Formal Fault Tree Analysis of State Transition Systems

Jianwen Xiang[†], Kazuhiro Ogata[†,‡], Kokichi Futatsugi[†]

[†] Japan Advanced Institute of Science and Technology (JAIST)
Graduate School of Information Science
1-1 Asahidai, Nomi, Ishikawa 923-1292, Japan

[‡] NEC Software Hokuriku, Ltd.
1 Anyoji, Tsurugi, Ishikawa 920-2141, Japan

{jxiang, ogata, kokichi}@jaist.ac.jp

Abstract

Fault Tree Analysis (FTA) is a traditional deductive safety analysis technique that is applied during the system design stage. However, traditional FTA does not consider transitions between states, and it is difficult to decompose complex system fault events that are composed of multiple normal components' states rather than individual component failures. To solve these problems, we first propose two different fault events of fault trees, and then present a formal fault tree construction model by introducing the concept of transition rules for event decomposition, in which the semantics of gates and minimal cut sets of fault trees are revised compared with traditional FTA.

1. Introduction

Fault Tree Analysis was first developed in the 1961 to facilitate analysis of the Minuteman missile system [14] and has been widely used in the aerospace, electronics, and nuclear industries for safety and reliability analyses for years. FTA was originally developed for hardware system analysis, and the fault trees were often constructed after a detailed system operation diagram had been provided, i.e., after the system design stage. In this case, with respect to a specific system fault (undesired hazard), we can easily attribute it to some component (or sub-system) failures straightforward, and it will not cause any incorrectness problem of the fault trees supposing the operation diagram itself is correct and consistent.

However, with respect to software and integrated complex system analysis, the situation changed. And the approach of carrying out system safety analysis with fault trees before the system design stage is usually more desirable, since it can provide useful information and concrete safety requirements for the subsequent system design and implementation. But in this case, an inevitable problem is how to ensure the generated informal fault trees are correct, i.e., the sub-events can formally result in their top event with respect to each logic gate of the fault trees. This is because the fault trees are usually constructed by intuition and incomplete information at this stage.

To solve this informal and incorrect problem, several formal fault tree models have been developed. Examples of them can be found in [5] (Kirsten M. Hansen and Anders P. Ravn, 1998), [3] (David Coppit, Kevin J. Sullivan, and J. B. Dugan, 2000), and [12] (Gerhard Schellhorn, Andreas Thums and Wolfgang Reif, 2002). But these methods mainly focused on providing formal semantics for fault tree constructors, such as different logic gates; they seldom considered how to formally construct the fault tree in a deductive manner. The common method is to develop the formal model and the fault trees as separate documents. That is to say, building up the fault trees is driven by intuition, while the events and sub-events of a gate are formalized afterward with respect to the formal model [11]. This approach is effective for quickly constructing a fault tree, but the informal construction creates problems later, when verifying its correctness.

A related issue is that, transitions between states are not represented in traditional fault trees [7], and thus it is difficult to resolve a system fault which consists of event combination (several normal object states) rather than depending on individual component failures. And in a number of formal approaches of FTA [5, 6], events are just decomposed according to the structure of the formula describing the event. This is especially a problem in distributed systems which are usually modeled as transition systems. For example, considering a railway level crossing control system, a hazard collision is defined as when a train is on the crossing, the barriers are open, namely $Collision \overset{def}{=} OnCrossing(tr) \wedge Open(ba)$. The hazard collision consists of two normal object states, and we can not simply decompose it into two sub-events as $Crossing(tr)$ and $Open(ba)$, since neither of them is a fault independently and nobody would like to build a level crossing to ensure that a train never passes the crossing or the barriers are never open. Here, one important principle for developing proper fault trees should be stressed, such as figured out in the fault tree handbook [13]: "all events/nodes that are linked together on a fault tree should be written as faults except, those statements that are added as simply remarks (e.g., conditioning events)".

This problem also stems from that traditional FTA mainly focuses on fault occurrence rather than fault existence as for qualitative analysis of fault trees, and thus all the fault events (states) are considered as nonrepairable (unchangeable) [13]. This principle is useful for hardware system analysis, but it may cause troubles when analyzing transition systems which consist of several objects as we figured out above. In [11], Reif, et al. noticed this problem and proposed to "define events and sub-events of a gate separately, and to check their correct interrelation explicitly", but they did not discuss how to derive the sub-events in an instructive formal way.

Therefore, in the work described here, based on our previous work [16], we further propose a formal fault tree analysis model based on classical propositional logic and basic concepts of state transition. In this model, the semantics of events, gates, and minimal cut sets of fault trees are revised and augmented compared with standard FTA, and a formal fault tree construction procedure is proposed to deal with the decomposition problem of complex system states. The example of a distributed radio-based railroad crossing control system will be used to explain our approach.

The rest of this paper is organized as follows. Section 2 provides background material on FTA. Section 3 elaborates our formal fault tree analysis of state transition, which consists of the discussion of fault events in fault trees, analysis of transition rules for event decomposition, notation and semantics of gates, and the formal fault tree construction model. We conclude with a case study and a discussion in Section 4 and 5, respectively.

2. Fault Tree Analysis

Fault tree analysis [13] is a deductive safety analysis technique which is applied during the design phase. The technique was first developed in the 1960s to facilitate analysis of the Minuteman missile system [14] and has been supported by a rich body of research since its inception. It is a top-down approach; input consists of knowledge of the system's functions as well as its failure modes and their effects. The result of the analysis is a set of combinations of component failures that can result in a specific malfunction. The approach is graphical, constructing fault trees using standardized symbols [13], as shown in Figure 1. There are several variations and extensions, but in this article we limit ourselves to the following symbols.

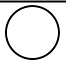
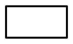
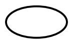
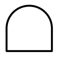
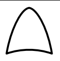
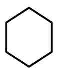
	BASIC EVENT – A basic initiating fault requiring no further development.
	INTERMEDIATE EVENT – An event that results from a combination of events through a logic gate.
	CONDITIONING EVENT – Specific conditions or restrictions that apply to any logic gates.
	AND gate – The output fault occurs only when all the input children faults occur.
	OR gate – The output fault occurs only when one or more the input children faults occur.
	INHIBIT gate – Output fault occurs if the (single) input occurs in the presence of an enabling condition (the enabling condition is represented by a CONDITIONING EVENT connected to the gate)

Figure 1. Fault tree symbols

The goal of the analysis is to find all the minimal cut sets, where a *minimal cut set* is a smallest combination (intersection) of component failures (basic fault events) which, if they all occur, will cause the top event to occur. The combination is a "smallest" combination in that all the failures are needed for the top event to occur; if one of the failures in the cut set does not occur, then the top event will not occur (by this combination) [13].

Essentially, fault tree analysis is a qualitative model which reveals the possible combinations of identified basic events sufficient to cause the undesired top event. But it is also often used in probabilistic analyses, such as failure rate calculation of the top event (given the failure rates of basic fault events) [8] and allocation of software reliability [15]. In this paper we are not concerned with the probabilistic extensions to our model.

3. Formal FTA of State Transition Systems

3.1. Definition of events

In traditional fault tree analysis terminology, an event in a fault tree except the conditioning events is called *fault* event, meaning the fault occurrence of *one* specific system or component state [13, 2]. The event can be understood as a root node, intermediate node (intermediate event), or leaf (basic event) in a fault tree, depending on the event is classified as “state-of-system” or “state-of-component” [13].

But in the sense of transition systems, a fault event often refers to a conjunction of *several normal* object (subsystem) states, such as the event $Collision$ in the crossing control system discussed in the last section. In this case, an undesired system state which consists of a combination of several normal object states can not be simply decomposed into several sub-events directly, just according to the structure of the formula describing the event. And generally speaking, it is also difficult to attribute it to some independent component failures directly.

To avoid such misunderstanding and solve the decomposition problem, we classify the fault event into the following two cases:

- Single Fault Event: occurrence of a *single fault* system or object state, which can be represented by a state predicate, p.
- Conjunct Fault Event: occurrence of a conjunction of several *normal* object states, e.g., $p \wedge q$ (p and q are state predicates), in which neither p nor q is a fault event independently, but their simultaneous occurrence constitutes an undesired system state. In other words, we can understand it as p occurs in a wrong time when q holds, and vice versa.

The importance to distinguish the conjunct fault event and single fault event is that, with respect to distributed system analysis, a system state usually consists of several object states which are changed by transitions; and thus an undesired system state may consist of several object states which are not fault events independently. Such as the above example, a train on crossing and the barriers are open are not fault events with respect to the train and the barriers, respectively. Without such distinction, it is difficult to resolve a conjunct fault event correctly using traditional FTA as we discussed above.

3.2. Analysis of transition rules

In our formal fault tree analysis, we define an important term in our model, namely *transition rule*.

Definition 3.1 (Transition Rule) *Give an occurrence of a system or object state c, a transition rule states all the immediate, sufficient, and necessary causes for c.*

A standard transition rule can be interpreted by the following formula:

$$A_1 \vee A_2 \vee \ldots \vee A_n \Rightarrow c \tag{1}$$

where A_i ($i = 1, 2, \ldots, n$) is a conjunction of predicates which will result in the occurrence of c, and $\Rightarrow$ denotes a one-step transition or rewrite relation in a sense of rewriting logic.

This is similar to the concept of *transition* defined in standard transition system models [9]. One difference is that from the point of view of fault tree analysis, it stresses that we should consider *all* the possibilities (including physical environment faults such as hardware defects, human errors, and so on) with respect to the occurrence of c, not only limited to the software-to-be or system design. From this standpoint, a transition rule in FTA can also be regarded as a union of several transitions which result in the same occurrence of a object state in standard transition systems.

Before discussing how to decompose the conjunct fault event with the transition rule (the decomposition of the single fault event can be regarded as a simple instantiation of the conjunct fault event, and in the following discussions we focus our attention on the conjunct fault event), one important *constraint* of the transition rule should be introduced as follows.

Give a conjunct fault event that consists of two object state predicate p and q^α (we use q^α to denote that the object q is in its state α, and use q^β to denote any other different state of q except q^α. This is because in practice, an object may have several different states, and only using q and $\neg q$ may cause misunderstanding and confusing in the following discussions) , and suppose a transition rule for the occurrence of p is in the form of:

$$A_1 \vee A_2 \vee \ldots \vee A_n \Rightarrow p$$

Then for each A_i ($i = 1, 2, \ldots, n$), it can *not* cause any occurrence of q^β, where $q^\alpha \neq q^\beta$.

The reason is obvious. If $A_i \Rightarrow q^\beta$, then there is a contradiction between the transition rule and the conjunct fault event, and it is impossible to derive the previous system state (predecessor) of the conjunct fault event. And if A_i violates the constraint, we should remove A_i from the transition rule. In case no A_i complies with the constraint, then we should consider another transition rule for q^α instead of p.

More specifically, the constraint can be classified into the following three sub-cases depending on whether A_i contains q^α or q^β.

1. If A_i contains q^α, say $A_i = C \wedge q^\alpha$, where C is a conjunction containing no state predicate of q, then the formula $C \wedge q^\alpha \Rightarrow q^\alpha$ must hold (some references defined it as a kind of 'idling transition' [9], but here we call it as a formula rather than a transition rule because in the sense of FTA, it does not cause the occurrence of q^α, and it just states that even with condition C, q^α will not be changed).

2. If A_i contains q^β, say $A_i = C \wedge q^\beta$, the the transition rule $C \wedge q^\beta \Rightarrow q^\alpha$ must also hold.

3. If A_i contains no state predicate of q, say $A_i = C$, then the above two sub-cases may either or both hold. However, with respect to the second one, i.e., $C \wedge q^\beta \Rightarrow q^\alpha$, it is difficult for us to identify q^β and conform the above transition rule since there is no knowledge about q^β in A_i at this moment. And actually, this transition rule can be covered afterward when analyzing the occurrence of q^α. To this end, we only require that the formula $C \wedge q^\alpha \Rightarrow q^\alpha$ holds with respect to this subcase.

Based on the above analysis, we present the *patterns of sub-events* to the conjunct fault event, $p \wedge q^\alpha$, as follows.

1. If A_i contains no state predicate of the object q, then the sub-event is: $A_i \wedge q^\alpha$.

2. If A_i contains either q^α or q^β, then the sub-event is: A_i.

The proof of the above patterns is straightforward based on the discussion of constraint of transition rules as follows.

Pattern-1: Since $A_i \Rightarrow p$ and $A_i \wedge q^\alpha \Rightarrow q^\alpha$, then $A_i \wedge q^\alpha \Rightarrow p \wedge q^\alpha$, and the sub-event is $A_i \wedge q^\alpha$.

Pattern-2: Since $A_i \Rightarrow p$ and $A_i \Rightarrow q^\alpha$, then $A_i \Rightarrow p \wedge q^\alpha$, and the sub-event is A_i.

3.3. Notation and semantics of gates

After discussing the fault events and transition rules for formal fault tree construction, we should consider how to represent the transition rule and the event as well as its sub-events connected by a logic gate in a proper formal way. The semantics of gates should also be defined as for fault tree analysis and system safety analysis.

First of all, as for the decomposition of conjunct fault events, the corresponding transition rule should be recorded in the fault trees; otherwise it may cause misunderstanding or troubles to check the correctness of the fault tree when it is reviewed by other analyzers. To this end, we propose to *explicitly* label (or record) the transition rule in (or besides) the corresponding logic gate for further reviewing and rechecking.

	SC	NC
T-AND gate	$I_1 \wedge I_2 \rightarrow O'$	$O' \rightarrow I_1 \wedge I_2$
T-OR gate	$I_1 \vee I_2 \rightarrow O'$	$O' \rightarrow I_1 \vee I_2$
T-INHIBIT gate	$I_1 \wedge C \rightarrow O'$	$O' \rightarrow I_1$
AND gate	$I_1 \wedge I_2 \rightarrow O$	$O \rightarrow I_1 \wedge I_2$
OR gate	$I_1 \vee I_2 \rightarrow O$	$O \rightarrow I_1 \vee I_2$
INHIBIT gate	$I_1 \wedge C \rightarrow O$	$O \rightarrow I_1$

Figure 2. Semantics of gates

A gate labeled with a transition rule should be distinguished from the standard gates of FTA, in which the semantics of gates are just defined in a simple form of "$Output = Inputs$" based on Boolean algebra. With respect to the standard gates, there is no state transition (time-delay) between the input events and output event, such as the standard OR-gate can be understood as a kind of fault space (event) splitting (division), and the input events are just defined as (more concrete) restatements of the output event [13]. To this end, we propose that a gate labeled with a transition rule is a *transition gate* (short: T-gate), otherwise it is a standard gate. Both of these two kinds of gates are used in our formal fault trees.

Two important concepts for defining the formal semantics of gate are the sufficient (correct) and necessary (complete) conditions. The *sufficient condition* (SC) states that if the input events occur, the output event must occur also. The *necessary condition* (NC) states that the output event must not happen without the input *fault* events. These two concepts are also useful for the classification and calculation of minimal cut sets that we will discuss later.

Suppose there are two input fault events I_1 and I_2, an output fault event O, and a conditioning (normal) event C, the formal semantics of the T-gates as well as the standard gates are listed in Figure 2 (for simplification, we use the symbol $'$ to denote the successor state).

It should be noted that in Figure 2, with respect to the conditioning event C in the T-INHIBIT and INHIBIT gates, we do not consider it in the necessary conditions of the gates. This is because the main goal of FTA is to find the primary fault events; and to ensure the top undesired event never happens, we usually focus on how to prevent the primary fault events rather than those normal (conditioning) events. This point is also illustrated by the definition of necessary condition of gate.

Based on Figure 2 and the above discussions, we further propose two kinds of minimal cut sets of fault trees as follows.

Sufficient Minimal Cut Set : A sufficient minimal cut set `SMCS` is a smallest combination of primary events (including conditioning events) which, if they all occur, will cause the top event `T` to occur, namely `SMCS` $\rightarrow$ `T`

Necessary Minimal Cut Set : A necessary minimal cut set NMCS is a smallest combination of only primary *fault* events which, if the top event occurs, then they must also occur, namely $\mathtt{T} \rightarrow \mathtt{NMCS}$.

The sufficient minimal cut sets can help us understand under which conditions, the primary fault events will result in the top hazard event, and thus makes the analysis more comprehensively. In contrast, because of $\neg\mathtt{NMCS} \rightarrow \neg\mathtt{T}$, the necessary minimal cut sets can help us focus on and derive more manageable and concrete sub safety requirements as for system safety analysis. Both of them complement each other and can be calculated by the sufficient and necessary conditions of gates defined in Figure 2, respectively.

It should be noted that in standard FTA, the definition of minimal cut set is a combination of primary fault events 'sufficient' for the top event [13]. This definition seems no problem if we limit ourselves to only AND and OR gates for analysis such as discussed in [13]. However, in case a fault tree consists of conditioning events and some other gates, e.g., INHIBIT gate, such a definition is not precise enough and may cause confusion and contradiction.

3.4. Formal fault tree construction model

Based on the above discussion and analysis, we present our formal fault tree construction model below.

We take a catastrophic failure as the root node of the fault tree, namely R. Assume R is a conjunct fault event consisting of two state predicates, i.e., $p \wedge q^{\alpha}$, where q^{α} denotes that object q is in its α state as we defined in Section 3.2, the regression procedure for the formal fault tree construction is as follows (in case R is a single fault event consisting of only one state predicate p, the regression procedure can follow the traditional fault tree construction fundamentals [13], and it can be regarded as a simple instantiation of the following procedure).

Initial step Define the formal specification of $R = p \wedge q^{\alpha}$.

Deductive step Let $A_1 \vee \ldots \vee A_n \Rightarrow p$ be the transition rule selected, and A_i $(i = 1, \ldots, n)$ complies with the constraint, that is, $A_i \nRightarrow q^{\beta}$, where q^{β} denotes any other state of object q except q^{α}.

- For each A_i, if A_i does not contain q^{α} or any q^{β}, then $M_i := A_i \wedge q^{\alpha}$, else $M_i := A_i$, where M_i is a intermediate variable.
- $R := M_1 \vee M_2 \vee \ldots \vee M_n$

Iteration step

(1) Decompose the resulting R to some sub-events by an *appropriate* logic gate or edge;

(2) Integrate and record the corresponding transition rule into the logic gate or edge for further revising;

(3) Then for each sub-event, redo the inductive and iteration steps recursively until a basic event or the chosen abstraction level is reached.

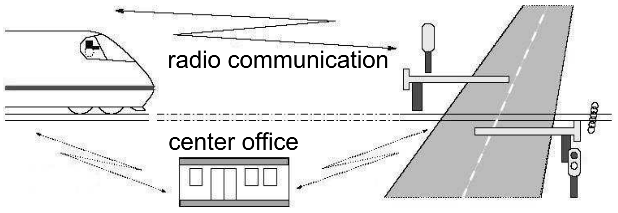


Figure 3. An railway crossing control system

To help engineers construct the fault tree more efficiently, we present two important general guidelines for selecting the "appropriate logic gate or edge" in iteration step (1) as follows.

- If the antecedent of the transition rule consists of more than one conjunctions or causes, i.e., $i \geq 2$, then a T-OR gate should be introduced.
- Otherwise the resulting R is still a conjunction, and there are three possibilities to decompose the conjunction and connect the corresponding sub-events as follows.
  - If the resulting R can be decomposed into several meaningful fault events completely, then a T-AND gate should be used.
  - If the resulting R can be identified as one or several fault events together with some normal (conditioning) events, then a T-INHIBIT gate should be introduced.
  - If the resulting R can only be understood as one conjunct fault event, then we should use an edge labeled with the corresponding transition rule to connect the sub-event.

4. Example

In this section, we illustrate the formal fault tree analysis of state transition by analyzing the hazards of a railway crossing control system [1]. An overview of this system is given in Figure 3, and its brief informal description is as follows [12, 11].

Shortly before the train approaches the level crossing, it sends a 'secure' signal to the level crossing in order to check the status of the crossing. When the level crossing receives the command 'secure', it switches on the traffic lights, and then closes the barriers. After they have been closed, the level crossing is safe for a certain period of time and a 'release' signal is sent to the train, which indicates that the train may pass the crossing. After the train has passed the crossing, it sends back a 'passed' signal, which allows the crossing to open the barriers and switch back to its initial state. If no signal is received, the crossing waits for some minutes and then opens the barriers to protect cars against endless waiting (and is then unsafe).

In the crossing control system, the main task is to prevent the collision event, i.e., when a train is on crossing, but the barriers are open. The root node **R** of the fault tree is then formally defined as the following conjunct fault event:

R : $OnCrossing(tr) \wedge Open(ba)$

where tr and ba represent the train and barriers, respectively.

Focused on the predicate "$OnCrossing(tr)$" to analyze which conditions will cause a train on crossing, we derive the following transition rule:

$\mathbf{T}_1$: $(BypassSignal \vee BrakeFailure \vee Release(tr)) \Rightarrow OnCrossing(tr)$

which states that either the driver bypasses the stop signal (illegal driving or human misoperation), or the brake of the train fails (hardware failure), or the train has got a permission – 'release' signal (system design) will cause the train on crossing. This transition rule complies with the constraint since none of these three causes will not change the state of barriers, therefore we get three corresponding sub-events connected by an T-OR gate as follows.

$\mathbf{S}_{1a}$: $BypassSignal \wedge Open(ba)$

$\mathbf{S}_{1b}$: $BrakeFailure \wedge Open(ba)$

$\mathbf{S}_{1c}$: $Release(tr) \wedge Open(ba)$

Here, we regard $BypassSignal$ and $BrakesFailure$ as basic fault events, and since $Open(ba)$ itself is a normal event, an INHIBIT-gate connected with the conditioning event $Open(ba)$ can be introduced to further simplify both $\mathbf{S}_{1a}$ and $\mathbf{S}_{1b}$ (shown in Figure 4). The corresponding events are defined as follows.

$\mathbf{B}_1$: $BypassSignal$

$\mathbf{B}_2$: $BrakeFailure$

$\mathbf{C}_1$: $Open(ba)$

Then focused on $Release(tr)$ of $\mathbf{S}_{1c}$, we derive another transition rule which states that, the causes for a train to get a 'release' signal are the level crossing has sent this signal and there is no radio communication failure between them.

$\mathbf{T}_2$: $Release(cr) \wedge \neg RadioFailure \Rightarrow Release(tr)$

where cr represents the level crossing.

Regressing $\mathbf{S}_{1c}$ with $\mathbf{T}_2$, we derive a sub-event $\mathbf{S}_2$ and a conditioning event $\mathbf{C}_2$ which are connected with an INHIBIT gate as follows.

$\mathbf{S}_2$: $Release(cr) \wedge Open(ba)$

$\mathbf{C}_2$: $\neg RadioFailure$

Notice here we regard $\mathbf{C}_2$ as a conditioning event because it is a normal event, and $\mathbf{S}_2$ can be identified as a fault event regardless of $\mathbf{C}_2$.

Focused on $Release(cr)$ of $\mathbf{S}_2$, we derive another transition rule, that is, only after the crossing has *confirmed* (accepted) a 'secure' request from a train, it can send a 'release' signal to the train. The transition rules is defined as follows.

$\mathbf{T}_3$: $Secure(cr) \Rightarrow Release(cr)$

And we derive the corresponding sub-event as:

$\mathbf{S}_3$: $Secure(cr) \wedge Open(ba)$

Notice here we use a edge to connect $\mathbf{S}_2$ and $\mathbf{S}_3$ since $\mathbf{S}_3$ can not be further divided into smaller ones. In this case, $\mathbf{T}_3$ should also be labeled besides the edge for revising (see Figure 4).

Further analysis will try to regress $Secure(cr)$, and we would get a transition as follows, that is, when the crossing receives a 'secure' request from a train and *the barriers are open*, the crossing will accept this request (notice here for simplicity, $Open(ba)$ is regarded as shared variable to ensure that there is no train on the crossing at this moment).

$$Secure(tr) \wedge \neg RadioFailure \wedge Open(ba) \Rightarrow Secure(cr)$$

However, based on the system design knowledge, we know that once the crossing has accepted a 'secure' request, then it must close the barriers until it gets a 'passed' signal from the train. In other words, this transition rule *violates* the constraint $A \not\Rightarrow q^{\beta}$. Therefore, we should focus on $Open(ba)$ and derive another transition rule as follows.

$\mathbf{T}_4$: $(TimeOut \vee Passed(cr)) \Rightarrow Open(ba)$

$\mathbf{T}_4$ states that two possibilities for the barriers to open are either a time-out event (the crossing has been waiting for the '*passed*' signal over a designed time d, and then opens the barriers to protect cars against endless waiting) occurs, or the crossing has just received a '*passed*' signal from a train. Use this transition rule to regress $\mathbf{S}_3$, we get two corresponding sub-events connected by an T-OR gate below.

$\mathbf{S}_{4a}$: $Secure(cr) \wedge TimeOut$

$\mathbf{S}_{4b}$: $Secure(cr) \wedge Passed(cr)$

Keep doing analysis in the similar way, finally we derive all the transition rules and events as follows, and the entire fault tree is shown in Figure 4.

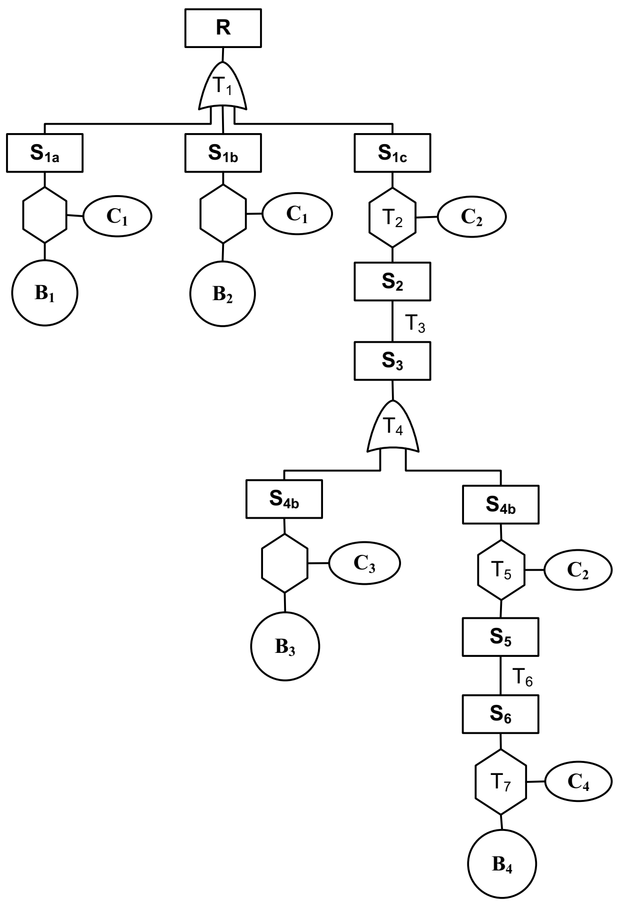


Figure 4. Formal fault tree of collision

C$_3$: $Secure(cr)$

B$_3$: $TimeOut$

T$_5$: $Passed(tr) \wedge \neg RadioFailure \Rightarrow Passed(cr)$

S$_5$: $Passed(tr) \wedge Secure(cr)$

T$_6$: $OnCrossing(tr) \Rightarrow Passed(cr)$

S$_6$: $OnCrossing(tr) \wedge Secure(cr)$

T$_7$: $Secure(tr) \wedge \neg RadioFailure \wedge Open(ba) \Rightarrow Secure(cr)$

C$_4$: $Secure(tr) \wedge \neg RadioFailure$

B$_4$: $Open(ba) \wedge OnCrossing(tr)$

We regard $TimeOut$ as a basic fault event because to solve it need human intervention rather than system (software) design. And from Figure 4, several important issues are disclosed as follows.

- Some *mutual exclusion* problems have been discovered , such as the event **S**$_6$, which states that when a train on the crossing, the crossing can not responds a ‘secure’ request from another train.
- From **R** to **B**$_4$, an *mutual dependency* between the fault events has been discovered. In contrast, the corresponding safety properties (the negations of the fault events) are also mutual dependent, which has been formally verified in our previous work [16].
- There are four minimal cut sets in the fault tree, i.e., **B**$_1$, **B**$_2$, **B**$_3$, and **B**$_4$, each of them is a basic fault event. Therefore, to ensure the system is safety, we need only focus our attention on these basic fault events; while to understand what will cause the collision hazard more comprehensively, we can deduce the sufficient minimal cut sets, in which the conditioning events should be taken into account, such as the sufficient minimal cut set $BrakeFailure \wedge Open(ba)$.

5. Concluding Remarks

In this paper we have presented a formal fault tree analysis of state transition, and our main technical contributions can be summarized as follows.

- We extend traditional FTA for transition system analysis, which is supported by our augmented and refined formal semantics of fault tree constructors (events, gates, and minimal cut sets), and the introduction of transition rules for event decomposition and fault tree construction.
- The correctness of our formal fault tree is guaranteed by the construction process itself, thus avoiding the problems that often arise with traditional methods. At the same time, it gives domain experts the ability to discover transition rules, design principles, and hidden relationship between the fault events in a stepwise and instructive way, which are also useful and important knowledge for the subsequent formal system specification and verification;

We realize that one important advantage of FTA is the ease with which the trees can be read and understood and thus reviewed by experts and used by designers [7]. This is one reason that we further develop our formal fault tree analysis based on basic concepts of state transition instead of temporal logic as we proposed in our previous work [16].

Another more important motivation is that currently, we are focusing on how to formally model, specify, and verify different systems and applications more efficiently and effectively with OTS/CafeOBJ (observational transition system) [4, 10], therefore to propose a formal FTA based on the common framework — state transition systems or OTSs, may make the combination of these two techniques more consistent, i.e., the results of system safety analysis (FTA) can be used directly for formal system specification and verification with OTS/CafeOBJ, and in reverse, formal system specification can help us develop more reliable fault trees. This is also one of our future works.

It should be noted that our work has not yet been applied in full scale industrial practice, although it has been demonstrated by one case in this paper. Our next step is trying to find more (big scale) case studies to further improve our study.

6. Acknowledgement

This research is conducted as a program for the ”Fostering Talent in Emergent Research Fields” in Special Coordination Funds for Promoting Science and Technology by Ministry of Education, Culture, Sports, Science and Technology (MEXT, Japan).

References

[1] Betriebliches lastenheft für funkfahrbetrieb. stand 1.10, 1996.

[2] N. J. Bahr. *System Safety Engineering and Risk Assesment: A Practical Approach*, chapter Fault Tree Analysis. Taylor & Francis, 1997.

[3] D. Coppit, K. J. Sullivan, and J. B. Dugan. Formal semantics of models for computational engineering: A case study on dynamic fault trees. In *Proc. of The 11th International Symposium on Software Reliability Engineering*, pages 270–282, San Jose, California, USA, Oct 2000.

[4] K. Futatsugi, A. T. Nakagawa, and T. Tamai. *CAFE: an Industrial-Strength Algebraic Formal Method*. Elsevier Science, Amsterdam, 2000.

[5] K. M. Hansen and A. P. Ravn. From safety analysis to software requirement. *IEEE Transactions on Software Engineering*, 24(7):573–584, Jul 1998.

[6] A. Lankenau and O. Meyer. Formal methods in robotics: Fault tree based verification. In *Proc. of The Third International Software Quality Week Europe*, 1999.

[7] N. G. Leveson. *Safeware: System Safety and Computers*. Addison-Wesley Pub., Sep 1995.

[8] M. R. Lyu, editor. *Handbook of Software Reliability Engineering*. McGraw-Hill, 1995.

[9] Z. Manna and A. Pnueli. *The Temporal Logic of Reactive and Concurrent Systems*. Springer-Verlag, New York, 1992.

[10] K. Ogata and K. Futatsugi. Proof scores in the ots/cafeobj method. In *Proc. of The 6th IFIP WG6.1 International Conference on Formal Methods for Open Object-Based Distributed Systems (FMOODS 2003)*, volume 2884 of *LNCS*, pages 170–184. Springer, 2003.

[11] W. Reif, G. Schellhorn, and A. Thums. Safety analysis of a radio-based crossing control system using formal methods. In *Proc. 9th IFAC Symposium Control in Transportations Systems 2000*, pages 289–294, Braunschweig, Germany, 2000.

[12] G. Schellhorn, A. Thums, and W. Reif. Formal fault tree semantics. In *Proc. of The 6th World Conference on Integrated Design and Process Technology*, Pasadena, CA, 2002.

[13] W. E. Vesely, F. F. Goldberg, N. H. Roberts, and D. F. Haasl. Fault tree handbook. Technical Report NUREG-0492, U.S. Nuclear Regulatory Commission, Washington, D.C, Jan 1981.

[14] H. A. Watson and B. T. Laboratories. Launch control safety study. Technical report, Bell Telephone Laboratories, Murray Hill, NJ, 1961.

[15] J. Xiang, K. Futatsugi, and Y. He. Fault tree analysis of software reliability allocation. In *Proc. of The 7th World Multiconference on Systemics, Cybernetics and Informatics*, volume Volume II - Computer Science and Engineering, pages 460–465, Orlando, USA, Jul 2003.

[16] J. Xiang, K. Futatsugi, and Y. He. Fault tree and formal methods in system safety analysis. In *Proc. of The 4th International Conference on Computer and Information Technology*, pages 1108–1115, Wuhan, China, Sep 2004. IEEE.