A New Methodology for Automatic Fault Tree Construction based on Component and Mark Libraries

Mr. Ashish Bhagavatula[1], Dr. Jun Tao[2], Dr. Sarah Dunnett[1], Dr. Paul Bell[3]
[1]Department of Aeronautical and Automotive Engineering,
Loughborough University, Loughborough LE11 3TU, UK
[2]School of Automobile and Traffic Engineering,
Wuhan University of Science and Technology, Wuhan 430081, China
[3]Department of Computer Science,
Loughborough University, Loughborough LE11 3TU, UK

Abstract

During the design stage of the development of a new system, automated fault tree construction would produce results a lot sooner than the manual process and hence be highly beneficial in order to modify the system design based on identified weakest areas. Although much work has been performed in this area, the construction of fault trees is still generally done manually. In this paper, a new methodology of constructing fault trees from a system description is proposed. Multi-state input/output tables are introduced, which have the capability to capture output deviations during the normal operation of a component as well as under the influence of abnormality or failure. Two libraries, namely a component library and a mark library, are introduced. The former stores component models and the latter stores a range of marks. The main purpose of a mark is to identify a certain feature of the system, such as a feedback loop or multiple redundancies. These two libraries are used to redraw the system in a graphical environment where the designer can witness the system come together and also input the necessary failure data for each component. An algorithm has been developed, that uses input/output tables and marks, to automatically construct fault trees for failure modes of interest.

In order to demonstrate this methodology, it is applied to an automotive emission control system, and a fault tree is generated using the algorithm developed in this work.

1. Introduction

With increase in complexity of engineering systems day by day, the necessity to identify hazards and control them in time is ever increasing. In today's world, it is a must for any new system to be thoroughly analysed for potential hazards, preferably in the early design stages before progressing in the product life cycle. Currently, this is not dealt with in the most efficient manner. During the system design phase, once an initial system design is completed, it is handed over to the Risk and Reliability Department who then manually analyse the system and generate an appropriate reliability model. Often, within the time they get back to the design team with proposed design improvements, the system design has already progressed a long way. At this point, the design team either make the changes which prove expensive, or carry on with the existing design. In the latter case, the risk of a hazardous failure during operation increases and so does the probability of uncertain

failures. It is this gap which demands an automated reliability model generator to avoid the time lapse between the design team and the risk and reliability team. In this paper, Fault Tree Analysis (FTA) has been selected as the appropriate reliability model

FTA has been identified as one of the most commonly employed reliability models in the aeronautical and automotive industry. FTA is a deductive approach to determine the various combinations of hardware failures, software failures and human errors that could lead to undesirable events at a system level. A certain system failure is selected as the top event for which a fault tree is generated. This tree combines basic failures with the help of logic gates in order to present all possible combinations that could lead to the top event. [1]

The construction of fault trees can be time consuming and tedious depending on the complexity of the system. As the task is done manually, it is also prone to errors and misinterpretations. There have been several attempts in the past to explore the requirement of automatic fault tree generation. The most ground breaking of methodologies are considered to be the digraph technique [2] and the decision table method [3]. The digraph technique uses deviations to model failures which allow more freedom to describe different types of failures. The decision tables can take into account a larger number of failure modes with clarity due to its tabular format.

Besides these, commercial software has been developed to help automate the construction process. For example, a program called HiP-HOPS was developed by University of Hull and launched in 2012 [4]. This software has been integrated with Mathworks Matlab and SimulationX in order to be able to extract information from the system model directly. Hip-HOPs requires the user to specify failure modes for each component as well as failure expressions which link the failure modes and inputs to their respective outputs. This requirement gives the user flexibility to define the behaviour of system components but at the same time can be time consuming for the user. This program definitely aids the user to build a fault tree but it is believed that the automation can be done to a greater extent by using predefined generic component tables and mark operators which will be explained in Section 4.

The very first requirement of automatic fault tree generation is defining the system in a way that bridges the semantic gap. Various approaches have been considered for this stage such as system description in the form of Computer-Aided Design (CAD), Process and Instrumentation Diagrams (P&ID), Mathworks Simulink or even a personalised Graphic User Interface (GUI). Lately, System Modelling Language (SysML) has gained a lot of importance as it has a vast number of features to describe the system, by taking into account system structure as well as behaviour. In this paper, such specifics of system description haven’t been discussed, but are being explored for future work in this field.

2. Overview of Methodology

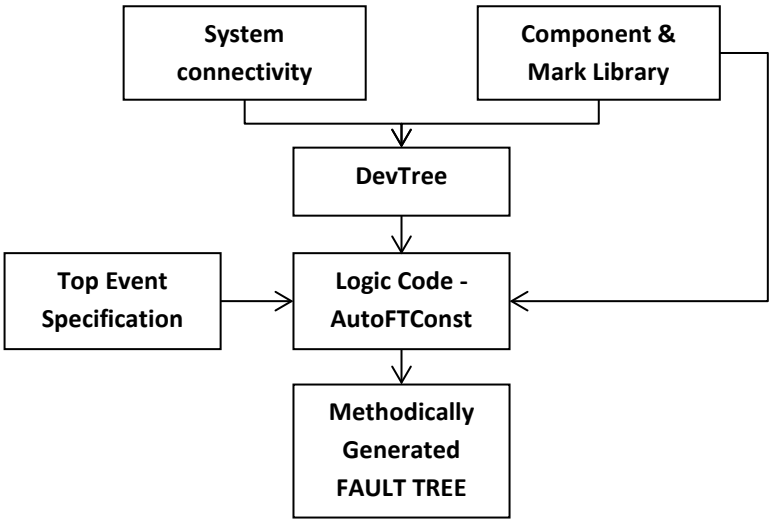


Figure 1 - Overview of methodology

The methodology is explained with the help of a flow diagram as can be seen in Figure 1. Each block has been numbered and is expanded as follows:-

(1) System Connectivity: List of system components, inputs and outputs for each component.

(2) Component and Mark Library: The component library stores a multi-state Input / Output (I/O) table for each component. The mark library stores a range of marks to identify complex features in a system such as loops or redundancies.

(3) Deviation Tree (DevTree): It is an intermediate representation of the system which can be used to generate fault trees automatically for any given top event.

(4) Top Event Specification: Critical system level failures for which a fault tree needs to be generated can be specified here. The format in which the top event is specified can be in two different ways:-

- **a.** Specification by means of deviation in an output from a component
- **b.** Specification by means of a partial fault tree

In a system, the top event could either be failure of a certain component, abnormality at a certain point in the system or a combination of component failures. It is for this reason; the logic code must be able to accept the top event in different formats.

(5) Logic Code – AutoFTConst: The algorithm for automatic fault tree construction will be written in C++ or Java. It will include predefined rules to formulate logic gates, extract component tables and deal with marks. These predefined rules are expanded upon in Sections 4 and 5.

3. Example System Description

In order to demonstrate the new methodology, it is applied to an automotive emission control system. This system is adapted from the lambda closed loop control system and the changes made are explained in this section.

The original system can be seen in Figure 2, it comprises of 6 components which have been labelled [5]. The composition of mixture of fuel and air supplied to the engine is continuously maintained within the optimum deviation range by a closed loop control. Hence, the emissions must be measured for unburnt oxygen content, and the injected fuel quantity is immediately corrected based on this measurement. This measurement is sent from the emission flow sensor located in the exhaust pipe, to the controller [6]. The air flow sensor also continuously sends mass flow data to the controller. Both these measurements are sent in voltages and the controller in turn sends a voltage signal to the injection valve to correct its timing accordingly.

1 Air-flow sensor, 2 Engine, 3 Lambda sensor, 4 Catalytic converter, 5 Injection valves, 6 Lambda closed-loop control, U_S Sensor voltage, U_V Valve-actuation voltage, V_E Injection quantity.

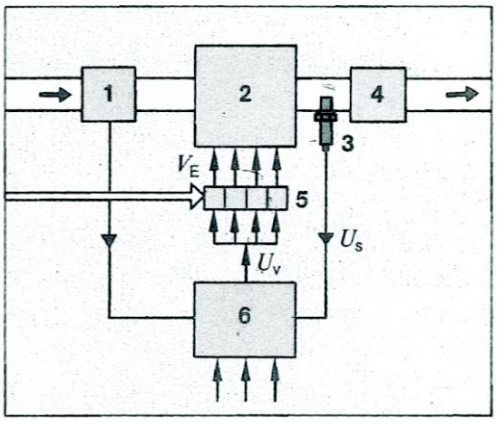
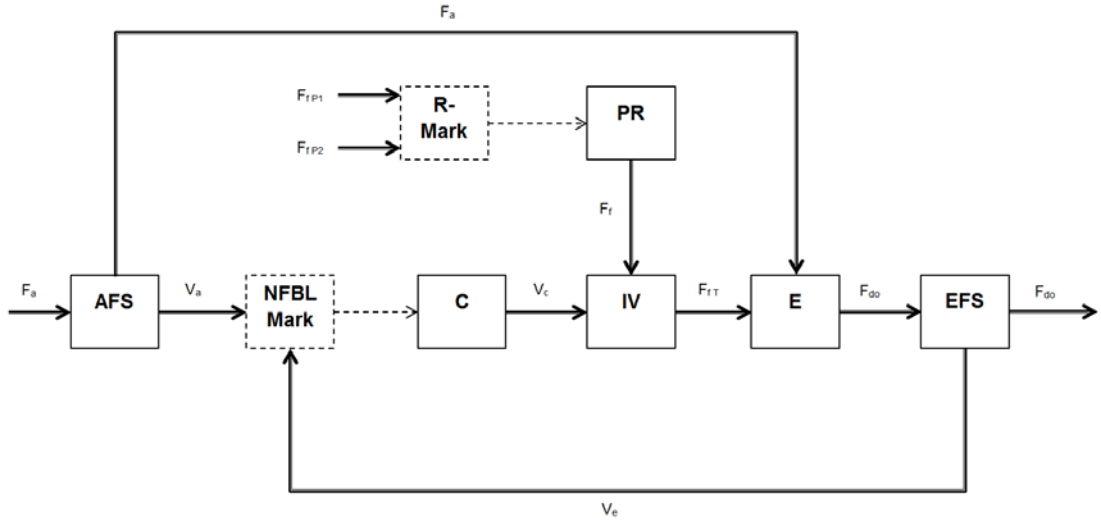


Figure 2 - Original Emission Control System [5]

Changes were made to the original system based on the following criteria: Requirement of a simple system with limited number of components, a negative feedback loop and a redundancy feature in order to demonstrate the concept of marks.

Paying attention to the above stated criteria, the following changes were made to the original system:-

1. Removal of the catalytic convertor as it is not part of the closed loop control.
2. Treating the injection valves as a single entity.
3. Addition of the pressure regulator which regulates fuel flow from two fuel pumps and supplies it to the injection valve.



AFS – Air Flow Sensor	C – Controller	IV – Injection Valve	NFBL Mark – Negative Feed-Back Loop Mark
EFS – Emission Flow Sensor	E – Engine	PR – Pressure Regulator	R-Mark – Redundancy Mark

Figure 3 - Adapted Emission Control System

3.1 Description of Variables – Mass flow, Voltage

The adapted system is drawn in the form of a block diagram and can be seen in Figure 3. Each variable is explained as follows:-

Mass Flow Rates:-

F_a: Intake Air
F_{fP1}: Fuel from Pump 1
F_{fP2}: Fuel from Pump 2
F_f: Regulated Fuel
F_{fT}: Fuel - timed injection
F_{do}: Emission flow; Subscript ‘do’ represents Density of unburnt Oxygen

Voltages:-

V_a: Voltage sent by AFS to C, notifying mass flow rate of intake air
V_e: Voltage sent by EFS to C, notifying amount of unburnt oxygen in the emission
V_c: Voltage sent by C to IV in order to time fuel injection into the engine

3.2 System Boundaries

1. Source of fuel is shown to come from two fuel pumps, each connected to a fuel tank. The fuel tanks and pumps are not part of the system as these components have been assumed to be failure-proof.
2. Power sources for the valves, sensors and the controller are not part of the system and this marks another boundary of the system under study.
3. Ordinarily, in an automotive engine, the intake air flow passes through an air filter before it goes through the air flow sensor and to the engine. In this system however, this component has been disregarded and is therefore considered failure-proof as well.

4. Application of Methodology to Example System

This section illustrates the procedure of systematic fault tree generation by applying it to the emission control system. The very first step is generating multi state I/O tables for each component in the system. Such tables can be generated for a variety of components that appear in aeronautical and automotive systems in order to enable reusability and reduce user effort during systematic fault tree construction.

4.1 Component Library

The component library stores multi-state I/O tables for each component. There are 6 components in the emission control system, hence 6 tables. The structure of this table is quite different to the traditional decision table structure [3]. These tables use deviations to represent failure modes which provide a higher degree of flexibility to represent the type of failure accurately. The tables are self-explanatory except for a few notations that are explained below:-

Variable Relationship: This states the relationship between the input and output in terms of direction of flow. ‘S’ implies single directional flow and ‘B’ implies bi-directional flow.
Deviation: Any variable in a system can deviate from its original state due to abnormality or failure. A deviation of ±10 implies uncontrollable disturbance and a deviation of ±1 implies controllable disturbance. ‘w’ implies whole range of deviation.
Transmit Coefficient: The relationship between the input and output based on proportionality. ‘1’ implies direct proportionality and ‘-1’ implies inverse proportionality. A transmit coefficient of ‘0’ implies that the failure is independent of the input.

Tables 1 to 6 represent the I/O tables for all the components in the emission control system and can be seen as follows:-

State	No.	Variable Relationship	Output	Transmit Coefficient	Input
Normal	1	S	$V_a(w)$	1	$F_a(w)$
	2	S	$F_a(w)$	1	$F_a(w)$
Failure Modes					
Measurement Error	1		$V_a(w)$		
Sensor Broken	2		$V_a(w)$	0	

Table 1 - Air Flow Sensor (AFS)

State	No.	Variable Relationship	Output	Transmit Coefficient	Input
Normal	1	S	$V_e(w)$	1	$F_{do}(w)$
	2	S	$F_{do}(w)$	1	$F_{do}(w)$
Failure Modes					
Measurement Error	1		$V_e(w)$		
Sensor Broken	2		$V_e(w)$	0	

Table 2 - Emission Flow Sensor (EFS)

State	No.	Variable Relationship	Output	Transmit Coefficient	Input
Normal	1	S	$V_c(w)$	1	$V_a(w)$
	2	S	$V_c(w)$	1	$V_e(w)$
Failure Modes					
Controller failure 1	1		$V_c(w)$		
Controller failure 2	2		$V_c(w)$	0	

Table 3 - Controller (C)

State	No.	Variable Relationship	Output	Transmit Coefficient	Input
Normal	1	S	$F_f(w)$	1	$F_{fp1}(w)$
	2	S	$F_f(w)$	1	$F_{fp2}(w)$
Failure Modes					
PR valve failed open	1		$F_f(-10)$		
PR valve failed closed	2		$F_f(+10)$		
PR valve stuck	3		$F_f(w)$	0	

Table 4 - Pressure Regulator (PR)

State	No.	Variable Relationship	Output	Transmit Coefficient	Input
Normal	1	S	$F_{f\,T}(w)$	1	$V_c(w)$
	2	S	$F_{f\,T}(w)$	1	$F_f(w)$
Failure Modes					
Timing Problem	1		$F_{f\,T}(w)$		
IV broken	2		$F_{f\,T}(w)$	0	

Table 5 - Injection Valve (IV)

State	No.	Variable Relationship	Output	Transmit Coefficient	Input
Normal	1	S	$F_{do}(w)$	1	$F_a(w)$
	2	S	$F_{do}(w)$	-1	$F_{f\,T}(w)$
Failure Modes					
Engine failure 1	1		$F_{do}(+10)$		
Engine failure 2	2		$F_{do}(-10)$		
Engine failure 3	3		$F_{do}(w)$		

Table 6 - Engine (E)

In Table 3, 'Controller failure 1' refers to an error in the voltage sent to the injection valve. 'Controller failure 2' indicates that there is either no voltage sent or the voltage sent is independent of its inputs and hence such a failure cannot be compensated for by the loop.

4.2 Mark Library

The mark library is a new concept introduced in this paper. While tracing a fault through a system automatically, it is difficult to model the trace algorithm around complex features such as loops or redundancy. The logic in most cases becomes erratic. The mark library is introduced in order to deploy rules in the form of operators when such features are encountered in the system. For this system, two marks, namely Negative Feed-Back Loop Mark (NFBL Mark) and Redundancy Mark (RMark) are introduced.

NFBL Mark: During automatic fault tree generation, when an event is found to be part of the NFBL, the fault tree takes the form of the mini-fault tree shown in Figure 4.

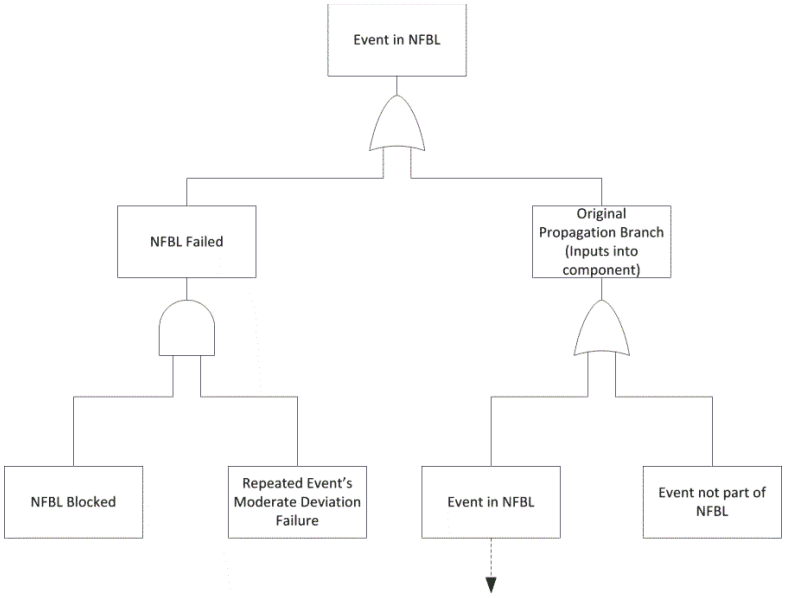


Figure 4 - NFBL Operator

The NFBL operator implies that the top event occurs either if the loop itself has failed or if the fault propagates from another component. The event 'NFBL Failed' implies that the loop itself has failed and is thereby unable to compensate any moderate disturbances. The event 'NFBL Blocked' will combine under an OR gate, failure modes of components in the NFBL that are not affected by the input. All these failure modes have a transmit coefficient of 0 in the I/O tables which implies that their failure is independent of the input and hence cannot be compensated by the loop. Event 'Original Propagation Branch' traces the fault to connecting components. If any of these input events are part of the NFBL too, the same operator is deployed again, if not, they are dealt with the help of an algorithm which is explained in Section 5.
RMark: The redundancy mark is deployed in case of multiple redundancies. In such a scenario, a failure occurs only if the necessary input has failed as well as all the redundancies. Hence the inputs are combined under an AND gate as can be seen in Figure 5.

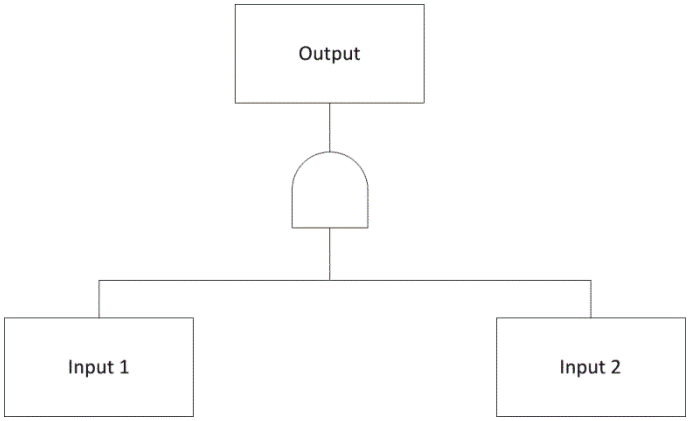


Figure 5 - Redundancy Mark Operator

4.3 Deviation Tree

Deviation Tree is an intermediate structure which portrays the principal diagram along with its inputs, outputs and the transmit coefficient between them. This structure can be automatically generated and is used along with component tables and mark operators to generate a fault tree for any given top event in a systematic manner.

The deviation tree is similar to the topology graph generated by Andrews and Henry [7]. In their paper, a topology graph was generated as an intermediate representation to aid with identification of loops.

The method of identifying the NFBL loop from the deviation tree is by following the path beginning and ending at the very same event. For the example system, this path would be E-IV-C-EFS-E as can be seen in Figure 6. The deviation tree terminates when event E-F_{do} (w) is encountered for the second time.

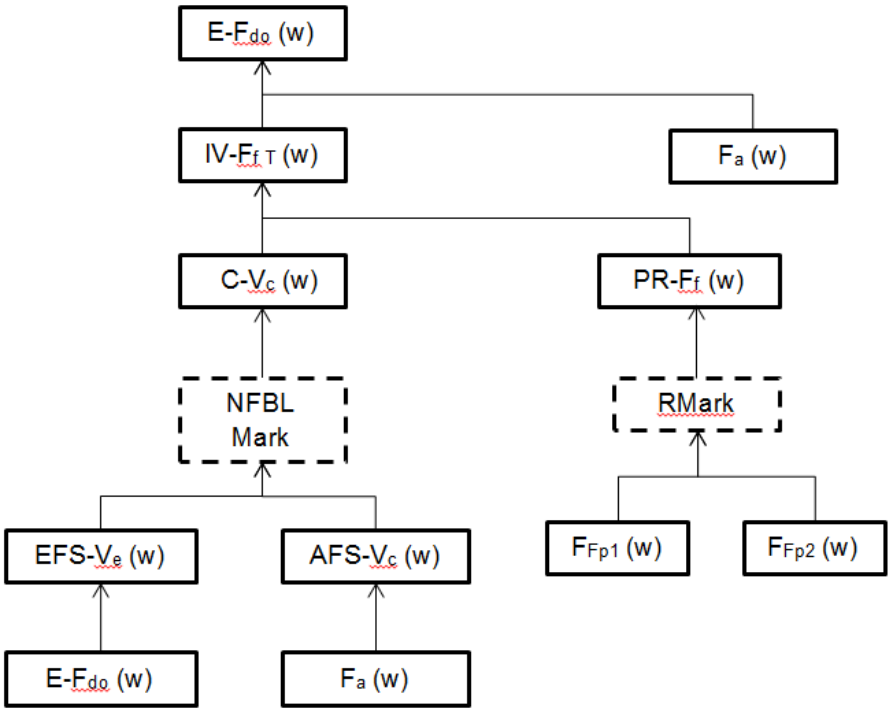


Figure 6 - Deviation Tree

Besides the marks introduced in this paper, there is much scope to model operators for other complex features, for example, 'Negative Feed Forward Loop' or 'Vote Redundancy'.

5. Fault Tree Generation

In this section a fault tree is generated in a systematic manner following the methodology described in the previous sections.

The top event selected is an output deviation of F_{do} (-1) from the Engine. As explained previously, F_{do} is the mass flow rate of emissions and a deviation of -1 implies that the density of unburnt oxygen is lower than the desired air-fuel ratio thereby indicating a rich mixture and in turn black smoke out of the exhaust. The reason for selection of this top event is the environmental impact of the deviation.

The algorithm searches the deviation tree for the component associated with the top event and this becomes the starting point to trace the fault through the system. While generating the fault tree, each event is checked if it belongs to any of the marks. If it is, the appropriate operators are deployed, if not, the algorithm follows the rules stated below:-

1. Check I/O tables under 'Failure Modes' if deviation can be caused by any of them. All such failures are combined under an OR gate.
2. Check I/O tables 'Normal State' to track fault through component inputs and transmit coefficient. If more than one input is found, combine events under OR gate.

Figure 7 shows the fault tree generated for top event F_{do} (-1). The events are named in the following way – 'Component acronym' followed by a number which represents the failure mode in the I/O tables. For example event 'E-3' denotes 'Engine failure 3'. In the example system, 4 out of 6 components are part of the NFBL which explains the repeated occurrence of event 'NFBL Failed' in the fault tree. It is also essential to make the above algorithm work seamlessly with the marks while deploying logic gates and events. Figure 4 shows the NFBL operator which has a block 'Event not part of NFBL'. This allows events outside the NFBL to be developed at the same time, for example, event 'F_a (-1)' or 'PR-F_f (+1)'. The deployment of the NFBL events has been explained in Section 4.2. The fault tree generated can be further trimmed down by deleting repeated events and inconsistent events i.e. events that have already occurred higher up in the fault tree by using techniques introduced in the past. [8]

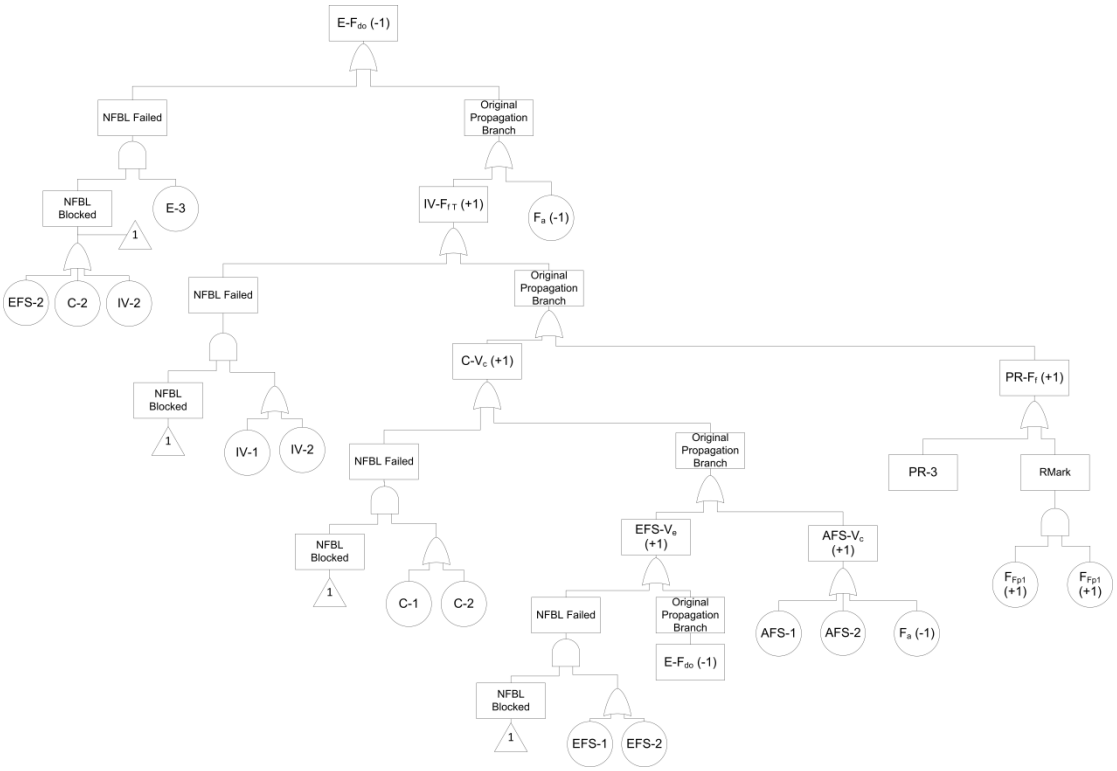


Figure 7 - Fault Tree for Top Event 'Black Smoke out of Exhaust'

6. Conclusion

The concern of assisted fault tree construction during the design stage has been undergoing research since the 1970s and has gone a long way since then. Component and mark libraries can prove useful in reducing human effort and retaining human supervision at the same time. A component library can significantly assist systematic fault tree generation by providing the user with predefined component tables that represent the behaviour of components under failure or abnormality. A mark library can reduce the semantic gap in representing complex systems by taking into account the effect of loops or

redundancies while generating the fault tree. In this paper, only 6 component tables and 2 marks were employed, but it provides a platform for expansion.

Component and Mark libraries can be designed in a way that they can simply be dragged and dropped onto the system diagram. Having linked the component tables and mark operators to the concerned components, they can either be used in their original form or be edited to suit the requirements of specific components that are not found in the library. This ensures human supervision and at the same time avoids repeatability in terms of defining failure modes for each component.

A generic component library can be developed for each discipline which stores component tables for discipline specific components. This methodology is still being developed to accommodate systems from different disciplines of engineering. Editing and pruning procedures are also being developed so that the tree can be reduced to a concise form by eliminating repeated and contradictory events. Human supervision with minimal effort would be the ideal degree of automation for a technique as sensitive as fault tree generation.

7. References

[1] S. Pilot, “What is a Fault Tree Analysis?,” 2002. [Online]. Available: 1. http://asq.org/quality-progress/2002/03/problem-solving/what-is-a-fault-tree-analysis.html.

[2] S. A. Lapp and G. J. Powers, “Computer-aided Synthesis of Fault-trees,” *Reliab. IEEE Trans.*, vol. R-26, no. 1, pp. 2–13, 1977.

[3] S. L. Salem, G. E. Apostolakis, and D. Okrent, “A new methodology for the computer-aided construction of fault trees,” *Ann. Nucl. Energy*, vol. 4, no. 9–10, pp. 417–433, 1977.

[4] U. of Hull, “HiP-HOPS User Manual.” 2013.

[5] R. Bosch, *Automotive Electric/Electronic Systems*. 1988.

[6] M. T. Sunderland, “Lambda Sensors,” 2008. [Online]. Available: 10. http://www.picoauto.com/applications/lambda-sensor.html.

[7] J. D. Andrews and J. J. Henry, “A computerized fault tree construction methodology,” *Proc. Inst. Mech. Eng. Part E J. Process Mech. Eng.*, vol. 211, no. 3, pp. 171–183, 1997.

[8] A. Majdara and T. Wakabayashi, “Component-based modeling of systems for automated fault tree generation,” *Reliab. Eng. Syst. Saf.*, vol. 94, no. 6, pp. 1076–1086, 2009.