IEEE SysCon 2009 —3rd Annual IEEE International Systems Conference, 2009
Vancouver, Canada, March 23–26, 2009

A New Approach for Computer-Aided Fault Tree Generation

Aref Majdara
Department of Management Science and Technology
Graduate School of Engineering, Tohoku University
Sendai, Japan
majdara.aref@most.tohoku.ac.jp

Toshio Wakabayashi
Department of Management Science and Technology
Graduate School of Engineering, Tohoku University
Sendai, Japan
toshio.wakabayashi@qse.tohoku.ac.jp

***Abstract*–A new method for automated fault tree generation is presented in this paper. In this method, every system is modeled as a set of components and flows. Each component is described by its function tables and state transition tables. Saving the designed components in a component library keeps them available for future use in other systems. A trace-back algorithm works on system model, and generates the required fault trees. Reencountered or inconsistent events are removed from the tree. The proposed approach is capable of modeling different types of systems having various kinds of components and connections. A sample system is presented to show how this method works.**

Keywords-fault trees; component-based modeling; function table; state transition table; flows; trace-back algorithm

I. INTRODUCTION

Since 1970's, there have been many attempts to get benefit of computers to make the fault tree construction procedure faster and easier [1]. Almost all of the automated fault tree generators can be considered as having two main steps. The first step is to model the system configuration in an appropriate way to be manipulated by an algorithm or a computer program. In the second step, an automated fault tree synthesis algorithm or program gets this model as its input, analyses it, and generates the required fault trees. The challenging part is usually in the first step, i.e. modeling the system with all necessary details in a systematic way. Previous studies have used different methods for system modeling, including digraphs [2]-[7], decision tables [8]-[10], state diagrams [11]-[13], semantic network modeling [14], and other methods [15]-[19].

We have tried to get the useful ideas from each of these methods, and propose a new modeling approach with higher capabilities. In this paper we present a new method for automated fault tree construction, which mainly consists of a component-based approach for system modeling, and a trace-back algorithm for synthesizing the fault trees (shown in Fig. 3).

II. COMPONENT-BASED MODELING OF SYSTEMS

In this approach, every system is considered as a network of components connected to each other, and different types of flows like energy, command or material, being transported among these components. The input-output relations for each component of the system are described in its *function table*. For components having different operational states, a *state transition table* describes how the component jumps from one state to another.

To see how our modeling approach works, let us have a look on the sample system shown in figure 1. It is a simple train detection system, presented by Andrews and Henry in their paper [8]. This system is composed of a track circuit to detect the train, and a signal circuit, to make the appropriate signaling.

In order to prepare the system model, we need enough knowledge on both configuration and functional description of the system. When there is no train, G1 excites relay core, which in turn attracts the contact, so that the circuit for the green light is closed; thus, in this case, the green light is on and the red light is off. When a train arrives, the track circuit is short-cut through the train axle. Therefore, the relay is not excited, and consequently, the red light is on while the green light is off.

A system can be modeled in different ways, depending on the purpose of the modeling and the details we would like to be included in the model. Based on this functional description of the train detection system and our modeling goals, we decided to model the system as shown in figure 2.

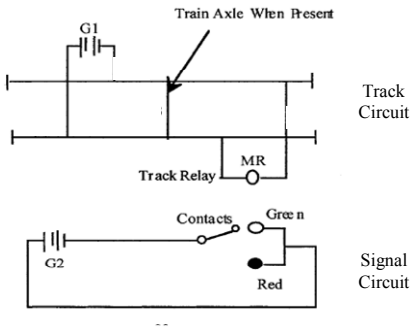

Figure 1. Train detection system

This component-based representation gives us the hardware configuration of the system in form of relations between components. However, it does not explain that for a single component, how the outputs are related to inputs and other

9781-4244-3463-3/09/$25.00 ©2009 IEEE

parameters. To describe what happens inside each of these components, we use function tables and state transition tables. A complete system model consists of a configuration model like figure 2, tables of components, and some extra settings for the system.

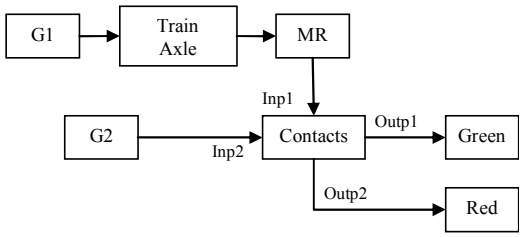

Figure 2. Component-based model of the

A. Function tables

Each component in the system model has some inputs, outputs, and an internal parameter called the functionality condition, which tells us whether the component is working or it is in a failure condition. A function table shows how the output of a component is determined based on the values of its inputs, state, and functionality condition. As an example, the function table for the *contacts* in our sample system is shown in table 1.

TABLE I. FUNCTION TABLE FOR RELAY CONTACT

Inp1	Inp2	Functionality Condition	Outp1	Outp2
–	0	–	0	0
0	1	ok	0	1
1	1	ok	1	0
–	1	Fail to close	0	1
–	1	Fail to open	1	0

This table shows how the values of the 2 outputs are determined based on the values of inputs and the functionality condition. Two failure modes are considered in this table. Each of the other components of the system model has its own function table. The component-based representation plus all the function tables provide enough information about the system, so that the required fault trees could be generated.

B. StateTtransitionTtables

Some components may have more than one operating states. For example a simple switch can be open or close at any given time. For this type of components we use a *state transition table* to show how the component goes from one state to another state. The change from one state to another state can be made as a result of a change in some inputs or more specifically, *commands*. Actually, state transition tables can be considered as tabular representations of the well-known *state diagrams*. State transition table of a manual switch is shown in table 2, as an example.

The *final state* values of the state transition table appear as a new column in the function table of the same component. Then the output of the component will be determined based on its state and the value of its inputs. For example, for the simple switch described before, we will have a function table as shown in table 3.

TABLE II. STATE TRANSITION TABLE FOR A MANUAL SWITCH

First State	Command	Functionality Condition	Final State
Open	Close	OK	Close
Open	Close	Fail-to-Close	Open
Open	Open	-	Open
Close	Open	OK	Open
Close	Open	Fail-to-Open	Close
Close	Close	-	Close
Open	No command	-	Open
Close	No command	-	Close

TABLE III. FUNCTION TABLE FOR THE MANUAL SWITCH

Flow Input	State	Flow Output
0	-	0
-	Close	0
1	Open	1

C. Flows

A *flow* is defined as any material, information, energy, or command which can propagate through the system and be transferred from one component to another. A system can be viewed as a pipeline structure of flows and equipments along flow paths [14]. We categorize flows into following four groups:

- Material flow; like water, gas, steam
- Energy flow; such as electricity, light, heat, sound, torque
- Information flow; like signals or data
- Commands; such as pushing a button by an operator

Flows and their properties appear in input and output columns of function tables and state transition tables. In train detection system, there are electrical and light flows as inputs and outputs of components.

D. Top event

In this method, a top event is defined by assigning specific values to some of the system parameters, like inputs or outputs of some components. For example a top event for the train detection system could be “Red light is off when a train is present”. As in this example, top events definitions usually consist of a main phrase plus some conditioning terms.

III. FAULT TREE SYNTHESIS

When the system modeling is accomplished and the top event is defined, it is time to analyze the model to generate the fault tree for the specified top event. For this purpose, we use a

trace-back algorithm which starts from the point of occurrence of the top event in the system, moves from one component to another until it finds all of the possible combination of faults and failures of components which can cause the top event.

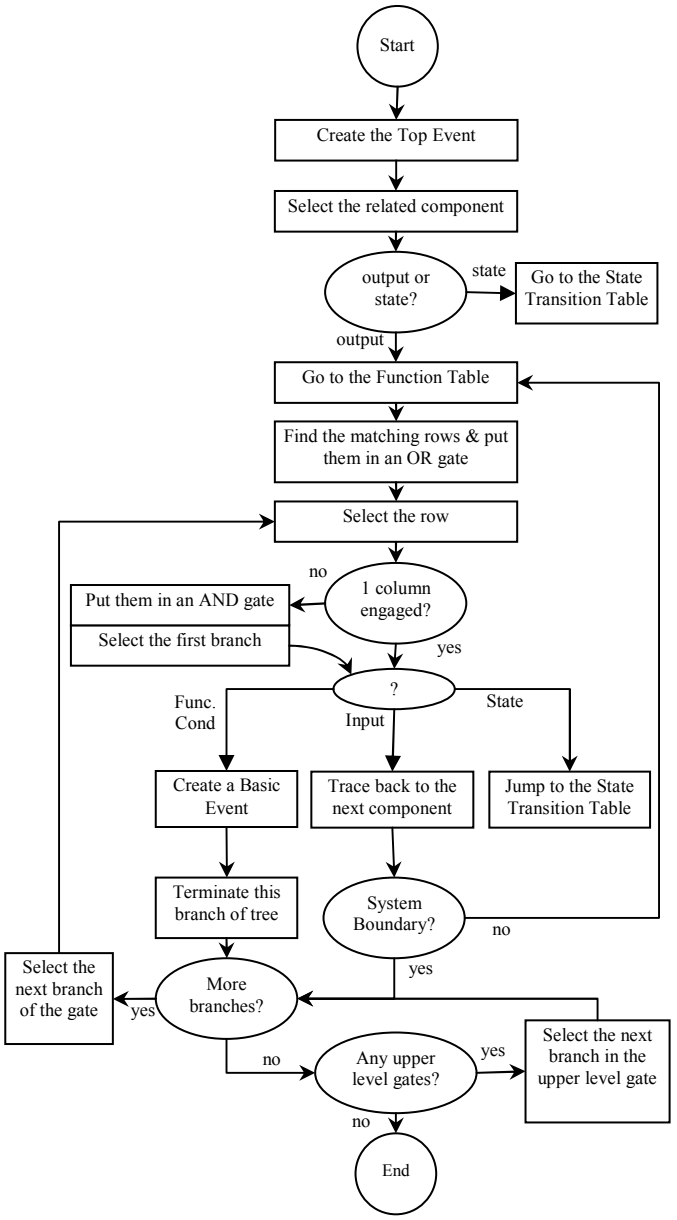

Figure 3. Trace-back flowchart for fault tree generation

A. *Trace-back Algorithm*

The trace-back algorithm is represented as a flowchart in figure 3. For each of the components, the algorithm examines the function tables and state transition tables to find the affecting faults and failures. Components failures appear as basic events in the fault tree, while faults related to an input of a component create intermediate an event in the tree, and the algorithm goes on to the component from which this input is coming.

The algorithm stops when all of the created gates and intermediate events are fully developed as far as possible. When for some intermediate events it is not either possible or necessary to develop further, the event is replaced by an *undeveloped* event.

As can be seen in two points of the flowchart, when there is an event which is related to a state of the component, the algorithm jumps to the state transition table of that component. There is a procedure similar to figure 3 for analyzing state transition tables.

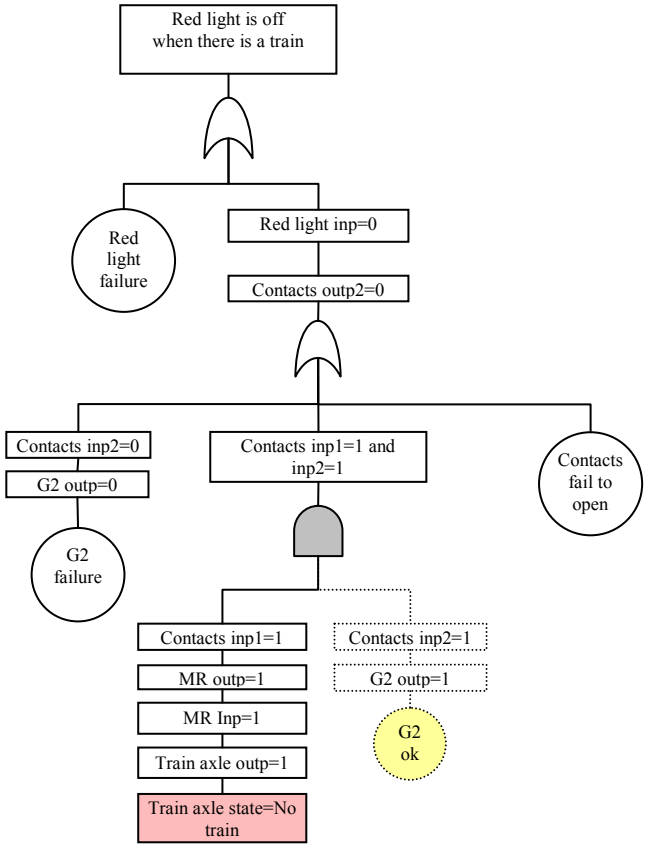

Figure 4. Fault tree of the train detection system before final modifications

The generated fault tree consists of one top event as the root of the tree, some intermediate events shown as rectangles, basic events in circular shapes, and logic AND and OR gates. As a basic rule, events related to normal operation of components do not appear in the fault tree. Thus, the event "G2 ok" and the two events on its top should be removed from the tree. There are also other modifications to this tree, as explained in the next section. The final form of the fault tree is shown in figure 5. There is no undeveloped event in this tree, as all of the events are developed to the end. We show undeveloped events as diamond shapes in fault trees, when required.

The fault tree for the event of red light being off while a train is present, is generated by using the described algorithm.

Figure 4 shows the primary appearance of the generated fault tree, which needs some modifications.

B. Fault Tree Modifications

1) Consistency Checking: When a fault tree is being generated, it is checked for serial consistency, and inconsistent events are removed from the tree. Serial inconsistency occurs when 2 events in the same branch of the tree are inconsistent. In this case, the event in the bottom is removed. Also, when the tree is constructed completely, parallel consistency check is performed. If two inconsistent events are placed in 2 different branches of an AND gate, there is a parallel inconsistency; and the AND gate is removed along with all of its branches. For example, in the fault tree of figure 4, there is an event requiring that there is no train. Since it is stated in the top event statement that there *is* a train, this intermediate event is inconsistent with the top event (serial inconsistency) and should be removed from the tree. Consequently, all of the events directly on the top of this event are omitted, until a gate is encountered. If it is an AND gate, the gate will be removed, too.

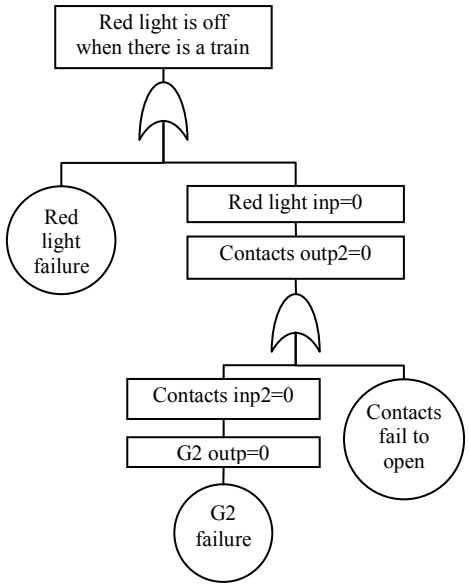

Figure 5. Final form of the fault tree for train detection system

2) Removing Reencountered Events: Repeatedly encountered events can lead to infinite loops in fault tree structure. In order to avoid this, a reencountered event shoud be replaced by an undeveloped event. It should be noted that an event is considered as a reencountered event only if it is exactly the same as an event which exists inside the *same* branch of the tree. In other words, repeated events are permitted in different branches of the tree, of course.

3) Fault Tree Simplifications: Different kinds of simplifications could be applicable to a tree. After removal of inconsistent events, there might be some gates with one input. These gates can be simply replaced by a simple connecting line. Large trees with similar parts can be made smaller and more comprehendible by using *transfer-in* and *transfer-out* gates [20].

IV. CONLCUSIONS

A new method for algorithmic generation of fault trees was presented in this paper. The approach consists of a component-based technique for system modeling and a trace-back algorithm for synthesis of the trees. The purpose was to get the useful ideas from different previous methods, and use them in this approach; because any of them have their own positive and negative points. For example, digraph-based approaches are really suitable for modeling process systems and control loops, but modeling discrete quantities seems to be inconvenient in these methods, especially when some discrete parameters have wide ranges of variation, which leads to very large digraphs, with lots of directed connections.

The simple train detection system helped us demonstrate how the proposed method works. After the trace-back algorithm generates the primary fault tree, inconsistent and reencountered events are removed from the tree. Also, events which represent a normal operating status of a component are omitted.

If we model the components independent of the system, any newly designed component can be placed in a component library for future use in various types of systems. This makes system modeling procedure faster and easier.

This approach makes it possible to include many necessary details of systems in our models. State transition tables are suitable for modeling sequential events, for which most of the other proposed methods do not offer a clear procedure.

Currently, we are working on implementation of this approach as a computer program and some sample component libraries, so that it can be applied for large scale and actual systems.

REFERENCES

[1] A. Carpignano and A. Poucet. (1994). Computer assisted fault tree construction: a review of methods and concerns. *Reliability Engineering and System Safety* [Online]. *44(3),* pp. 265–278. Available: http://cat.inist.fr/?aModele=afficheN&cpsidt=4241114

[2] Y. Wang, T. Teague, H. West, and S. Mannan. (2002, Jul.). A new algorithm for computer-aided fault tree synthesis. *Journal of Loss Prevention in Process Industries* [Online]. *15*, pp. 265–277. Available: www.ingentaconnect.com/content/els/09504230/2002/00000015/00000004/art00011

[3] K. K. Vemuri and J. B. Dugan. (1999, Dec.). Automatic synthesis of Fault Trees for Computer-Based Systems. *IEEE Trans. Reliab.* [Online]. *48*, pp. 394–402. Available: http://ieeexplore.ieee.org/stamp/stamp.jsp?arnumber=00814522

[4] C. Chuei-Tin and H. Her-Chuan. (1992). New developments of the digraph-based techniques for fault tree synthesis. *Ind. Eng. Chem. Res.*

[Online]. *31,* pp. 1490–1502. Available: http://pubs.acs.org/cgi-bin/archive.cgi/iecred/1992/31/i06/pdf/ie00006a010.pdf

[5] S. A. Lapp and G. J. Powers. (1977, Apr.). Computer-aided synthesis of fault-trees. *IEEE Trans. Reliab.* [Book]. *R-26*, pp. 2–13.

[6] Y. Wang, W. J. Rogers, H. H. West, M. S. Mannan. (2003, Sept.). Algorithmic fault tree synthesis for control loops. *Journal of Loss Prevention in Process Industries.* [Online]. *16*, pp. 427–441. Available: http://dx.doi.org/10.1016/S0950-4230(03)00043-3

[7] N. H. Ulerich and G. J. Powers. (1988, Jan.). On-Line hazard aversion and fault diagnosis in chemical processes: The digraph + fault tree method. *IEEE Trans. Reliab.* [Online]. *37*, pp. 171–177. Available: http://ieeexplore.ieee.org/stamp/stamp.jsp?arnumber=00003738

[8] J. D. Andrews and J. J. Henry, "A computerized fault tree construction methodology," in *Proc. of the Institution of Mechanical Engineers,* 1997; 211(E), pp. 171–183.

[9] J. J. Henry and J. D. Andrews. (1998, Dec.). Computerized fault tree construction for a train braking system. *Quality and Reliability Engineering International.* [Online]. *13(5)*, pp. 299–309. Available: http://www3.interscience.wiley.com/journal/10701/abstract

[10] S. L. Salem, et. al. (1977). A new methodology for the computer aided construction of fault tree. *Annals of Nuclear Energy* [Book]. *4,* pp. 417–433.

[11] A. Rauzy. (2002, Oct.). Mode Automata and their compilation into fault trees. *Reliability Engineering and System Safety.* [Online]. *78(1)*, pp. 1–12. Available: http://dx.doi.org/10.1016/S0951-8320(02)00042-X

[12] P. Liggesmeyer and M. Rothfelder, "Improving system reliability with automatic fault tree generation," in *Proc. 28th Ann. Int. Symp. Fault-Tolerance Computing,* Munich, 1998, pp. 90–99.

[13] J. R. Taylor. (1982, Jun.). An algorithm for fault-tree construction. *IEEE Trans. Reliab.* [Book]. *R-31*, pp. 137–146.

[14] H. Kumamoto, E. J. Henley. (1995). Automated fault tree synthesis by semantic network modeling, rulebased development and recursive 3-value procedure. *Reliability Engineering and System Safety.* [Online]. *49,* pp. 171–188. Available: http://dx.doi.org/10.1016/0951-8320(95)00029-2

[15] Y. Papadopoulos, "Marhun M. Model-Based Synthesis of Fault Trees from Matlab-Simulink models," in *Proc. Int. Conf. Dependable Systems and Networks (DSN'01).* Sweden, 2001, pp. 77–82.

[16] Y. Papadopoulos, J. McDermid, A. Mavrides, C. Scheidler, M. Maruhn. "Model-Based Semiautomatic Safety Analysis of Programmable Systems in Automotive Applications," in *Proc. ADAS 2001, Int. Conf. Advanced Driver Assistance System,* Birmingham, 2001, pp. 53–57.

[17] S. Mukharejee and A. Chakraborty, "Automated Fault Tree Generation: Bridging Reliability with Text Mining," in *Proc. Reliability and Maintainability Symp,* Orlando, FL, 2007, pp. 83–88.

[18] A. G. T. Raaphorst, B D. Netten, and R. A. Vingerhoeds. "Automated Fault-Tree Generation for Operational Fault Diagnosis," in *Proc. Int. Conf. Electric Railways in a United Europe,* Amsterdam, 1995, pp. 173–177.

[19] R. C. De Vries. (1990, Apr.). An Automated Methodology for Generating a Fault Tree. *IEEE Trans. Reliab.* [Online]. *39(1)*, pp. 76–86. Available: http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=52615

[20] *Fault Tree Handbook: NUREG-0492.* U.S. Nuclear Regulatory Commission. Washington, D. C. Jan. 1981.