*Transactions of the Korean Nuclear Society Spring Meeting*
*Jeju, Korea, May 7-8, 2015*

# New algorithm to detect modules in a fault tree for a PSA

Woo Sik Jung[a]
[a] *Sejong University, 209 Neungdong-Ro, Gwangjin-Gu, Seoul 143-747, South Korea*

## 1. Introduction

A module or independent subtree is a part of a fault tree whose child gates or basic events are not repeated in the remaining part of the fault tree. Modules are necessarily employed in order to reduce the computational costs of fault tree quantification. This paper presents a new linear time algorithm to detect modules of large fault trees. The size of cut sets can be substantially reduced by replacing independent subtrees in a fault tree with super-components.

Chatterjee [1] and Birnbaum [2] developed properties of modules, and demonstrated their use in the fault tree analysis. Locks [3] expanded the concept of modules to non-coherent fault trees. Independent subtrees were manually identified while coding a fault tree for computer analysis [4]. However, nowadays, the independent subtrees are automatically identified by the fault tree solver [5].

A Dutuit and Rauzy (DR) algorithm to detect modules of a fault tree for coherent or non-coherent fault tree was proposed in 1996 [6]. It has been well known that this algorithm quickly detects modules since it is a linear time algorithm.

## 2. Definitions and DR method [6]

A fault tree has terminal events (basic events), intermediate events (gates), and top event(s). A fault tree thus has logical interrelationships of basic events that lead to single or multiple top events. Basic events are elementary component failures, and top event is a system failure. The basic assumption of a fault tree is that basic events are mutually independent. Failures of components, that is, basic events are logically propagated to the top event through the nested logical gates. Fig. 1 depicts a small fault tree that is used throughout this paper. In this study, a single top event is assumed instead of multiple tops.

$Event_{in}(v)$ is a set of nodes (gates or basic events) that are reachable downward way from a node $v$ [6], and $Event_{out}(v)$ is further defined in this study as a set of nodes that are reachable from the root node (top event) without visiting a node $v$.

A module is defined as a gate whose terminal or intermediate events do not occur elsewhere in a fault tree [6]. In other words, a node $v$ is a module if there is no other downward way from the root node to any node in $Event(v)$ without visiting $v$. That is, a node $v$ is a module if the relation is satisfied

$$Event_{in}(v) \cap Event_{out}(v) = \Phi \qquad (1)$$

Top event $v$ is always a module since $Event_{out}(v)$ is an empty set.

Dutuit and Rauzy [6] proposed an efficient module detection algorithm (DR method) that is based on the depth-first leftmost traversal of a fault tree. The visiting steps to nodes are written along the connecting lines of nodes in Fig. 1. In this paper, the node index (numbers in the node names) is increased according to the first visiting order.

Visiting steps $Visit_{first}(v)$, $Visit_{second}(v)$, and $Visit_{last}(v)$ are introduced for the explanation of the DR method [6]. Additionally, the steps such as $Visit_{min}(v)$ and $Visit_{max}(v)$ are also defined as

$$Visit_{min}(v) = \min_{w \in Event_{in}(v)} Visit_{first}(w) \qquad (2)$$

$$Visit_{max}(v) = \max_{w \in Event_{in}(v)} Visit_{last}(w). \qquad (3)$$

In the DR method, a node $v$ is a module if its visiting steps satisfy the inequalities

$$Visit_{first}(v) < Visit_{min}(v) < Visit_{max}(v) < Visit_{second}(v). \qquad (4)$$

All visiting steps for nodes can be easily found by tracing the depth-first leftmost traversal of the fault tree in Fig. 1. Since visiting steps of {g0, g3, g6} satisfy the inequalities of the module definition in Eq. (4), they are modules of the fault tree in Fig. 1 as

$$Module(g0) = \{g0, g3, g6\}$$

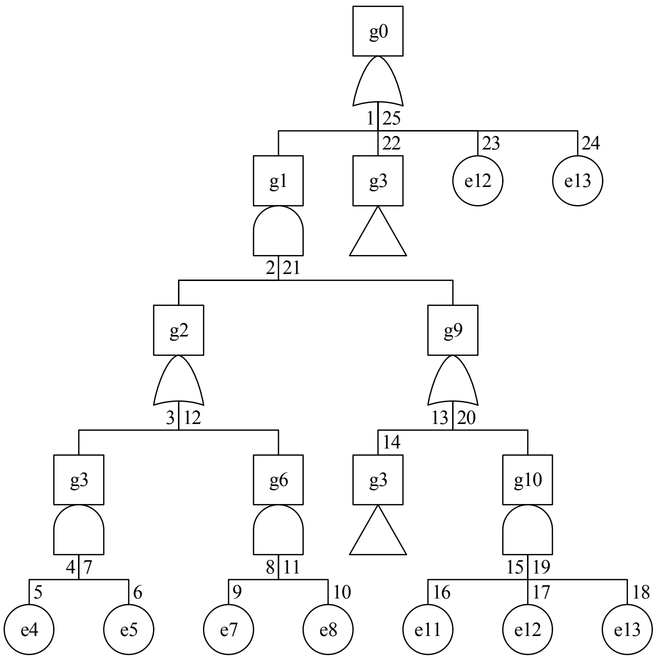

**Fig. 1. DR method to find modules [6]**

## 3. New algorithm to find modules

All nodes are visited along a depth-first leftmost traversal. The traversal starts and finally ends at the root node with zero module measures. For clear explanation of the new algorithm, the numbers in node names are increased along the traversal.

For efficient module detection, module measures are newly introduced in this study. In this study, repeated number of a node $v$ in a fault tree is defined in this study as $Count(v)$. When leaving a repeated node $v$ for the first time and going to next node $w$, module measures are increased one time as

$$P_{in/out}(w) = P_{out}(v) + (Count(v) - 1) \quad (5)$$

$$Q_{in/out}(w) = Q_{out}(v) + (Count(v) - 1)\alpha_v \ . \quad (6)$$

Whenever leaving this repeated node $v$ from the second time and going to next node w, module measures are decreased stepwise as

$$P_{in/out}(w) = P_{out}(v) - 1 \quad (7)$$

$$Q_{in/out}(w) = Q_{out}(v) - \alpha_v \ . \quad (8)$$

In Eqs. (6) and (8), $\alpha_v$ is an integer value as

$$\alpha_v = v \ . \quad (9)$$

In Eqs. (5) to (9), $P_{in/out}(w)$ is one of $P_{in}(w)$ and $P_{out}(w)$, and $Q_{in/out}(w)$ denotes one of $Q_{in}(w)$ and $Q_{out}(w)$.

If there are no changes in the module measures between entering and leaving a gate $v$

$$P_{in}(v) = P_{out}(v) \quad (10)$$

$$Q_{in}(v) = Q_{out}(v) \ , \quad (11)$$

the gate $v$ is a module of the fault tree.

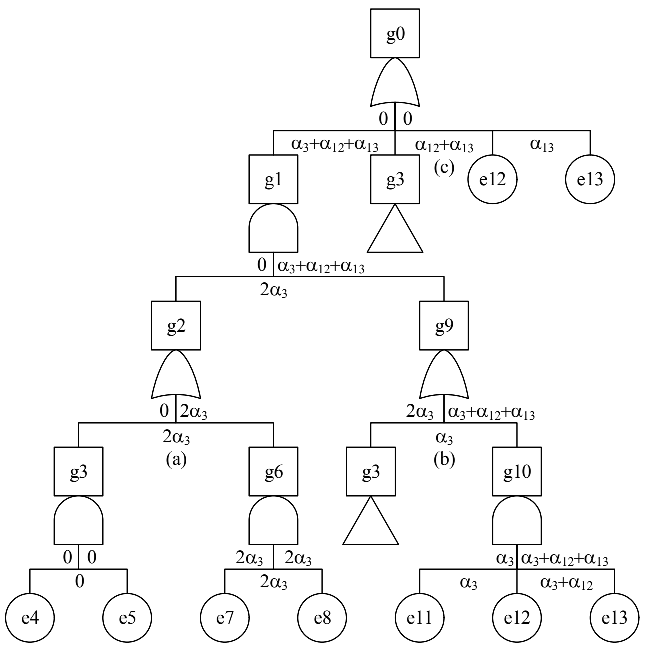

**Fig. 2. New method to find modules**

Changes of module measures $Q_{in}(v)$ and $Q_{out}(v)$ are illustrated in Fig. 2. Furthermore, their variations along the traversal in a whole fault tree are depicted in Fig. 2 and listed in Table 1. Module measures are increased by Eqs. (5) and (6) one time when leaving repeated nodes such as {g3, e12, e13} for the first time, and decreased stepwise by Eqs. (7) and (8) whenever leaving repeated nodes such as {g3, e12, e13} the other time. However, there are no changes in module measure when leaving non-repeated nodes.

**Table 1. Module identification**

| Node $v$ | $Q_{in}(v)$ | $Q_{out}(v)$ | $P_{in}(v)$ | $P_{out}(v)$ | Module |
|---|---|---|---|---|---|
| g0 | 0 | 0 | 0 | 0 | Yes |
| g1 | 0 | $\alpha_3 + \alpha_{12} + \alpha_{13}$ | 0 | 3 | No |
| g2 | 0 | $2\alpha_3$ | 0 | 2 | No |
| g3 | 0 | 0 | 0 | 0 | Yes |
| e4 | 0 | 0 | 0 | 0 | |
| e5 | 0 | 0 | 0 | 0 | |
| g6 | $2\alpha_3$ | $2\alpha_3$ | 2 | 2 | Yes |
| e7 | $2\alpha_3$ | $2\alpha_3$ | 2 | 2 | |
| e8 | $2\alpha_3$ | $2\alpha_3$ | 2 | 2 | |
| g9 | $2\alpha_3$ | $\alpha_3 + \alpha_{12} + \alpha_{13}$ | 2 | 3 | No |
| g10 | $\alpha_3$ | $\alpha_3 + \alpha_{12} + \alpha_{13}$ | 1 | 3 | No |
| e11 | $\alpha_3$ | $\alpha_3$ | 1 | 1 | |
| e12 | $\alpha_3$ | $\alpha_3$ | 1 | 1 | |
| e13 | $\alpha_3 + \alpha_{12}$ | $\alpha_3 + \alpha_{12}$ | 2 | 2 | |

## 4. Conclusions

The new algorithm minimizes computational memory and quickly detects modules. Furthermore, it can be easily implemented into industry fault tree solvers that are based on traditional Boolean algebra, binary decision diagrams (BDDs), or Zero-suppressed BDDs.

The new algorithm employs only two scalar variables in Eqs. (5) to (8) that are volatile information. After finishing the traversal and module detection of each node, the volatile information is destroyed. Thus, the new algorithm does not employ any other additional computational memory and operations. It is recommended that this method be implemented into fault tree solvers for efficient probabilistic safety assessment (PSA) of nuclear power plants.

## REFERENCES

[1] P. Chatterjee, "Modularization of fault trees: A method to reduce the cost of analysis", Reliability and Fault Tree Analysis, SIAM, pp 101-137, 1975.
[2] Z.W. Birnbaum and J. P. Esary, "Modules of Coherent Binary Systems," SIAM J Appl. Math., Vol. 13, pp. 442–462, 1965.
[3] O.M. Locks, "Modularizing, Minimizing, and Interpreting the K & H Fault-Tree," IEEE Transactions on Reliability, Vol. R-30, no. 5, pp. 411–415, 1981.
[4] U.S. NRC, Reactor safety study - an assessment of accident risk in U.S. commercial nuclear power plants, WASH-1400, NUREG-75/014, 1975.
[5] W.S. Jung, "ZBDD algorithm features for an efficient Probabilistic Safety Assessment," Nuclear Engineering and Design, Vol. 239, pp. 2085–2092, 2009.
[6] Y. Dutuit and A. Rauzy, "A Linear Time Algorithm to Find Modules of Fault Trees," IEEE Transactions on Reliability, Vol. 45, no. 3, pp. 422–425, 1996.