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

Yassmeen Elderhalli(✉), Osman Hasan, Waqar Ahmad, and Sofiène Tahar

Electrical and Computer Engineering, Concordia University, Montréal, Canada
{y_elderh,o_hasan,waqar,tahar}@ece.concordia.ca

Abstract. Dynamic fault trees (DFTs) have emerged as an important tool for capturing the dynamic behavior of system failure. These DFTs are analyzed qualitatively and quantitatively using stochastic or algebraic methods. Model checking has been proposed to conduct the failure analysis of systems using DFTs. However, it has not been used for DFT qualitative analysis. Moreover, its analysis time grows exponentially with the number of states and its reduction algorithms are usually not formally verified. To overcome these limitations, we propose a methodology to perform the formal analysis of DFTs using an integration of theorem proving and model checking. We formalize the DFT gates in higher-order logic and formally verify many algebraic simplification properties using theorem proving. Based on this, we prove the equivalence between raw DFTs and their reduced forms to enable the formal qualitative analysis (determine the cut sets and sequences) of DFTs with theorem proving. We then use model checking to perform the quantitative analysis (compute probabilities of failure) of the formally verified reduced DFT. We applied our methodology on 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

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 an 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 model a more realistic representation

A. Dutle et al. (Eds.): NFM 2018, LNCS 10811, pp. 139–156, 2018.
https://doi.org/10.1007/978-3-319-77935-5_10

of systems compared to SFTs. For example, the spare DFT gate can model the failure of car tires and their spares, which cannot be modeled using SFT gates.

Fault Tree Analysis (FTA) [1] has become an essential part of the design process of safety-critical systems, 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 results about the probability of failure of the top event and the mean-time-to-failure (MTTF) among other metrics [1]. Dynamic FTA is commonly carried out using algebraic [3] or Markov chain based approaches [2]. In the former, an algebra similar to the Boolean algebra is used to determine the structure function of the top event, which can be simplified to determine a reduced form of the cut sets and sequences. The probabilistic analysis of the DFT 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 the failure analysis of safety-critical systems, formal methods 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]. 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. Therefore, one cannot ascertain that the analysis results after reduction are accurate or correspond to the original system. On the other hand, while in theory higher-order logic (HOL) theorem proving can cater for the above shortcomings, its support for FTA has been limited to SFTs [6].

In this paper, we propose to overcome the above-mentioned limitations of formal DFT analysis by using an integration of model checking and theorem proving. Firstly, we use theorem proving for modeling DFTs and verifying the equivalence between the original and the reduced form of the DFT. The formally verified reduced DFT can then be used for qualitative analysis (determining the cut sets and sequences) as well as for quantitative analysis using model checking. Thus, our proposed methodology tends to provide more sound results than sole model checking based analysis thanks 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 with a reduced DFT model of a given system for the quantitative analysis. In order to illustrate the utilization and effectiveness of our proposed methodology, we analyzed five DFT benchmarks, namely: 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 the HOL4 theorem prover [11]. We use the STORM model checker to formally analyze the original as well as the reduced DFT. The results show that using the verified reduced DFT for the quantitative analysis allows us to reduce both the number of generated states by the model checker by up to 6 times and the time required to perform the analysis up to 133000 times faster.

2 Related Work

DFT analysis has been done 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 [12], where the DFT is divided into two parts: static and dynamic. The static subtree is analyzed using 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 [13]. 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 [14]. 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 CPU 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 a 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 corre-

sponding Markovian model and then analyze the safety characteristics quantitatively of the given system using the model checker [15]. The STORM model checker accepts the DFT to be analyzed in the Galileo format [13] 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 for large systems. 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 and Hasan [6,16] formalized SFTs 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 paper, 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 a quantitative analysis on the reduced DFT. As a result, a formally verified reduced form of the cut sequences is obtained. In addition, both the generated state machine and the analysis time are reduced.

3 Proposed Methodology

Our proposed methodology of the formal DFT analysis is depicted in Fig. 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 [17]. 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 DFT 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.

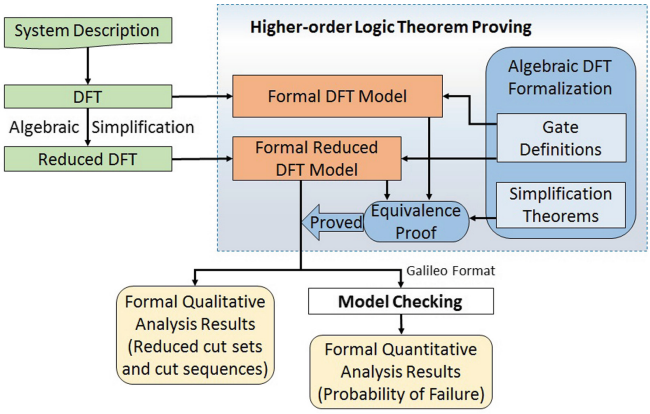


Fig. 1. Overview of proposed methodology

The qualitative analysis represents an important and 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 SFTs, the qualitative analysis is performed by finding the cut sets, which are the combination of events that causes system failure without providing any information about the required sequence that will cause the failure. The temporal behavior of dynamic gates allows representing the failure dependencies and sequences in a system. Due to this 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, we utilize a HOL theorem prover to verify a reduced expression of the structure function of the top event, which ensures that the reduction process is correct. Using this

reduced structure function, a formally verified reduced form of the cut sequences can also be determined.

The 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 get a reduction in the analysis time and number of states. In this paper, 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 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 failure event is true. It is also assumed that the events are non-repairable.

4.1 Identity Elements

Two identity elements are defined, namely the *ALWAYS* and the *NEVER* elements. The *ALWAYS* identity element represents an event with a failure time equal 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*
$\vdash$ `ALWAYS` = (`0 : extreal`)

Definition 2. *NEVER element*
$\vdash$ `NEVER` = `PosInf`

where `extreal` is the HOL data-type for extended real numbers, which are real numbers including positive infinity ($+\infty$) and negative infinity ($-\infty$). The `PosInf` is a HOL symbol representing ($+\infty$).

4.2 Temporal Operators

We also formalize three temporal operators in order to model the dynamic behavior of the DFT: *Simultaneous* (Δ), *Before* ($\lhd$) and *Inclusive Before* ($\trianglelefteq$). The *Simultaneous* operator has two input events, representing 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:

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

For any two basic events, if the failure distribution of the random variables associated with these basic events is continuous, then they cannot have the same time of failure [3], 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 (from 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 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

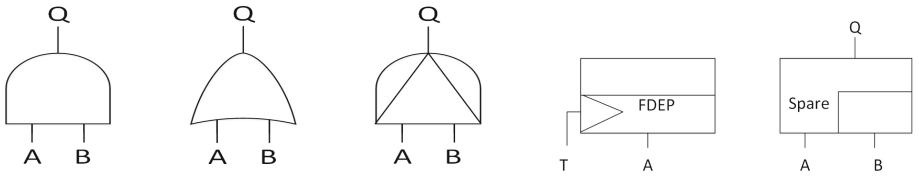
Definition 4. *Before Operator*
⊢ ∀ (A : extreal) B. D_BEFORE A B = if (A < B) then A else PosInf

Definition 5. *Inclusive Before Operator*
⊢ ∀ (A : extreal) B. D_INCLUSIVE_BEFORE A B = if (A ≤ B) then A else PosInf

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

4.3 Fault Tree Gates

Figure 2 shows the main FT gates [2]; dynamic gates as well as the static ones. Although, the AND (·) and OR (+) gates, shown in Figs. 2a and b, are considered as static operators or gates, their behavior can be represented using the time of



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

Fig. 2. Fault tree gates

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 HOL 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 a sequence, conventionally from left to right. For the PAND gate, shown in Fig. 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 can be expressed using the temporal operators as:

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

We define the PAND gate in HOL as:

Definition 8. *PAND gate*
$\vdash \forall$ (A : extreal) B. PAND A B = if (A $\leq$ B) then B else PosInf

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

Theorem 1. $\vdash \forall$ A B. PAND A B = D_AND B (D_INCLUSIVE_BEFORE A B)

The Functional DEPendency gate (FDEP), shown in Fig. 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 Fig. 2d, the occurrence of T triggers the occurrence of A. This states that A occurs in two different ways: firstly, when A occurs by itself and secondly, when the trigger T occurs. This implies 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 gate in HOL as:

Definition 9. *FDEP gate*
$\vdash \forall$ (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 gate is equivalent to an OR gate as follows:

Theorem 2. $\vdash \forall$ A T. FDEP A T = D_OR A T

The spare gate, shown in Fig. 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 Fig. 2e, has a main input (A) and a spare input (B). After the failure of input A, the spare input 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: hot spare gate (HSP), cold spare gate (CSP) or 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 Fig. 2e, B will be represented using two variables; B_d for the dormant state and B_a for the active 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 B fails in its dormant state (inactive) then A fails with no spare to replace it. The second way is when A fails first then B (the spare part) is activated and then B fails in its active state. For the general case, when 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 considering the two input events occur at the same time. The general form of the warm spare gate can be expressed mathematically as:

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

We formally define 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:

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

The above equation describes 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 formally 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 the predicate `ALL_DISTINCT` ensures that A and B_a are not equal, implying that they cannot fail at the same time, and `COLD_SPARE B_d` makes sure that the spare B is a cold spare, i.e., it cannot fail in its dormant mode (B_d).

The failure distribution of the spare part in the HSP remains the same in both states, i.e., the dormant and the active states. The output of the HSP fails when both the primary and the spare inputs fail, and the sequence of failures

does not matter, as the spare part has only one failure distribution. The HSP can be expressed mathematically as:

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

where A is the primary input and B is the spare. It is formally defined in HOL as:

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

We formally 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)

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 Fig. 3, are expressed mathematically as follows (assuming that A, B and C are basic events):

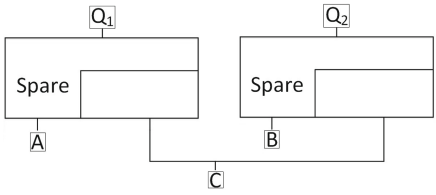


Fig. 3. Spare gates with shared spare

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

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

The last term in Eq. 13 ($A.(B \lhd A)$) indicates that if B occurs before A, then the spare part C is used by the second spare gate. This implies 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)))

4.4 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 [3]. We formally verified over 80 simplification theorems for the operators, defined in the previous subsection, including commutativity, associativity and idempotence of the AND, OR and Simultaneous operators, in addition to more complex theorems that include a combination of temporal 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, defined using the `extreal` data-type in HOL. During the verification process, most sub-goals were automatically verified using tactics that utilize theorems from the `extreal` HOL theory. Some of these formally verified theorems are listed in Table 1. The complete list of formally verified theorems and more details about their verification can be found in [18].

Table 1. Some formally verified simplification theorems

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

Figure 4 shows an example of a simple DFT [8]. This DFT consists of two cascaded PAND gates with three basic events; A, B and C. The temporal operators can be used to express the behavior of the PAND gate as follows [8]:

$$Q = C.(B \trianglelefteq C).(A \trianglelefteq (C.(B \trianglelefteq C))) \tag{15}$$

Using this expression, we cannot determine the required sequence of failure for the basic events that will cause the system failure, since the basic events are repeated in the expression. Using the algebraic simplification theorems, this structure function can be reduced to [8]:

$$Q = C.(B \lhd C).(A \lhd C) \tag{16}$$

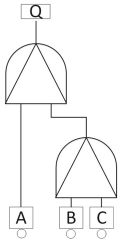


Fig. 4. Simple DFT example

We verified this reduction using HOL4, which implies that this reduction process is correct and the new reduced DFT expression reflects the behavior of the original DFT.

```
⊢ ∀ A B C.
    ALL_DISTINCT [A; B; C] ⇒
      (PAND A (PAND B C) =
      D_AND C (D_AND (D_BEFORE A C) (D_BEFORE B C)))
```

From this reduced expression, we can identify that two different sequences can cause the system failure; $[A, B, C]$ or $[B, A, C]$.

$$Q = C \;.\; (A \lhd C) \;.\; (B \lhd A) \;+\; C \;.\; (B \lhd C) \;.\; (A \lhd B) \tag{17}$$

This reduced form of the structure function is verified in HOL to be equal to the top event of the original tree as:

```
⊢ ∀ A B C.
    ALL_DISTINCT [A; B; C] ⇒
      (PAND A (PAND B C) =
      D_OR(D_AND C (D_AND (D_BEFORE A C) (D_BEFORE B A)))
           (D_AND C (D_AND (D_BEFORE B C) (D_BEFORE A B))))
```

Since we have formally verified that the structure function is composed of the above-mentioned two sequences, we can conclude that the system will fail if any of these sequences occurs. In order to prevent or reduce the probability of failure of the top event, we should prevent the occurrence of these sequences, i.e., we should prevent the failure of A and B before C. This means that using the first part of our proposed methodology, we have been able to obtain a verified reduced form of the top event as well as a verified reduced form of the cut sequences.

5 Experimental Results

In order to illustrate the effectiveness of our proposed methodology, we conducted the formal DFT analysis of five benchmarks. The first benchmark, depicted in Fig. 5, is a scaled version of the original cascaded PAND DFT [7,8] with repeated

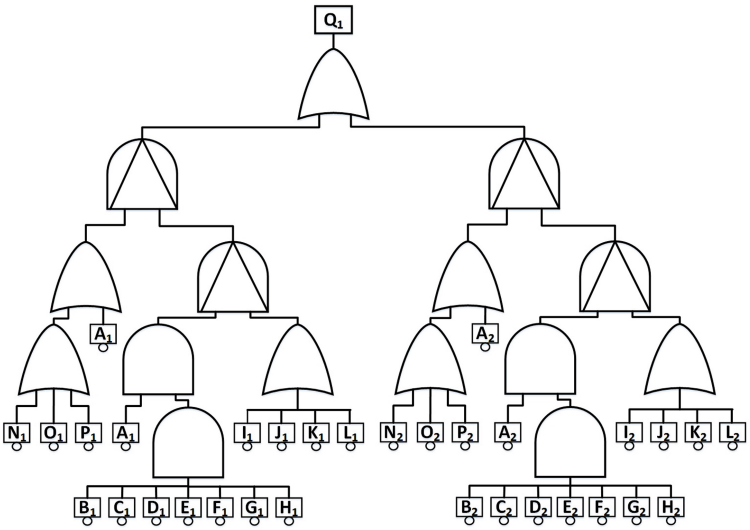


Fig. 5. Scaled cascaded PAND DFT

events. It has two similar subtrees with different basic events and a top event that fails whenever one of these subtrees 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 or B) has two spare parts, and the unit fails 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. The last benchmark is a Hypothetical Cardiac Assist System (HCAS) [3,7], which consists of two bumps with a shared spare, two motors and a CPU with a spare. Both CPUs are functionally dependent on a trigger, which represents the crossbar switch and the system supervisor. In the sequel, we describe the formal analysis of the first benchmark. Details of the rest of the benchmarks as well as the HOL4 scripts and STORM models are available at [17,18].

5.1 Formal Verification of the Reduced Cascaded PAND DFT

The first step in the proposed methodology is to create a formal model for both the original DFT and the reduced one. Then, the equivalence property between them is formally verified in HOL. This is followed by determining the cut sets and sequences. The top event (Q_1) of the system, shown in Fig. 5, is reduced using the simplification theorems and this reduction is verified in HOL as follows [18]:

Theorem 5. $\vdash \forall A_1\ B_1\ C_1\ D_1\ E_1\ W_1\ G_1\ H_1\ I_1\ J_1\ K_1\ L_1\ N_1\ O_1\ P_1\ A_2\ B_2\ C_2\ D_2\ E_2\ W_2\ G_2\ H_2\ I_2\ J_2\ K_2\ L_2\ N_2\ O_2\ P_2.$ ALL_DISTINCT $[A_1; B_1; C_1; D_1; E_1; W_1; G_1; H_1; I_1; J_1; K_1; L_1; N_1; O_1; P_1; A_2; B_2; C_2; D_2; E_2; W_2; G_2; H_2; I_2; J_2; K_2; L_2; N_2; O_2; P_2] \Rightarrow$
$(Q_1 = (I_1+J_1+K_1+L_1).(A_1 \lhd (I_1+J_1+K_1+L_1)).((B_1.C_1.D_1.E_1.W_1.G_1.H_1) \lhd (I_1+J_1+K_1+L_1)) + (I_2+J_2+K_2+L_2).(A_2 \lhd (I_2+J_2+K_2+L_2)).((B_2.C_2.D_2.E_2.W_2.G_2.H_2) \lhd (I_2+J_2+K_2+L_2)))$

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. We can observe from the above theorem 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. $\vdash \forall A_1\ B_1\ C_1\ D_1\ E_1\ W_1\ G_1\ H_1\ I_1\ J_1\ K_1\ L_1\ N_1\ O_1\ P_1\ A_2\ B_2\ C_2\ D_2\ E_2\ W_2\ G_2\ H_2\ I_2\ J_2\ K_2\ L_2\ N_2\ O_2\ P_2.$ ALL_DISTINCT $[A_1; B_1; C_1; D_1; E_1; W_1; G_1; H_1; I_1; J_1; K_1; L_1; N_1; O_1; P_1; A_2; B_2; C_2; D_2; E_2; W_2; G_2; H_2; I_2; J_2; K_2; L_2; N_2; O_2; P_2] \Rightarrow$
$(Q_1 =$ PAND A_1 $(I_1+J_1+K_1+L_1).$ PAND $(B_1.C_1.D_1.E_1.W_1.G_1.H_1)$ $(I_1+J_1+K_1+L_1)+$ PAND $A_2(I_2+J_2+K_2+L_2).$ PAND $(B_2.C_2.D_2.E_2.W_2.G_2.H_2)$ $(I_2+J_2+K_2+L_2)$

5.2 Quantitative 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 2. 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. The reduction in the analysis time is also evident in the rest of the benchmarks, as given in Table 2. 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 implies 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, which allows solving the previously unresolved problems. We used STORM command (*firstdep*) to resolve the non-deterministic behavior in the original DFT to generate a CTMC instead of a MA. The results in Table 3 show that the number of states for the reduced DFTs is generally less than that of the original DFT with resolved dependencies, except for the HECS DFT, which requires further investigation. This 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. We believe that the proposed methodology can be implemented with any theorem prover that supports extended real data-type and with any probabilistic model checker that supports DFT analysis. In addition, we believe that applying this methodology to any DFT will reduce the analysis time as well as the number of states specially if the DFT has a non-deterministic behavior. This methodology can be enhanced if the model checker supports the temporal operators in addition to the supported FT gates. This means that we can use the result of the reduction directly without rewriting it in terms of FT gates. Moreover, the transformation process of the verified reduced DFT expression from the theorem prover script to its corresponding Galileo format can be automated to facilitate the overall analysis.

Table 2. 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	0.0954423939
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 h.

** No probabilities are recorded (analysis did not finish).

Table 3. 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	0.0954423939
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.

6 Conclusion

In this paper, we proposed a formal dynamic fault tree analysis methodology integrating theorem proving and model checking approaches. We first formalized the dynamic fault tree gates and operators in HOL theorem proving based on the time of failure of each gate. Using this formalization and the `extreal` library in HOL4, we also proved over 80 simplification theorems that can be used to verify the reduction of any DFT. We used these theorems to verify the equivalence of the raw and reduced DFTs 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 another novel contribution. The quantitative analysis of the reduced structure function is performed using the STORM model checker. This ensures that the model checking results correspond to the original DFT, since we use the formally verified reduced DFT model for the quantitative analysis. Both the qualitative and quantitative analyses were conducted on five benchmark DFTs, providing formally verified reduced cut sets and sequences, as well as the corresponding probabilities of failure. 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 HOL, which will allow us to have a complete framework for formal DFT analyses using theorem proving.

References

1. Ruijters, E., Stoelinga, M.: Fault tree analysis: a survey of the state-of-the-art in modeling, analysis and tools. Comput. Sci. Rev. **15–16**, 29–62 (2015)
2. Stamatelatos, M., Vesely, W., Dugan, J., Fragola, J., Minarick, J., Railsback, J.: Fault tree handbook with aerospace applications. NASA Office of Safety and Mission Assurance (2002)

3. Merle, G.: Algebraic Modelling of Dynamic Fault Trees, Contribution to Qualitative and Quantitative Analysis. Ph.D. thesis, ENS, France (2010)
4. Dehnert, C., Junges, S., Katoen, J.-P., Volk, M.: A Storm is coming: a modern probabilistic model checker. In: Majumdar, R., Kunčak, V. (eds.) CAV 2017. LNCS, vol. 10427, pp. 592–600. Springer, Cham (2017). https://doi.org/10.1007/978-3-319-63390-9_31
5. Volk, M., Junges, S., Katoen, J.P.: Fast dynamic fault tree analysis by model checking techniques. IEEE Trans. Ind. Inf. **14**, 370–379 (2017). https://doi.org/10.1109/TII.2017.2710316
6. Ahmad, W., Hasan, O.: Towards formal fault tree analysis using theorem proving. In: Kerber, M., Carette, J., Kaliszyk, C., Rabe, F., Sorge, V. (eds.) CICM 2015. LNCS (LNAI), vol. 9150, pp. 39–54. Springer, Cham (2015). https://doi.org/10.1007/978-3-319-20615-8_3
7. Boudali, H., Crouzen, P., Stoelinga, M.: A compositional semantics for dynamic fault trees in terms of interactive Markov chains. In: Namjoshi, K.S., Yoneda, T., Higashino, T., Okamura, Y. (eds.) ATVA 2007. LNCS, vol. 4762, pp. 441–456. Springer, Heidelberg (2007). https://doi.org/10.1007/978-3-540-75596-8_31
8. 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)
9. Malhotra, M., Trivedi, K.S.: Dependability modeling using petri-nets. IEEE Trans. Reliab. **44**(3), 428–440 (1995)
10. Boudali, H., Dugan, J.: A new Bayesian network approach to solve dynamic fault trees. In: IEEE Reliability and Maintainability Symposium, pp. 451–456 (2005)
11. HOL4 (2017). hol.sourceforge.net
12. Pullum, L., Dugan, J.: Fault tree models for the analysis of complex computer-based systems. In: IEEE Reliability and Maintainability Symposium, pp. 200–207 (1996)
13. Galileo. www.cse.msu.edu/~cse870/Materials/FaultTolerant/manual-galileo.htm
14. Arnold, F., Belinfante, A., Van der Berg, F., Guck, D., Stoelinga, M.: DFTCalc: a tool for efficient fault tree analysis. In: Bitsch, F., Guiochet, J., Kaâniche, M. (eds.) SAFECOMP 2013. LNCS, vol. 8153, pp. 293–301. Springer, Heidelberg (2013). https://doi.org/10.1007/978-3-642-40793-2_27
15. Ghadhab, M., Junges, S., Katoen, J.-P., Kuntz, M., Volk, M.: Model-based safety analysis for vehicle guidance systems. In: Tonetta, S., Schoitsch, E., Bitsch, F. (eds.) SAFECOMP 2017. LNCS, vol. 10488, pp. 3–19. Springer, Cham (2017). https://doi.org/10.1007/978-3-319-66266-4_1
16. Ahmad, W., Hasan, O.: Formalization of fault trees in higher-order logic: a deep embedding approach. In: Fränzle, M., Kapur, D., Zhan, N. (eds.) SETTA 2016. LNCS, vol. 9984, pp. 264–279. Springer, Cham (2016). https://doi.org/10.1007/978-3-319-47677-3_17
17. Elderhalli, Y.: DFT Formal Analysis: HOL4 Script and Storm Benchmarks (2017). http://hvg.ece.concordia.ca/Publications/TECH_REP/DFT_TR17
18. Elderhalli, Y., Hasan, O., Ahmad, W., Tahar, S.: Dynamic Fault Trees Analysis using an Integration of Theorem Proving and Model Checking. Technical report, Concordia University, Canada (2017). https://arxiv.org/abs/1712.02872