Computerized fault tree construction for improved reliability analysis

A. Majdara & T. Wakabayashi
Department of Management Science and Technology,
Graduate School of Engineering, Tohoku University, Japan

Abstract

Fault Tree Analysis is a well-known method for reliability evaluation of systems. However, manual construction of fault trees is a tedious and time-consuming task. Thus, many researchers tried to get benefit of high speed and accuracy of digital computers to automate this process. Automated construction of fault trees can be very useful in system reliability analysis, especially in design step, where we need to choose the most reliable design out of several design options.

In this paper we will present the computer code we have developed for automated fault tree generation. The program is actually the implementation of an approach we have developed for algorithmic construction of fault trees. The main part of this approach is a component-based method for system modeling. In this method, a system is modeled as a set of components connected to each other. Every component is described in a function table. This modeling approach is capable of modeling a wide range of devices and concepts in different types of systems. The model prepared in this part is then used as an input to the "fault tree synthesis algorithm", and the result is the fault tree for the specified top event. A case study is done for a part of a UAV system. The results generated by the program are compared with the manually constructed fault trees.

Keywords: fault tree, computer-aided fault tree construction, component-based modeling, function table, state transition table, trace-back algorithm.

1 Introduction

A fault tree is a graphical representation of the various combinations of faults and failures that will result in an undesired event, which is called top event [1–3]. Since manual construction of fault trees is tedious and time consuming and it is vulnerable to human mistakes, there have been interests in computerizing this task [4–13]. Automating the fault tree construction process can make it faster and easier. Also, computer-aided fault tree generation, if implemented appropriately, will have fewer systematic errors than the manual approach.

Generally, any automated approach for fault tree construction can be considered as having two phases: First, system modeling; and second, fault tree synthesis. The challenging part is the first phase, in which the system under study should be modeled appropriately without ignoring any necessary details. Also, the system model must be suitable for being used by an automated procedure, in the second step.

Various modeling approaches are utilized by different researchers. Some of the most commonly used approaches are digraphs [4–6], decision tables [7–9], and state diagrams [10, 11]. Kumamoto and Henley proposed a semantic network modeling approach [12]. Papadopoulos et al. used Matlab–Simulink models for model-based synthesis of fault trees [13].

We have developed an improved methodology for computer-aided fault tree construction, which consists of a component-based approach for system modeling and a trace-back algorithm for fault tree synthesis. The approach is then implemented as a computer program which can receive the system model as an input and generate the fault trees quickly. In the next section, a brief overview of the methodology is presented. Then, in section 3, the computer program for fault tree construction is introduced. Section 4 presents a case study we have performed on a part of an Unmanned Aerial Vehicle (UAV). The last part of this paper contains discussions and conclusions.

2 General description of the approach

We use a component-based approach for system modeling, meaning that every entity in a system is modeled as a component, or a parameter of a component. This includes actual devices, human operators, external events, etc. Then, the system will be modeled as a set of components connected to each other. Various types of flows can be transported from one component to another, inside the system. The details of the modeling methodology are explained in [14].

Then, in the second step, an algorithm is applied to this system model to generate the fault trees. This *trace-back algorithm* starts from the component in which the top event occurs, and examines all the components that might be involved in causing the top event to occur, and tries to find all the possible fault paths that can lead into the top event. The faults and failures of components are combined to each other by logical AND and OR gates, to form the tree structure.

3 Computer program for fault tree generation

We have implemented the whole approach as a computer code, with the following parts:

- Interface for making component models
- Graphical interface for system design

- Fault tree synthesis procedures
- Output window for viewing of the results

In the following sections, any of these four parts will be explained.

3.1 Interface for making component models

In this approach, *components* are the building blocks of systems. Thus, in order to be able to model a system, first we should prepare the required component models. A typical component is modeled as a simple box with a caption, several input and output ports, and a *function table.* As shown in figure 1, a component can interact with the other components of the system via its input and output ports.

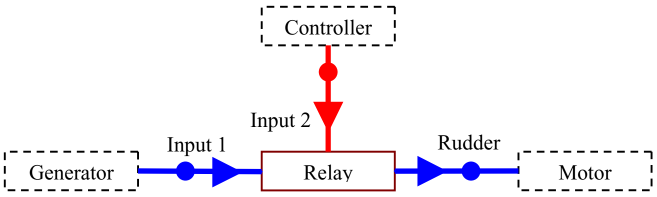

Figure 1: Component model for a relay as a part of a system.

For each component, the input-output relationships are described in a function table. Figure 1 shows the component model for a relay, and the function table for this component is as shown in table 1.

Table 1: Function table for a relay.

Input 1	*Input 2*	*Functionality Condition*	*Output*
1	1	OK	1
0	-	-	0
1	-	Stuck Closed	1
-	-	Fail to Close	0
-	0	OK	0

The *functionality condition* column shows whether the component is working in its normal operating mode or it is in a failed state.

Making a new component model in this program takes place in three steps:

3.1.1 Defining a new component

Basic information about the component should be entered at the first step. The component-designing window allows the user to make the component models to be later used in the system-modeling step. To do so, a name and a category should be selected for any new component. Also, the number of input and output ports should be specified. Any newly designed component is stored in a component library for future use. Categorization of the components allows having an easier access to the component in the system-modeling phase.

3.1.2 Adding detailed descriptions

After defining a new component, more detailed descriptions about the component should be provided in another window. This includes flow types for each of the ports, range of variation for these flows, and also failure modes of the component. For any parameter of a component, a *discrete* values domain should be specified.

3.1.3 Filling up the function table and / or state transition table

At this point, the function table for the component will be created automatically. However it needs to be filled up by the user. Each cell of the table is occupied by the values from the values domain specified in the previous step. In each table cell, these values appear as a drop down list. The user fills up the function table based on the information he has about the way the component functions. He should specify the output value for any of the different combinations of inputs, functionality condition, and any other parameter.

As explained in [14], some components may also need a state transition table, to describe how they move from one state to another. For these components, a similar procedure as above should be followed for filling up the state transition table.

The component modeled in these 3 steps is now ready to use in a system model. It will be stored in a component library for later being used in modeling systems.

3.2 Graphical interface for system design

A graphical interface is provided for modeling the hardware configuration of the system. In this window one can use the components in the library to design his system. A list of the components existing in the library is provided in this window. Any required component can be easily selected from its category in the library, and added to the system model. If some components we need do not exist in the library, we should go to the component-making window, create the components, and then get back to the system design window.

Actually, when all of the component models are already constructed, it will not take more than a few minutes to model the system as a set of components connected to each other.

After completing the system-modeling phase, the last thing to do is defining the top event. In our methodology, a top event is defined by assigning specific values to some of the parameters of the system. The top event window of the program assists this procedure by providing lists of all components, their parameters, and the values domain for any of these parameters.

3.3 Fault tree synthesis procedures

The main part of the program is its fault tree synthesis procedures, which are responsible to analyze the system model and generate the fault tree for the top event.

When the system is modeled completely and the top event is defined, we can run the program so that the fault trees are automatically generated. This part is actually the implementation of the trace-back algorithm. A flowchart for this algorithm is presented in [14].

The trace-back algorithm starts from the point of occurrence of the top event, and examines the function tables and state transition tables of the components to find all of the potential causes of that top event. The algorithm jumps from one component to another, in reverse directions, i.e. reverse to the direction of the flows.

When the algorithm is examining a component, its function table or state transition table is searched to find the rows with the output value of interest. Three different cases are possible: If there is a functionality condition of the component leading to that output value, a basic event is added to the tree and current branch of the tree is terminated.

However, if it is caused by the external inputs, trace-back is continued to the component from which these inputs are coming. Finally, in case of a *state* column affecting the output value for that row, the algorithm should jump to the corresponding state transition table. This procedure is continued until system boundaries are reached.

A similar procedure exists for the cases the algorithm is examining a state transition tables of the components.

3.4 Output window for viewing the results

After the fault trees are generated by the program, they can be viewed in an output page. The output page shows the resultant fault trees in formats compatible with the well-known fault tree standards. The fault trees can be edited and saved in this window.

4 Case study

In order to show the applicability of the approach to actual cases, and to survey the validity of the results of the developed computer program, a case study is performed. The system we chose for the case study is shown in figure 2. This system is a part of an UAV for flights in populated areas [15], which includes electric power systems, electronic flight control systems, and rudder actuators.

4.1 System description

The electrical power supply system of this UAV is responsible for providing electrical supply for the actuators and critical avionics. It consists of generators, converters, and batteries. The generators serve as starters as well as generators to charge the batteries.

In normal operation, the generators supply the actuators, while the critical avionics are supplied by the converters. The batteries are charged by the generators through the convertors. The convertors are bidirectional, meaning that

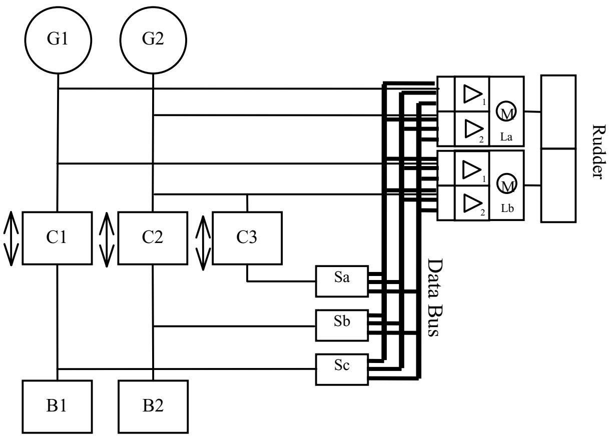

Figure 2: UAV system as a case study.

in the case of failure of generators, the batteries will supply the actuators and the electric motor via the converters.

The mechanical part of the system consists of actuators, motors, and the rudder. The actuators used in this design are electromechanical actuators, which are based on an electrical motor with a gear to linear or angular mechanical output and drive electronics. The actuators include position sensors and commutation sensors.

The critical avionics of the UAV is the part of the electronic units which uses the control commands from the sensors and flight conditions to compute the actuator commands for the vehicle.

4.2 Component-based modeling of the system

Based on the system description in the previous section, we have developed the component-based model for this system as shown in figure 3. Because of the redundant elements, the number of connections has increased dramatically. A system model like figure 3 is entered into the program, using the system design interface. In our component-based model the top event will de expressed as "Rudder Output = 0". Figure 3 only represents the hardware configuration of the system. For any of the components, the input-output relationships are described in their function tables. Tables 2 to 8 show the function tables of the components. The small components named as "J" and "E", represent Junctions and Extensions, respectively, which are virtual components used for modeling purposes only [14].

Table 2: Function table for G1, G2, B1, and B2.

Functionality Condition	*Output*
OK	1
Failed	0

Table 3: Function table for the rudder.

Inp1	*Inp2*	*Functionality Condition*	*Output*
0	0	-	0
-	-	Failed	0
1	-	OK	1
-	1	OK	1

Table 4: Function table for C1, C2.

Inp1	*Inp2*	*Functionality Condition*	*Outp1*	*Outp2*
0	-	-	0	-
-	0	-	-	0
-	-	Failed	0	0

Table 5: Function table for C3, S_a, S_b, and S_c.

Inp	*Functionality Condition*	*Outp*
0	-	0
-	Failed	0
1	OK	1

Table 6: Function table for L_a part 1, L_a part 2, L_b part 1, L_b part 2.

Inp1	*Inp2*	*Inp3*	*Power Input*	*Functionality Condition*	*Outp*
-	-	-	0	-	0
-	-	-	-	Failed	0
0	0	0	-	-	0
1	-	-	1	OK	1
-	1	-	1	OK	1
-	-	1	1	OK	1

Table 7: Function table for L_a, L_b.

Inp	*Functionality Condition*	*Outp*
0	-	0
-	Failed	0
1	OK	1

Table 8: Function table for data bus.

Inp1	*Inp2*	*Inp3*	*Functionality Condition*	*Outp1*	*Outp2*	*Outp3*
0	-	-	-	0	-	-
-	0	-	-	-	0	-
-	-	0	-	-	-	0
-	-	-	Failed	0	0	0
1	-	-	OK	1	-	-
-	1	-	OK	-	1	-
-	-	1	OK	-	-	1

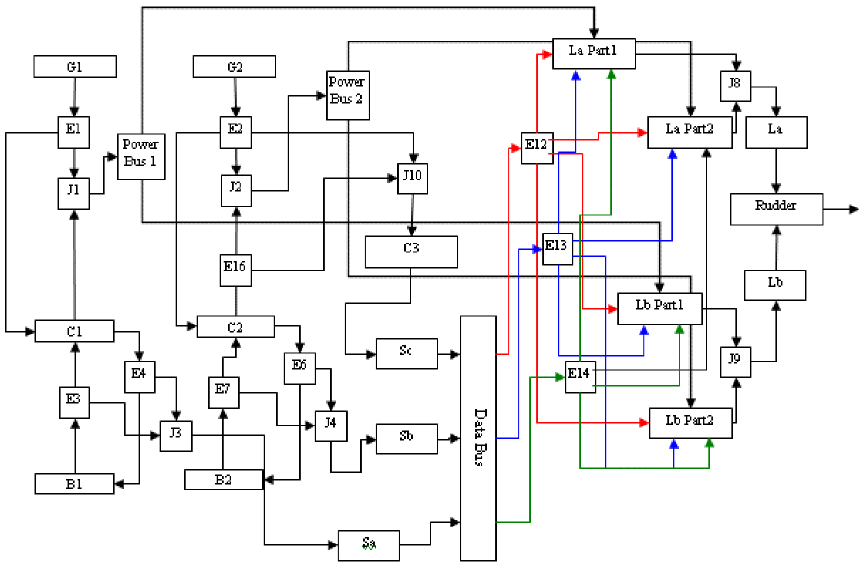

Figure 3: Component-based model for the UAV system.

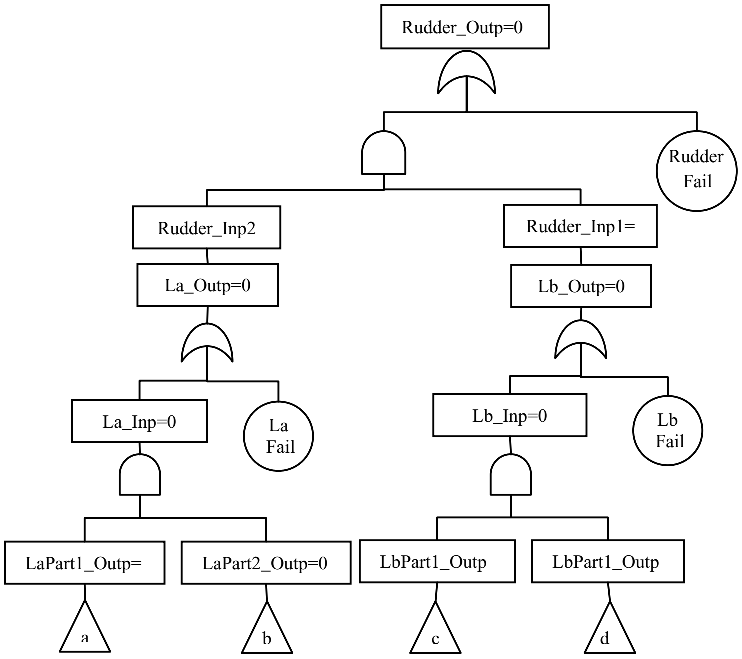

Figure 4: Fault tree for the UAV system.

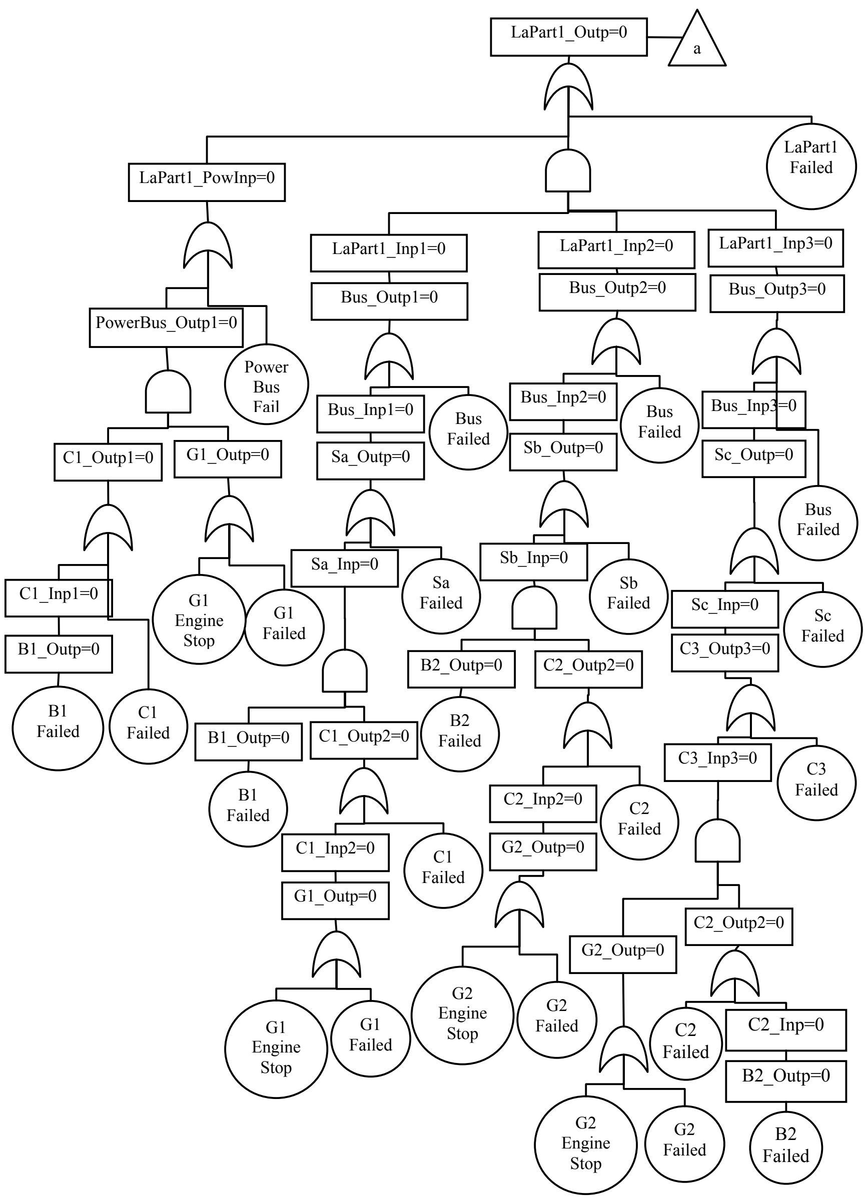

Figure 5: Fault tree for the event “LaPart1_Outp=0”.

4.3 Results

Nyström, et al performed fault tree analysis for this system [15]. The fault trees are shown in their paper. So, in order to be able to have a comparison, we have generated the fault trees for the same top event with same assumptions and conditions. The top event is the failure of the whole system, i.e. loss of rudder function. This is equivalent to the top event definition as "Rudder_Outp=0" in our methodology.

After top event definition and running the program, it takes less than 2 minutes for the program to generate the fault tree logic and draw the tree. Similar to the manually constructed fault trees, the fault tree generated by the program is very large, too. So, in order to make it useable we have broken it into 5 parts. The upper part of the tree, starting from the top event, is shown in figure 4. To make the results easily comparable to the original fault trees, we have located the transfer points in similar places to those of the manually constructed fault tree.

The 4 sub-trees of a, b, c, and d are quite similar in structure. Thus, we just show one of them here, which is the fault tree corresponding to transfer-in point a. This sub-tree is presented in figure 5.

The fault trees constructed by manual and automated approach have similar structures. In order to have a better comparison, we have created the Boolean expression for the automatically generated fault tree. The failure rates of the basic events of the tree are provided in the original work [15]. Having these data, and using the created Boolean expression, we have calculated the top event probability for the fault tree generated by the program. This value is compared to the top event probability for the manual fault tree.

5 Discussions and conclusions

Comparing the fault trees shows a good match between the manually and automatically constructed fault trees. Some minor differences can be seen in some points, mainly in the location or order of appearance of some events. This is mainly because the automated procedure is an exact step-by-step procedure and the trace-back algorithm moves from one component to the immediate antecedent of that component. However in manual fault tree construction, different people have different overviews of the system, so different results might be created.

The fault trees constructed by manual and automated approach have similar structures. In order to have a better comparison, we have created the Boolean expression for the automatically generated fault tree. The failure rates of the basic events of the tree are provided in the original work [15]. Having these data, and using the created Boolean expression, we have calculated the top event probability for the fault tree generated by the program. This value is compared to the top event probability for the manual fault tree.

As shown in table 9, the values for the top event probabilities are very close. In fact, the small difference in the values is due to a difference in the logic of the trees. It can be seen in [15] that in the manual fault tree, two intermediate events of "Loss of power supply from G2" and "Loss of power supply from C2" are combined by an OR gate (GATE 51), which we believe is a mistake. According to the caption of this gate, "Loss of power supply from G2 *and* C2", this seemingly should be an AND gate, which is the case in the automated fault tree.

Table 9: Comparing the results from the manual and automated fault trees.

	Manual FT	*Automated FT*
Basic Events	25	25
Intermediate Events	18	35
AND Gates	4	5
OR Gates	15	17
Top Event Probability	2.25E-4	2.70E-4

If this OR gate is replaced by an AND gate, the top event values for the two trees will be exactly the same, approving that the results generated by our program completely matches the manually constructed fault trees.

The higher number of intermediate events in the computer generated fault trees shows that this fault tree contains more details about the exact fault paths that can lead into the top event. In manual fault tree construction, some details and intermediate steps might be skipped.

Thus the validity of the results for this case has been verified. Use of computerized fault tree construction can assist the safety analysis process, especially for large systems. This is especially useful in system design step, where the most reliable design must be chosen out of many design options. Also, sometimes, as the sample case shown in this case study, using the results from the computerized fault tree construction may help us find some mistakes in the manually constructed fault trees.

Developing complete and customized component libraries for systems and plants makes this automated process suitable for being used for large-scale systems.

Reference

[1] Vesely, W.E., et al, *Fault tree handbook. NUREG-049,* Washington, DC: US Nuclear Regulatory Commission, 1981.

[2] Hasal, D.F., *Advanced concepts in fault tree analysis,* Seattle, WA, Aero-Space Division of the Boeing Company, 1965.

[3] Ericson, C.A., Fault tree analysis—a history. *Proc. of the 17th Int. system Safety Conference,* Orlando, 1999.

[4] Wang, Y., et al, A new algorithm for computer-aided fault tree synthesis. *Journal of Loss Prevention in Process Industries,* **15,** pp. 265–277, 2002

[5] Chuei-Tin, C. & Her-Chuan, H., New developments of the digraph-based techniques for fault tree synthesis. *Industrial & Engineering Chemistry Research,* **31,** pp.1490–1502, 1992

[6] Lapp, S.A. & Powers, G.J., Computer-aided synthesis of fault-trees. *IEEE Transactions on Reliability,* **R-26,** pp. 2–13, 1977

[7] Andrews, J.D. & Henry, J.J., A computerized fault tree construction methodology. *Proceedings of Institute of Mechanical Engineers,* **211 (E),** pp. 171–183, 1997

[8] Henry, J.J. & Andrews, J.D., Computerized fault tree construction for a train braking system. *Quality and Reliability Engineering International,* **13 (5),** pp. 299–309, 1997.

[9] Salem, S.L., Apostolakis, G.E., & Okrent, D., A new methodology for the computer-aided construction of fault tree. *Annals of Nuclear Energy,* **4,** pp. 17–433. 1977;

[10] Liggesmeyer, P. & Rothfelder, M., Improving system reliability with automatic fault tree generation. Proc. of the 28th Annual Int. Symposium on Fault-tolerance Computing, pp. 90–99, 1998.

[11] Taylor, J.R., An algorithm for fault-tree construction. *IEEE Transactions on Reliability,* **R-31 (2),** pp. 137–146, 1982.

[12] Kumamoto, H. & Henley, E.J., Automated fault tree synthesis by semantic network modeling, rulebased development and recursive 3-value procedure. *Reliability Engineering and System Safety,* **49,** pp. 171–188. 1995

[13] Papadopoulos, Y. & Marhun, M., Model-based synthesis of fault trees from Matlab–Simulink models. *Proc. of the Int. Conf. on Dependable Systems and Networks (DSN'01),* Sweden, pp. 77–82, 2001.

[14] Majdara, A. & Wakabayashi, T., Component-based modeling of systems for automated fault tree generation, *Reliability Engineering and System Safety,* **94,** pp. 1076-1086, 2009

[15] Nyström, B., et al, Fault tree analysis of an aircraft electronic power supply system to electrical actuators, *9th Int. Conf. on Probabilistic Methods Applied to Power Systems,* KTH, Stockholm, June 11-15, 2006.