).(C \trianglelefteq B)$
⊢ ∀ A B C. D_SIMULT A (D_AND B C) = D_OR (D_OR (D_AND (D_SIMULT A B) (D_SIMULT B C))(D_AND (D_SIMULT A B) (D_BEFORE C B)))(D_AND (D_SIMULT A C)(D_BEFORE B C))	$A\Delta(B.C) = (A\Delta B).(B\Delta C)$ $+(A\Delta B).(C \lhd B)$ $+(A\Delta C).(B \lhd C)$
⊢ ∀ A B C. D_SIMULT A (D_AND B C) = D_OR (D_AND (D_SIMULT A B) (D_INCLUSIVE_BEFORE C B)) (D_AND (D_SIMULT A C) (D_INCLUSIVE_BEFORE B C))	$A\Delta(B.C) = (A\Delta B).(C \trianglelefteq B)$ $+(A\Delta C).(B \trianglelefteq C)$
⊢ ∀ A B C. D_SIMULT A (D_BEFORE B C) = D_AND (D_SIMULT A B)(D_BEFORE B C)	$A\Delta(B \lhd C) = (A\Delta B).(B \lhd C)$
⊢ ∀ A B C. D_SIMULT A (D_INCLUSIVE_BEFORE B C) = D_AND (D_SIMULT A B) (D_INCLUSIVE_BEFORE B C)	$A\Delta(B \trianglelefteq C) = (A\Delta B).(B \trianglelefteq C)$
⊢ ∀ A B. D_OR A (D_SIMULT A B) = A	$A + (A\Delta B) = A$
⊢ ∀ A B. D_AND A (D_SIMULT A B) = D_SIMULT A B	$A.(A\Delta B) = A\Delta B$

HOL Theorems	DFT Algebra Theorems
⊢ ∀ A B C. D_AND (D_AND (D_SIMULT A B) (D_SIMULT B C))(D_SIMULT A C) = D_AND (D_SIMULT A B)(D_SIMULT B C)	$(A\Delta B).(B\Delta C).(A\Delta C) = (A\Delta B).(B\Delta C)$

5.4 Simplification Theorems using Inclusive Before Operator

Table 4 shows the HOL verified formalization of the theorems that can be used with the Inclusive Before operator.

Table 4: Simplification Theorems for Inclusive Before Operator

HOL Theorems	DFT Algebra Theorems
⊢ ∀ A B. D_AND (D_INCLUSIVE_BEFORE A B) (D_INCLUSIVE_BEFORE B A) = D_SIMULT A B	$(A \trianglelefteq B).(B \trianglelefteq A) = A\Delta B$
⊢ ∀ A B C. D_INCLUSIVE_BEFORE A (D_INCLUSIVE_BEFORE B C) = D_OR (D_OR (D_BEFORE A B) (D_AND (D_AND A B)(D_BEFORE C B))) (D_AND (D_SIMULT A B) (D_INCLUSIVE_BEFORE B C))	$A \trianglelefteq (B \trianglelefteq C) = (A \triangleleft B) + (A.B.(C \triangleleft B)) + (A\Delta B).(B \trianglelefteq C)$
⊢ ∀ A B C. D_INCLUSIVE_BEFORE (D_INCLUSIVE_BEFORE A B) C = D_AND (D_INCLUSIVE_BEFORE A B) (D_INCLUSIVE_BEFORE A C)	$(A \trianglelefteq B) \trianglelefteq C = (A \trianglelefteq B).(A \trianglelefteq C)$
⊢ ∀ A. D_INCLUSIVE_BEFORE NEVER A = NEVER	$NEVER \trianglelefteq A = NEVER$
⊢ ∀ A. D_INCLUSIVE_BEFORE A NEVER = A	$A \trianglelefteq NEVER = A$
⊢ ∀ A. D_INCLUSIVE_BEFORE A A = A	$A \trianglelefteq A = A$
⊢ ∀ A B C. D_INCLUSIVE_BEFORE A (D_OR B C) = D_AND (D_INCLUSIVE_BEFORE A B) (D_INCLUSIVE_BEFORE A C)	$A \trianglelefteq (B + C) = (A \trianglelefteq B).(A \trianglelefteq C)$

HOL Theorems	DFT Algebra Theorems
⊢ ∀ A B C. D_INCLUSIVE_BEFORE A (D_AND B C) = D_OR (D_INCLUSIVE_BEFORE A B) (D_INCLUSIVE_BEFORE A C)	$A \trianglelefteq (B.C) = (A \trianglelefteq B) + (A \trianglelefteq C)$
⊢ ∀ A B C. D_INCLUSIVE_BEFORE A (D_BEFORE B C) = D_OR(D_OR(D_BEFORE A B)(D_AND (D_AND A B) (D_INCLUSIVE_BEFORE C B))) (D_AND (D_SIMULT A B)(D_BEFORE B C))	$A \trianglelefteq (B \lhd C) = (A \lhd B)$ $+(A.B.(C \trianglelefteq B)$ $+(A \Delta B).(B \lhd C)$
⊢ ∀ A B C. D_INCLUSIVE_BEFORE A (D_SIMULT B C) = D_OR(D_OR(D_OR(D_OR(D_AND A (D_BEFORE B C))(D_AND A (D_BEFORE C B)))(D_BEFORE A B)) (D_BEFORE A C))(D_AND (D_SIMULT A B)(D_SIMULT B C))	$A \trianglelefteq (B \Delta C) = (A.(B \lhd C))$ $+(A.(C \lhd B))$ $+(A \lhd B) + (A \lhd C)$ $+(A \Delta B).(B \Delta C)$
⊢ ∀ A B C. D_INCLUSIVE_BEFORE (D_OR A B) C = D_OR (D_INCLUSIVE_BEFORE A C) (D_INCLUSIVE_BEFORE B C)	$(A + B) \trianglelefteq C = (A \trianglelefteq C)$ $+(B \trianglelefteq C)$
⊢ ∀ A B C. D_INCLUSIVE_BEFORE (D_AND A B) C= D_AND (D_INCLUSIVE_BEFORE A C) (D_INCLUSIVE_BEFORE B C)	$(A.B) \trianglelefteq C = (A \trianglelefteq C).(B \trianglelefteq C)$
⊢ ∀ A B C. D_INCLUSIVE_BEFORE (D_SIMULT A B) C = D_AND (D_SIMULT A B) (D_INCLUSIVE_BEFORE A C)	$(A \Delta B) \trianglelefteq C = (A \Delta B).(A \trianglelefteq C)$
⊢ ∀ A B C. D_INCLUSIVE_BEFORE (D_SIMULT A B) C = D_AND (D_SIMULT A B) (D_INCLUSIVE_BEFORE B C)	$(A \Delta B) \trianglelefteq C = (A \Delta B).(B \trianglelefteq C)$
⊢ ∀ A B C. D_INCLUSIVE_BEFORE (D_SIMULT A B) C = D_SIMULT (D_INCLUSIVE_BEFORE A C) (D_INCLUSIVE_BEFORE B C)	$(A \Delta B) \trianglelefteq C = (A \trianglelefteq C)$ $\Delta(B \trianglelefteq C)$

HOL Theorems	DFT Algebra Theorems
⊢ ∀ A B C. D_INCLUSIVE_BEFORE (D_BEFORE A B) C = D_AND (D_BEFORE A B) (D_INCLUSIVE_BEFORE A C)	$(A \lhd B) \trianglelefteq C = (A \lhd B).(A \trianglelefteq C)$
⊢ ∀ A B. D_OR A (D_INCLUSIVE_BEFORE A B) = A	$A + (A \trianglelefteq B) = A$
⊢ ∀ A B. D_OR B (D_INCLUSIVE_BEFORE A B) = D_OR A B	$B + (A \trianglelefteq B) = A + B$
⊢ ∀ A B. D_AND A (D_INCLUSIVE_BEFORE A B) = D_INCLUSIVE_BEFORE A B	$A.(A \trianglelefteq B) = A \trianglelefteq B$
⊢ ∀ A B. D_OR (D_INCLUSIVE_BEFORE A B) (D_INCLUSIVE_BEFORE B A) = D_OR A B	$(A \trianglelefteq B) + (B \trianglelefteq A) = A + B$
⊢ ∀ A B. D_OR (D_AND A (D_INCLUSIVE_BEFORE A B)) (D_AND B (D_INCLUSIVE_BEFORE B A)) = D_AND A B	$(A.(B \trianglelefteq A)) + (B.(A \trianglelefteq B)) = A.B$
⊢ ∀ A B. D_OR (D_INCLUSIVE_BEFORE A B) (D_AND A (D_INCLUSIVE_BEFORE B A)) = A	$(A \trianglelefteq B) + (A.(B \trianglelefteq B)) = A$
⊢ ∀ A B C. D_AND (D_AND (D_INCLUSIVE_BEFORE A B) (D_INCLUSIVE_BEFORE B C)) (D_INCLUSIVE_BEFORE A C)) = D_AND (D_INCLUSIVE_BEFORE A B) (D_INCLUSIVE_BEFORE B C)	$(A \trianglelefteq B).(B \trianglelefteq C).(A \trianglelefteq C) = (A \trianglelefteq B).(B \trianglelefteq C)$

5.5 Simplification Theorems for Combinations of Operators

Table 5 shows our formalization in HOL for some simplification theorems from [3], which can be used to simplify expressions involving combinations of operators.

Table 5: Simplification Theorems for Combinations of Operators

HOL Theorems	DFT Algebra Theorems
⊢ ∀ A B. D_OR (D_INCLUSIVE_BEFORE A B) (D_BEFORE A B) = D_INCLUSIVE_BEFORE A B	$(A \trianglelefteq B) + (A \lhd B) = A \trianglelefteq B$

HOL Theorems	DFT Algebra Theorems
⊢ ∀ A B. D_OR (D_INCLUSIVE_BEFORE A B) (D_SIMULT A B) = D_INCLUSIVE_BEFORE A B	$(A \trianglelefteq B) + (A \Delta B) = A \trianglelefteq B$
⊢ ∀ A B. D_AND (D_BEFORE A B)(D_SIMULT A B) = NEVER	$(A \triangleleft B).(A \Delta B) = NEVER$
⊢ ∀ A B C. D_AND (D_BEFORE A B)(D_SIMULT B C)= D_AND (D_BEFORE A C)(D_SIMULT B C)	$(A \triangleleft B).(B \Delta C) =$ $(A \triangleleft C).(B \Delta C)$
⊢ ∀ A B. D_AND (D_INCLUSIVE_BEFORE A B) (D_BEFORE A B)) = D_BEFORE A B	$(A \trianglelefteq B).(A \triangleleft B) = A \triangleleft B$
⊢ ∀ A B. D_AND (D_BEFORE A B) (D_INCLUSIVE_BEFORE B A) = NEVER	$(A \triangleleft B).(B \trianglelefteq A) = NEVER$
⊢ ∀ A B. D_AND (D_INCLUSIVE_BEFORE A B) (D_SIMULT A B) = D_SIMULT A B	$(A \trianglelefteq B).(A \Delta B) = A \Delta B$
⊢ ∀ A B. D_OR (D_OR (D_BEFORE A B) (D_SIMULT A B))(D_BEFORE B A) = D_OR A B	$(A \triangleleft B) + (A \Delta B) + (B \triangleleft A)$ $= A + B$
⊢ ∀ A B. D_OR (D_OR(D_AND A (D_BEFORE B A)) (D_SIMULT A B))(D_AND B (D_BEFORE A B) = D_AND A B	$(A.(B \triangleleft A)) + (A \Delta B) +$ $(B.(A \triangleleft B)) = A.B$
⊢ ∀ A B. D_OR (D_OR (D_BEFORE A B)(D_SIMULT A B)) (D_AND A (D_BEFORE B A)) = A	$(A \triangleleft B) + (A \Delta B)$ $+(A.(B \triangleleft A)) = A$
⊢ ∀ A B C. D_AND (D_AND (D_BEFORE A B) (D_BEFORE B C))(D_INCLUSIVE_BEFORE A C) = D_AND (D_BEFORE A B)(D_BEFORE B C)	$(A \triangleleft B).(B \triangleleft C).(A \trianglelefteq C) =$ $(A \triangleleft B).(B \triangleleft C)$

6 Experimental Results

In order to illustrate the effectiveness of the proposed methodology, we utilize it to conduct the formal DFT analysis of five benchmarks. The first benchmark, depicted in Figure 4, is a scaled version of the original cascaded PAND fault tree [7,8] with repeated events. In this work, we consider a scaled version of this DFT, i.e., two similar DFTs

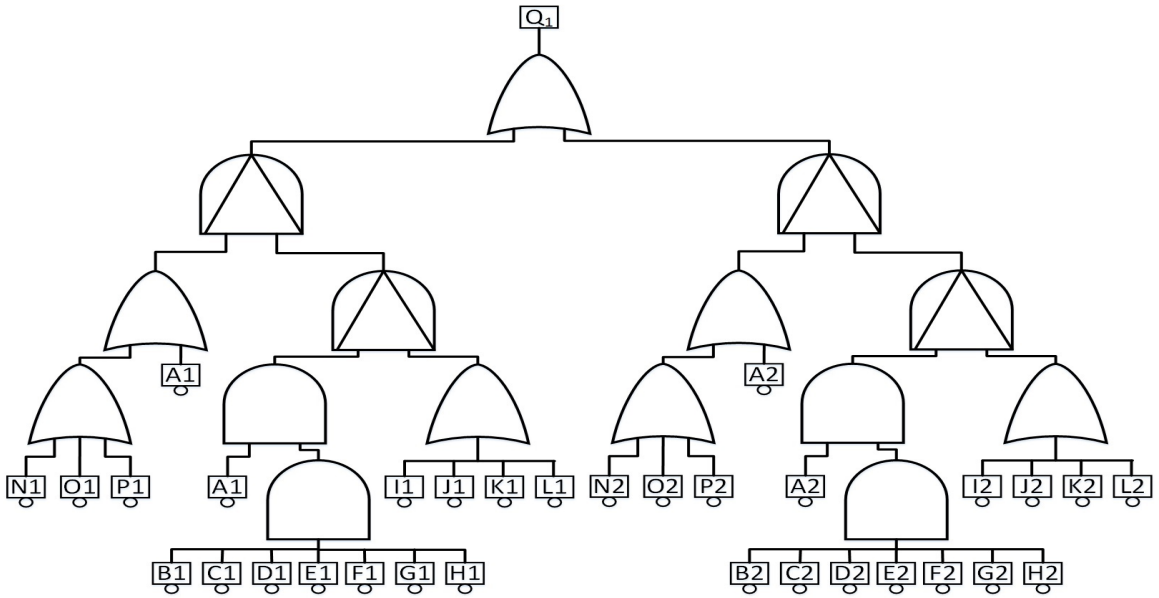


Figure 4: Scaled Cascaded PAND DFT

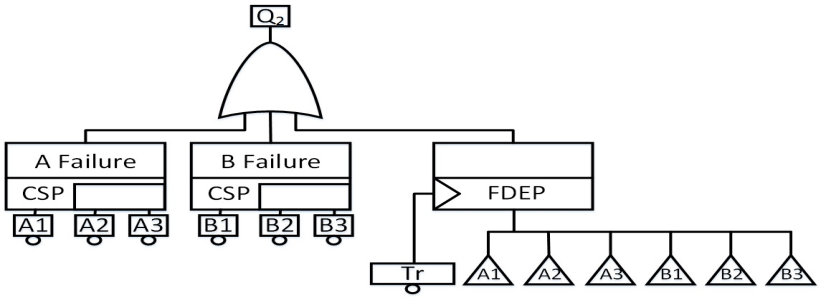


Figure 5: AHRS DFT

with different basic events and a top event that fails whenever one of these DFTs fails.

The second DFT is a modified and abstracted version of the Active Heat Rejection System (AHRS) [10], which consists of two thermal rejection units A and B. The failure of any of these two units leads to the failure of the whole system. Each main input (A_1 or B_1) has two spare parts, and the unit will fail with the failure of the main input and the spare inputs. All the inputs are functionally dependent on the power supply.

The third benchmark represents a Multiprocessor Computer System (MCS) [7, 9] with two redundant computers, having a processor, a disk and a memory unit. Each disk has its own spare and the two memory units share the same spare. The two processors are functionally dependent on the power supply.

The fourth benchmark is a Hypotheical Example Computer System (HECS) [2] consisting of two processors with a cold spare, five memory units, which are functionally dependent on two memory interface units and two system buses. The failure of the system also depends on the application subsystem, which in turn depends on the software, the hardware and the human operator.

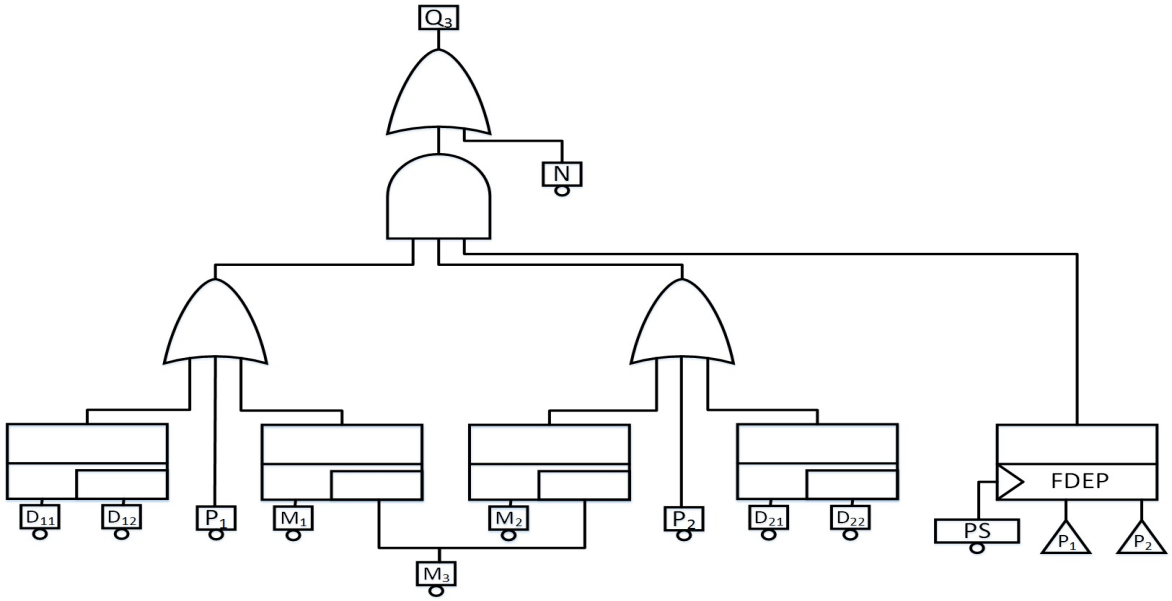


Figure 6: MCS DFT

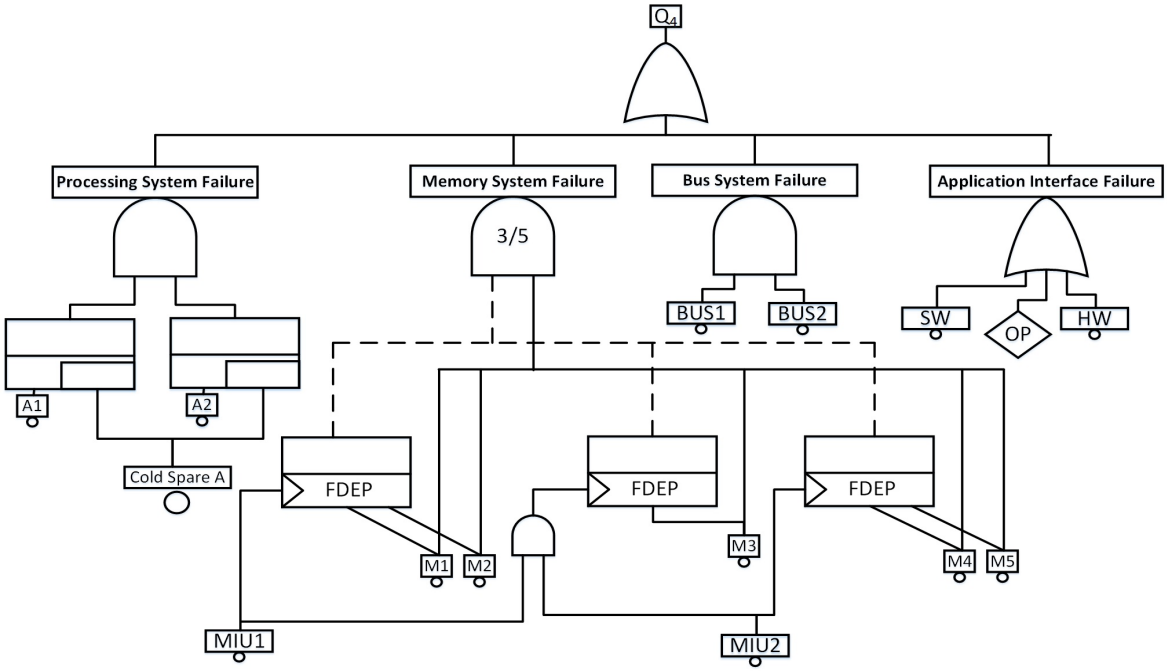


Figure 7: HECS DFT

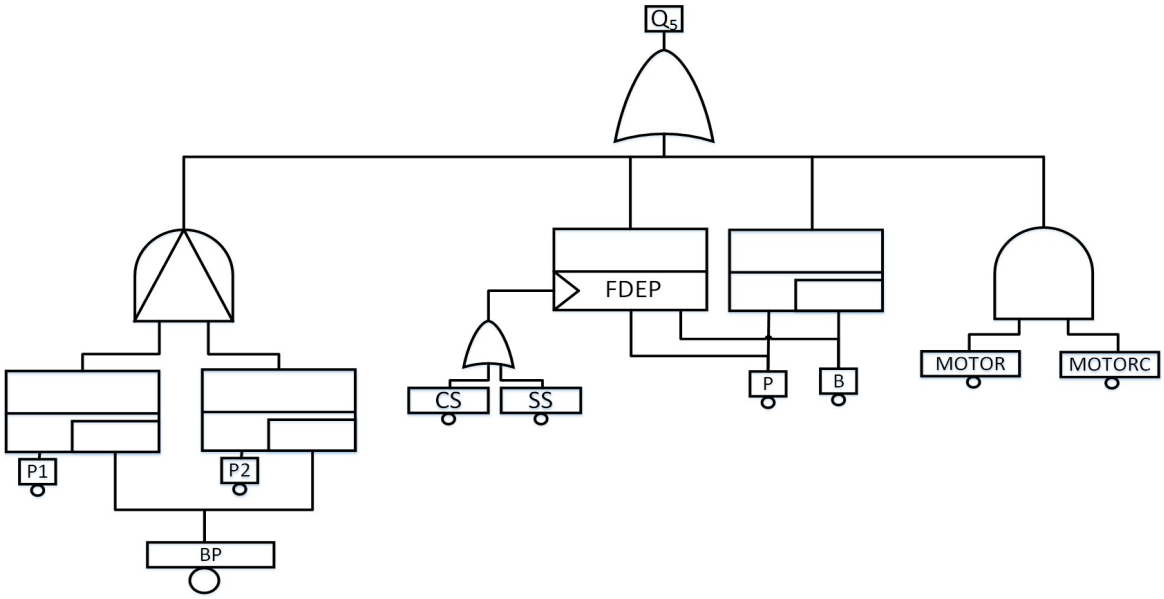


Figure 8: HCAS DFT

The last benchmark is a Hypothetical Cardiac Assist System (HCAS) [3, 7], which consists of two bumps (P_1 and P_2) with a shared spare (BP), two motors and a CPU (P) with a spare (B). Both CPUs are functionally dependent on a trigger, which represents the crossbar switch (CS) and the system supervisor (SS).

In the next section, the verification of the reduction will be introduced for each benchmark along with the reduced cut sets and sequences, then the quantitative analysis for the five benchmarks will be described in the subsequent section.

6.1 Verifying the Reduced DFTs

The first step in the proposed methodology is to create a formal model for both the original DFT and the reduced one. Then, these DFTs are checked if they are equal or not. After the equivalence verification, the cut sets and sequences can be determined.

6.1.1 Verifying the Reduced Cascaded PAND DFT (CPAND)

The top event (Q_1) of the system, shown in Figure 4, is reduced using the simplification theorems as follows:

$$\begin{aligned} Q_1 = (I_1 + J_1 + K_1 + L_1).(A_1 \lhd (I_1 + J_1 + K_1 + L_1). \\ ((B_1C_1.D_1.E_1.W_1.G_1.H_1) \lhd ((I_1 + J_1 + K_1 + L_1) \\ +(I_2 + J_2 + K_2 + L_2).(A_2 \lhd (I_2 + J_2 + K_2 + L_2). \\ ((B_2C_2.D_2.E_2.W_2.G_2.H_2) \lhd ((I_2 + J_2 + K_2 + L_2) \end{aligned} \tag{15}$$

We verify this simplification in HOL as follows:

Theorem 5. ⊢ ∀ A1 B1 C1 D1 E1 W1 G1 H1 I1 J1 K1 L1 N1 O1 P1 A2 B2 C2 D2 E2 W2 G2 H2 I2 J2 K2 L2 N2 O2 P2.

```
ALL_DISTINCT [A1;B1;C1;D1;E1;W1;G1;H1;I1;J1;K1;L1;N1;O1;P1;A2;B2;C2;D2;E2;W2;G2;H2;I2;
J2;K2;L2;N2;O2;P2] ⇒
(D_OR (PAND (D_OR (A1)(D_OR(D_OR N1 O1) P1))(PAND (D_AND A1 (D_AND (D_AND (D_AND (D_AND
(D_AND (D_AND B1 C1) D1) E1) W1) G1) H1))(D_OR (D_OR (D_OR I1 J1) K1) L1)))(PAND (D_OR
A2 (D_OR(D_OR N2 O2) P2))(PAND (D_AND A2 (D_AND (D_AND (D_AND (D_AND (D_AND
(D_AND B2 C2) D2) E2) W2) G2) H2)) (D_OR (D_OR (D_OR I2 J2) K2) L2)))=
D_OR (D_AND (D_OR (D_OR (D_OR I1 J1) K1) L1)(D_AND (D_BEFORE A1 (D_OR (D_OR (D_OR I1 J1)
K1) L1)) (D_BEFORE (D_AND (D_AND (D_AND (D_AND (D_AND (D_AND B1 C1) D1)E1) W1) G1) H1)
(D_OR (D_OR (D_OR I1 J1) K1) L1))))(D_AND (D_OR (D_OR (D_OR I2 J2 ) K2 ) L2) (D_AND
(D_BEFORE A2 (D_OR (D_OR (D_OR I2 J2) K2) L2))(D_BEFORE (D_AND (D_AND (D_AND (D_AND
(D_AND (D_AND B2 C2 ) D2) E2) W2) G2) H2) (D_OR (D_OR (D_OR I2 J2) K2) L2)))))
```

The predicate `ALL_DISTINCT` ensures that the basic events cannot occur at the same time. This condition was found to be a prerequisite for the above-mentioned consequence. From this reduction, it can be noticed that the basic events (N_1, O_1, P_1, N_2, O_2, P_2) have no effect on the failure of the top event since they are eliminated in the reduction. Considering the cut sets and sequences, the top event can fail in two cases. The first case corresponds to the first product in the structure function, which implies that the output event occurs if any one of the basic events (I_1, J_1, K_1, L_1) occurs and A_1 occurs before all of them and the inputs (B_1, C_1, D_1, E_1, W_1, G_1, H_1) occur before the inputs (I_1, J_1, K_1, L_1). The second case represents the second product of the second subtree, which is similar to the first product but with different basic events. Since the Galileo format (which is used to model a DFT in STORM) supports only DFT gates and not operators, it is required that the reduced form is represented using DFT gates only. This representation is verified in HOL as follows:

Theorem 6. ⊢ ∀ A1 B1 C1 D1 E1 W1 G1 H1 I1 J1 K1 L1 N1 O1 P1 A2 B2 C2 D2 E2 W2 G2 H2 I2 J2 K2 L2 N2 O2 P2.

```
ALL_DISTINCT [A1;B1;C1;D1;E1;W1;G1;H1;I1;J1;K1;L1;N1;O1;P1;A2;B2;C2;D2;E2;W2;G2;H2;I2;
J2;K2;L2;N2;O2;P2] ⇒
(D_OR (PAND (D_OR (A1)(D_OR(D_OR N1 O1) P1))(PAND (D_AND A1 (D_AND (D_AND (D_AND (D_AND
(D_AND (D_AND B1 C1) D1) E1) W1) G1) H1))(D_OR (D_OR (D_OR I1 J1) K1) L1)))(PAND (D_OR
A2 (D_OR(D_OR N2 O2) P2))(PAND (D_AND A2 (D_AND (D_AND (D_AND (D_AND (D_AND
(D_AND B2 C2) D2) E2) W2) G2) H2)) (D_OR (D_OR (D_OR I2 J2) K2) L2)))=
D_OR (D_AND (PAND A1 (D_OR (D_OR (D_OR I1 J1) K1) L1))(PAND (D_AND (D_AND (D_AND (D_AND
(D_AND (D_AND B1 C1) D1)E1) W1) G1) H1)(D_OR (D_OR (D_OR I1 J1) K1) L1))) (D_AND (PAND
A2 (D_OR (D_OR (D_OR I2 J2 ) K2 ) L2)(PAND (D_AND (D_AND (D_AND (D_AND (D_AND (D_AND B2
C2 ) D2) E2) W2) G2) H2)(D_OR (D_OR (D_OR I2 J2) K2) L2)))
```

6.1.2 Verifying the Reduced AHRS DFT

The top event (Q_2) of the system shown in Figure 5 is reduced using the algebraic simplification theorems, assuming that the spares are cold spares:

$$Q_2 = Tr + A_{3a}.(A_1 \lhd A_{2a}).(A_{2a} \lhd A_{3a}) + B_{3a}.(B_1 \lhd B_{2a}).(B_{2a} \lhd B_{3a}) \qquad (16)$$

We verify this in HOL as:

Theorem 7. ⊢ ∀ A1 A2_a A2_d A3_a A3_d Tr B1 B2_a B2_d B3_a B3_d.
ALL_DISTINCT [A1; A2_a; A2_d; A3_a; A3_d; Tr; B1; B2_a; B2_d; B3_a; B3_d] ∧
COLD_SPARE A2_d ∧ COLD_SPARE A3_d ∧ COLD_SPARE B2_d ∧ COLD_SPARE B3_d ⇒
(D_OR (WSP (FDEP Tr A1)(WSP (FDEP A2_a Tr) (FDEP A3_a Tr) (FDEP Tr A3_d))
(WSP (FDEP Tr A2_d) (FDEP A3_a Tr) (FDEP A3_d Tr))) (WSP (FDEP Tr A1)
(WSP (FDEP A2_a Tr)(FDEP A3_a Tr) (FDEP Tr A3_d)) (WSP(FDEP Tr A2_d) (FDEP A3_a Tr)
(FDEP A3_d Tr))) =
D_OR Tr (D_OR (D_AND (D_AND A3_a (D_BEFORE A1 A2_a))(D_BEFORE A2_a A3_a))
(D_AND (D_AND B3_a (D_BEFORE B1 B2_a))(D_BEFORE B2_a B3_a))))

This system has three sources of failure; the trigger, the sequence of failure of (A_1 then A_2 then A_3) and finally the sequence (B_1 then B_2 then B_3)

6.1.3 Verifying the Reduced MCS DFT

The top event (Q_3) of the system shown in Figure 6 is reduced using the algebraic simplification theorems to:

$$\begin{aligned} Q_3 = N + PS + (P_1 + D_{11}.(D_{12d} \lhd D_{11}) + D_{12a}.(D_{11} \lhd D_{12a} \\ + M_1.(M_{3d} \lhd M_1) + M_{3a}.(M_1 \lhd M_{3a}) + M_1.(M_2 \lhd M_1) \\ .(P_2 + D_{21}.(D_{22d} \lhd D_{21}) + D_{22a}.(D_{21} \lhd D_{22a} + M_2. \\ (M_{3d} \lhd M_2) + M_{3a}.(M_2 \lhd M_{3a}) + M_2.(M_1 \lhd M_2) \end{aligned} \tag{17}$$

We verify this in HOL as:

Theorem 8. ⊢ ∀ M1 M2 M3_a M3_d D11 D12_a D12_d D21 D22_a D22_d P1 P2 PS N.
ALL_DISTINCT [M1; M2; M3_a; M3_d; D11; D12_a; D12_d; D21; D22_a; D22_d; P1; P2; PS; N]
⇒
(D_OR N (D_AND (D_OR (D_OR (WSP D11 D12_a D12_d) (FDEP PS P1))
(shared_spare M1 M2 M3_a M3_d)) (D_OR (D_OR (WSP D21 D22_a D22_d)(FDEP PS P2))
(shared_spare M2 M1 M3_a M3_d))) =
D_OR (D_OR N PS)(D_AND (D_OR (D_OR P1 (WSP D11 D12_a D12_d))
(shared_spare M1 M2 M3_a M3_d)) (D_OR (D_OR P2 (WSP D21 D22_a D22_d))
(shared_spare M2 M1 M3_a M3_d))))

From this verified reduced function, the sources of failure are: N, PS or the failure of both computers by the failure of any element in each one.

6.1.4 Verifying the Reduced HECS DFT

The top event (Q_4) of the system shown in Figure 7 is reduced using the algebraic simplification theorems to [3]:

$$
\begin{aligned}
Q_4 = A_a.A_2.(A_1 \lhd A_a).(A_1 \lhd A_2) + A_a.A_1.(A_2 \lhd A_a).(A_2 \lhd A_1) \\
+MIU_1.MIU_2 + MIU_1.M_3 + MIU_1.M_4 + MIU_1.M_5 \\
+MIU_2.M_1 + MIU_2.M_2 + MIU_2.M_3 + M_1.M_2.M_3 \\
+M_1.M_2.M_4 + M_1.M_2.M_5 + M_1.M_3.M_4 + M_1.M_3.M_5 \\
+M_1.M_4.M_5 + M_2.M_3.M_4 + M_2.M_3.M_5 + \\
M_2.M_4.M_5 + M_3.M_4.M_5 + BUS_1.BUS_2 + HW + SW + OP
\end{aligned}
\tag{18}
$$

We verify this in HOL as:

Theorem 9. ⊢ ∀ A1 A2 A_a A_d M1 M2 M3 M4 M5 MIU1
MIU2 OP HW SW BUS1 BUS2. NEVER_events A_a A_d ∧ COLD_SPARE A_d ∧
ALL_DISTINCT[A1; A2 ;A_a; A_d; M1 ;M2; M3; M4; M5; MIU1; MIU2; OP; HW; SW; BUS1;
BUS2]⇒
(D_OR (D_OR (D_OR (D_AND (shared_spare A1 A2 A_a A_d) (shared_spare A2 A1 A_a A_d))
(D3of5 (FDEP M5 MIU2) (FDEP M4 MIU2) (FDEP M3 (D_AND MIU2 MIU1)) (FDEP M2 MIU1)
(FDEP M1 MIU1))) (D_AND BUS1 BUS2)) (D_OR (D_OR SW HW) OP) =
D_OR (D_OR (D_OR (D_OR (D_AND (D_AND (D_AND A_a A2) (D_BEFORE A1 A_a))
(D_BEFORE A1 A2)) (D_AND (D_AND (D_AND A_a A1) (D_BEFORE A2 A_a)) (D_BEFORE A2 A1)))
(D_OR (D_OR (D_OR (D_OR (D_OR (D_OR (D_OR (D_OR (D_OR (D_OR (D_OR (D_OR (D_OR (D_OR
(D_OR (D_OR (D_AND MIU1 MIU2) (D_AND MIU1 M3)) (D_AND MIU1 M4)) (D_AND MIU1 M5))
(D_AND MIU2 M1)) (D_AND MIU2 M2)) (D_AND MIU2 M3)) (D_AND (D_AND M1 M2) M3))
(D_AND (D_AND M1 M2) M4)) (D_AND (D_AND M1 M2) M5)) (D_AND (D_AND M1 M3) M4))
(D_AND (D_AND M1 M3) M5)) (D_AND (D_AND M1 M4) M5)) (D_AND (D_AND M2 M3) M4))
(D_AND (D_AND M2 M3) M5)) (D_AND (D_AND M2 M4) M5)) (D_AND (D_AND M3 M4) M5)))
(D_AND BUS1 BUS2)) (D_OR (D_OR SW HW) OP))

where D3of5 is a combination of AND and OR operators to create the voting 3 out of 5 gate. The cut sets and sequences of this function can be easily obtained from Equation 18. To be able to model this function in STORM, it was verified in HOL in terms of the dynamic gates as:

Theorem 10. ⊢ ∀ A1 A2 A_a A_d M1 M2 M3 M4 M5 MIU1 MIU2 OP HW SW BUS1 BUS2.
NEVER_events A_a A_d ∧ COLD_SPARE A_d ∧
ALL_DISTINCT[A1; A2 ;A_a; A_d; M1 ;M2; M3; M4; M5; MIU1; MIU2; OP; HW; SW; BUS1;
BUS2]⇒
(D_OR (D_OR (D_OR (D_AND (shared_spare A1 A2 A_a A_d) (shared_spare A2 A1 A_a A_d))
(D3of5 (FDEP M5 MIU2) (FDEP M4 MIU2) (FDEP M3 (D_AND MIU2 MIU1)) (FDEP M2 MIU1)
(FDEP M1 MIU1))) (D_AND BUS1 BUS2)) (D_OR (D_OR SW HW) OP) =
D_OR (D_OR (D_OR (D_OR (D_AND (shared_spare A1 A2 A_a A_d)
(shared_spare A2 A1 A_a A_d)) (D_OR (D_OR (D_OR (D_OR (D_OR (D_OR (D_OR (D_OR (D_OR
(D_OR (D_OR (D_OR (D_OR (D_OR (D_OR (D_OR (D_AND MIU1 MIU2) (D_AND MIU1 M3))
(D_AND MIU1 M4)) (D_AND MIU1 M5)) (D_AND MIU2 M1)) (D_AND MIU2 M2)) (D_AND MIU2 M3))
(D_AND (D_AND M1 M2) M3)) (D_AND (D_AND M1 M2) M4)) (D_AND (D_AND M1 M2) M5))
(D_AND (D_AND M1 M3) M4)) (D_AND (D_AND M1 M3) M5)) (D_AND (D_AND M1 M4) M5))
(D_AND (D_AND M2 M3) M4)) (D_AND (D_AND M2 M3) M5)) (D_AND (D_AND M2 M4) M5))
(D_AND (D_AND M3 M4) M5))) (D_AND BUS1 BUS2)) (D_OR (D_OR SW HW) OP))

6.1.5 Verifying the Reduced HCAS DFT

The top event (Q_5) of the system shown in Figure 8 is reduced using the algebraic simplification theorems to [3]:

$$\begin{aligned} Q_5 = CS + SS + MOTOR.MOTORC + P.(B_d \lhd P) + B_a.(P \lhd B_a) \\ + BP_a.(P_2 \lhd P_1).(P_1 \lhd BP_a) + P_2.(P_1 \lhd BP_a).(BP_a \lhd P_2) \end{aligned} \tag{19}$$

We verify this in HOL as:

Theorem 11. ⊢ ∀ CS SS MOTOR MOTORC P B_d B_a BP_a BP_d P1 P2. NEVER_events B_a B_d ∧
ALL_DISTINCT [P1; P2; BP_a; BP_d; P; B_a; B_d; CS; SS; MOTOR; MOTORC]∧
(D_BEFORE B_a P = NEVER) ∧ (D_AND (D_BEFORE BP_a P1) (D_BEFORE P1 P2) = NEVER) ∧
(D_AND (D_BEFORE BP_a P2) (D_BEFORE P2 P1) = NEVER) ∧
NEVER_events BP_a BP_d ∧ COLD_SPARE BP_d ⇒
(D_OR (D_OR (WSP (FDEP P (D_OR CS SS)) (FDEP B_a (D_OR CS SS))
(FDEP B_d (D_OR CS SS))) (D_AND MOTOR MOTORC)) (PAND (shared_spare P1 P2 BP_a BP_d)
(shared_spare P2 P1 BP_a BP_d)) =
D_OR (D_OR (D_OR (D_OR (D_OR (D_OR CS SS) (D_AND MOTOR MOTORC))
(D_AND P (D_BEFORE B_d P))) (D_AND B_a (D_BEFORE P B_a))) (D_AND (D_AND BP_a
(D_BEFORE P2 P1)) (D_BEFORE P1 BP_a))) (D_AND (D_AND P2 (D_BEFORE P1 BP_a))
(D_BEFORE BP_a P2)))

The cut sets and sequences for Q_5 can be obtained from the verified reduced function. To model this function in STORM, it was verified using the dynamic gates, assuming that B is a cold spare:

Theorem 12. ⊢ ∀ CS SS MOTOR MOTORC P B_d B_a BP_a BP_d P1 P2. NEVER_events B_a B_d ∧
ALL_DISTINCT [P1; P2; BP_a; BP_d; P; B_a; B_d; CS; SS; MOTOR; MOTORC]∧
(D_BEFORE B_a P = NEVER) ∧ (D_AND (D_BEFORE BP_a P1) (D_BEFORE P1 P2) = NEVER) ∧
(D_AND (D_BEFORE BP_a P2) (D_BEFORE P2 P1) = NEVER) ∧
NEVER_events BP_a BP_d ∧ COLD_SPARE BP_d ⇒
(D_OR (D_OR (WSP (FDEP P (D_OR CS SS)) (FDEP B_a (D_OR CS SS))
(FDEP B_d (D_OR CS SS))) (D_AND MOTOR MOTORC)) (PAND (shared_spare P1 P2 BP_a BP_d)
(shared_spare P2 P1 BP_a BP_d)) =
D_OR (D_OR (D_OR (D_OR CS SS) (D_AND MOTOR MOTORC)) (D_AND B_a (D_BEFORE P B_a)))
(PAND (shared_spare P1 P2 BP_a BP_d) (shared_spare P2 P1 BP_a BP_d)))

6.2 Probabilistic Analysis Results using STORM

The quantitative analysis for the five benchmarks was conducted using STORM on a Linux machine with i7 2.4 GHZ quad core CPU and 4 GB of RAM. The efficiency of the proposed methodology is highlighted by analyzing the original DFTs and the reduced ones. In addition, the probability of failure for each DFT is evaluated for different time bounds, e.g. the probability of failure after 100 working time units. A summary of the analysis results are given in Table 6. It can be noticed that the number of states is reduced as well as the total analysis time. For the first benchmark, the analysis time is reduced due to the huge reduction in the number of states. As mentioned earlier, many basic events are eliminated using the algebraic reduction theorems, which in turn reduced the total analysis time as well as the number of states. For the rest of the benchmarks, the analysis time is significantly reduced when the reduced DFT is

used in the analysis. This is mainly because of two reasons, firstly, the number of states is reduced, and secondly, the original DFT is modeled as a Markov Automata (MA) as there is a non-deterministic behavior, while the reduced DFT is modeled as a Continuous Time Markov Chain (CTMC). This means that in the reduced DFT the non-deterministic behavior caused by the failure dependency does not exist any more, as the reduction process depends on the time of failure of the gates. We used the STORM command (firstdep) [16] to resolve the non-deterministic behavior in the original DFT to generate a CTMC instead of a MA, and the results in Table 7 show that the number of states for the reduced DFTs is generally smaller than that of the original DFT with resolved dependencies, which emphasizes on the importance of the proposed methodology not only in providing a formal qualitative analysis but also in reducing the quantitative analysis cost in terms of time and memory, i.e., number of states.

Table 6: STORM Analysis Results (Before and After Reduction)

DFT	*Time*	*Before Reduction*			*After Reduction*		
	Bound	*#States*	*Analysis Time(sec)*	*Probability of Failure*	*#States*	*Analysis Time(sec)*	*Probability of Failure*
CPAND	1000	148226 (CTMC)	7.488	1.464103531e-4	66050 (CTMC)	3.032	1.464103348e-4
ARHS	10	74 (MA)	169.81	0.00995049462	10 (CTMC)	0.067	0.009950461197
	100	74 (MA)	*	**	10 (CTMC)	0.067	0.954423939
MCS	10	89 (MA)	139.7	0.01196434683	29 (CTMC)	0.061	0.01196434516
	100	89 (MA)	*	**	29 (CTMC)	0.060	0.1166464887
HECS	10	1051 (MA)	16359.83	0.01710278909	505 (CMTC)	0.123	0.01710276373
	100	1051 (MA)	*	**	505 (CMTC)	0.123	0.1762782397
HCAS	10	181 (MA)	275.31	2.000104327e-5	37 (CTMC)	0.070	2.99929683e-5
	100	181 (MA)	*	**	37 (CTMC)	0.071	0.000300083976

* The analysis did not finish within 4 hours
** No probabilities are recorded (analysis did not finish)

Table 7: STORM Analysis Results with Resolved Dependencies

DFT	*Time*	*Dependency resolved in STORM*			*Algebraic Reduction*		
	Bound	*#States*	*Analysis Time(sec)*	*Probability of Failure*	*#States*	*Analysis Time(sec)*	*Probility of Failure*
ARHS	10	10(CTMC)	0.068	0.009960461197	10 (CTMC)	0.067	0.009950461197
	100	10 (CTMC)	0.1	0.09544239393	10 (CTMC)	0.067	0.954423939
MCS	10	45 (CMTC)	0.064	0.01196434516	29 (CTMC)	0.061	0.01196434516
	100	45(CMTC)	0.064	0.1166464887	29 (CTMC)	0.060	0.1166464887
HECS	10	379 (CTMC)	0.118	0.01710276373	505 (CMTC)	0.123	0.01710276373
	100	379 (CTMC)	0.121	0.1762782397	505 (CMTC)	0.123	0.1762782397
HCAS	10	73 (CTMC)	0.076	1.999530855e-5	37 (CTMC)	0.070	2.99929683e-5
	100	73 (CTMC)	0.076	0.0002001091927	37 (CTMC)	0.071	0.000300083976
	100000	73 (CTMC)	0.077	0.2772192934*	37 (CTMC)	0.074	0.3460009685*

* The reported probability for the reduced DFT is closer to the probability reported in [3] for the same input failure distribution

7 Conclusion

In this work, we proposed a formal dynamic fault tree analysis methodology integrating theorem proving and model checking. We formalized the dynamic fault tree gates and operators in higher-order logic based on the time of failure of each gate. Using our formalization of the gates and the `extreal` library in HOL4, we proved over eighty simplification theorems that can be used to verify the reduction of any DFT. We used these theorems to verify the equivalence of the original and reduced DFT using theorem proving. In addition, we provided a formally verified qualitative analysis of the structure function in the form of reduced cut sets and sequences, which, to the best of our knowledge, is a novel contribution. The quantitative analysis of the reduced structure function is performed using model checking. This ensures that the model checking results correspond to the original DFT, since we use the formally verified reduced DFT in the quantitative analysis. Both the qualitative and the quantitative analyses were conducted on five benchmark DFTs, and the analysis results show that our proposed integrated methodology provides a formally verified reduced cut sets and sequences. In addition, the model checking results indicate that using the reduced DFT in the analysis has a positive impact on its cost in terms of both time and number of states. As a future work, we plan to provide the quantitative analysis of DFTs within the HOL theorem prover, which will allow us to have a complete framework for formal DFT analyses using theorem proving.

References

[1] E. Ruijters and M. Stoelinga, "Fault Tree Analysis: A Survey of the State-of-the-art in Modeling, Analysis and Tools," *Computer Science Review*, vol. 15-16, pp. 29 – 62, 2015.

[2] M. Stamatelatos, W. Vesely, J. Dugan, J. Fragola, J. Minarick, and J. Railsback, *Fault Tree Handbook with Aerospace Applications*. NASA Office of Safety and Mission Assurance, 2002.

[3] G. Merle, "Algebraic Modelling of Dynamic Fault Trees, Contribution to Qualitative and Quantitative Analysis," Ph.D. dissertation, ENS, France, 2010.

[4] C. Dehnert, S. Junges, J.-P. Katoen, and M. Volk, "A Storm is Coming: A Modern Probabilistic Model Checker," in *Computer Aided Verification*. Springer, 2017, pp. 592–600.

[5] M. Volk, S. Junges, and J.-P. Katoen, "Fast Dynamic Fault Tree Analysis by Model Checking Techniques," *IEEE Transactions on Industrial Informatics, doi: 10.1109/TII.2017.2710316*, 2017.

[6] W. Ahmad and O. Hasan, "Formalization of Fault Trees in Higher-order Logic: A Deep Embedding Approach," in *Dependable Software Engineering: Theories, Tools, and Applications*, ser. LNCS, vol. 9984. Springer, 2016, pp. 264–279.

[7] H. Boudali, P. Crouzen, and M. Stoelinga, "A Compositional Semantics for Dynamic Fault Trees in terms of Interactive Markov Chains," in *Automated Technology for Verification and Analysis*, ser. LNCS, vol. 4762. Springer, 2007, pp. 441–456.

[8] G. Merle, J. M. Roussel, J. J. Lesage, and A. Bobbio, "Probabilistic Algebraic Analysis of Fault Trees with Priority Dynamic Gates and Repeated Events," *IEEE Transactions on Reliability*, vol. 59, no. 1, pp. 250–261, 2010.

[9] M. Malhotra and K. S. Trivedi, "Dependability Modeling using Petri-nets," *IEEE Transactions on Reliability*, vol. 44, no. 3, pp. 428–440, 1995.

[10] H. Boudali and J. Dugan, "A New Bayesian Network Approach to Solve Dynamic Fault Trees," in *IEEE Reliability and Maintainability Symposium.*, 2005, pp. 451–456.

[11] L. Pullum and J. Dugan, "Fault Tree Models for the Analysis of Complex Computer-based Systems," in *IEEE Reliability and Maintainability Symposium*, 1996, pp. 200–207.

[12] Galileo, "www.cse.msu.edu/∼cse870/materials/faulttolerant/manual-galileo.htm."

[13] F. Arnold, A. Belinfante, F. Van der Berg, D. Guck, and M. Stoelinga, "Dftcalc: A Tool for Efficient Fault Tree Analysis," in *Computer Safety, Reliability, and Security*, ser. LNCS, vol. 8153. Springer, 2013, pp. 293–301.

[14] M. Ghadhab, S. Junges, J.-P. Katoen, M. Kuntz, and M. Volk, "Model-based Safety Analysis for Vehicle Guidance Systems," in *Computer Safety, Reliability, and Security*, ser. LNCS, vol. 10488. Springer, 2017, pp. 3–19.

[15] W. Ahmad and O. Hasan, "Towards Formal Fault Tree Analysis using Theorem Proving," in *Intelligent Computer Mathematics*, ser. LNCS, vol. 9150. Springer, 2015, pp. 39–54.

[16] STORM, "www.stormchecker.org/index.html," 2017.