ELSEVIER

PII: S0951-8320(96)00142-1

Reliability Engineering and System Safety **56** (1997) 101–105

0951-8320/97/$17.00

Analytic method to break logical loops automatically in PSA

Joon-Eon Yang, Sang-Hoon Han, Jin-Hee Park & Young-Ho Jin

Advanced Research Group, Korea Atomic Energy Research Institute, P.O. Box 105, Yusung-Gu, Taejeon, 305-600, Korea

(Received 12 December 1996)

In Probabilistic Safety Assessment (PSA), when the fault trees of support systems are merged, logical loops are generated due to the mutual dependencies of support systems. KAERI (Korea Atomic Energy Research Institute) has developed a method to break such logical loops automatically. The developed method provides the criteria to identify parts of the fault trees that cause the logical loops. Using a top-down approach, the logic of fault tree is expanded. During the expansion, the terms that cause the logical loops are identified based on the given criteria and are deleted automatically. By using the above procedure, we will get the new logic of fault trees without the logical loops. This method is implemented in KIRAP (KAERI Integrated Reliability Analysis Package) and tested for sample cases. The results obtained by the developed method are compared to ones obtained by the conventional method that has been used previously to break the logical loops in PSA. © 1997 Elsevier Science Limited.

1 INTRODUCTION

In Probabilistic Safety Assessment (PSA), when the fault trees of systems are merged, logical loops are generated due to the mutual dependencies, especially among the support systems such as service water system, instrument air system and electric power system, etc. For instance, a Component Cooling Water System (CCWS) provides cooling water to a Diesel Generator (DG), and the DG provides electric power to the CCWS. Such logical loops should be broken for the accident sequence quantification (ASQ) process of PSA. Up to now, the logical loop problem is solved by breaking the logical loops at the points where the dependencies among the support systems are relatively weak and developing new fault trees without the logical loops [1, 2]. This conventional method, however, has some drawbacks such as: (1) it is time consuming and difficult to review the relations among the support systems and to develop new fault trees without logical loops, (2) some minimal cut sets are not generated due to the arbitrary break of dependencies among the support systems.

KAERI (Korea Atomic Energy Research Institute) has developed a method to break the logical loops automatically. The developed method provides the criteria to identify parts of the fault trees that cause the logical loops. Using a top-down approach, the logic of the fault tree is expanded. During expansion, the terms that cause the logical loops are identified based on given criteria and are deleted automatically. By the above procedure, we will get new logic of fault trees without the logical loops.

Basically, the conventional and developed methods are the same in that both methods delete some parts of fault tree logic in order to break the logical loops. The developed method, however, gives us the exact criteria, which are based on Boolean equations, to identify parts of the fault trees that cause the logical loops. Such parts are deleted automatically during the expansion of fault tree logic while maintaining the proper relations among the support systems. This method, therefore, enables us to save man-power to review the dependencies among the support systems and to develop the new logic of fault trees without the logical loops. It also ensures complete results without missing terms.

The developed method is implemented in KIRAP (KAERI Integrated Reliability Analysis Package) [3] and tested for some fault trees with the logical loops. The generated minimal cut sets are compared to ones obtained by the conventional method.

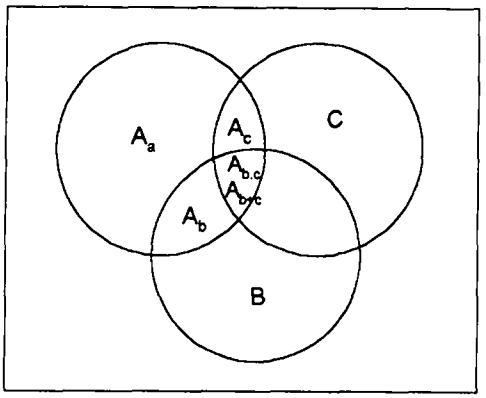

Fig. 1. The relations among three support systems: A, B and C.

2 ANALYTIC METHOD TO BREAK THE LOGICAL LOOP

2.1 Boolean expression of fault tree logic with the logical loops

For instance, let's assume that there are three support systems; A, B and C. The relations among these support systems are shown in Fig. 1.

The failure of support system A, then, can be represented by the following Boolean equation:

$$A = A_a + A_b.B + A_c.C, \tag{1}$$

where

A a set representing the failure of system A,
A_a a subset representing the failures of system A's part that cause the failure of system A without the simultaneous failure of system B or C,
A_b a subset representing the failures of system A's part that cause the failure of system A when the simultaneous failure of system B occurs,
A_c a subset representing the failures of system A's part that cause the failure of system A when the simultaneous failure of system C occurs,
$+$ OR operation and
$.$ AND operation.

However, in some cases, there can be some common parts between the subset A_b and A_c. So, to make sure that there are no common parts between A_b and A_c, we restructure above equation as follows:

$$A = A_a + A_b.B + A_c.C + A_{b+c}.(B + C) + A_{b.c}.B.C,$$

where

A_{b+c} a subset representing the failures of system A's part that cause the failure of system A when the failure of system B or C occurs,
$A_{b.c}$ a subset representing the failures of system A's part that cause the failure of system A when the failures of system B and C occur at the same time.

Hence, the following constraint can be applied to A_a, A_b, A_c, A_{b+c} and $A_{b.c}$:

$$A_i.A_j = \Phi(\text{the empty set})$$

$$(i,j = a,b,c,b + c,b.c; i \neq j). \tag{2}$$

This constraint means that there are no common parts among the subset A_b, A_c, A_{b+c} and $A_{b.c}$.

In the same way, the failure of systems B and C can be represented by the following Boolean equations, respectively:

$$B = B_b + B_a.A + B_c.C + B_{a+c}.(A + C) + B_{a.c}.A.C, \tag{3}$$

$$C = C_c + C_a.A + C_b.B + C_{a+b}.(A + B) + C_{a.b}.A.B. \tag{4}$$

2.2 Analytic method to break the logical loops automatically

Let us call subsets such as A_a, B_b and C_c basic event sets, and subsets such as A_b, A_c, A_{b+c}, $A_{b.c}$, etc. conditional event sets. The basic event set of each system represents the failures of each system's main parts that cause the failure of each system without the simultaneous failure of related support systems, e.g., the motor failure of pumps. The conditional event set represents the failure of system's part that can cause the failure of that system when the related support system fails simultaneously, e.g., the simultaneous failures of instrument air system and local backup air tank. The failure of a system can be caused by the element of a basic event set or when the elements of conditional event set are combined with those of other support systems' basic event set.

If we expand eqns (1), (3) and (4) for system A, we will get many terms with the form shown below:

$$A_i.[B \text{ or } C].A. \tag{5}$$

where

A_i an element of conditional event set of system A, e.g., A_b or A_c or A_{b+c} or $A_{b.c}$,
$[B$ or $C]$ the conditional event set for the system B or C that appears on the right hand side of eqns (3) and (4).

Equation (5) represents the relation that a failure of system A causes the failure of the system B or C, then this causes the failure of the system A again, that is, the logical loop. For the case where A_i is a basic event, e.g., A_a, this term is reduced by a Boolean reduction rule $X + X.Y = X$ [4] since A_a already exists in eqn (1).

Let us expand the above equation for a case where

A_i is A_b:

$$A_b.[B \text{ or } C].A = A_b.(A_a + A_b.B + A_c.C + A_{b+c} .(B + C) + A_{b.c}.B.C).[B \text{ or } C] \quad (6)$$

$$= (\underline{A_b.A_a} + A_b.B + \underline{A_b.A_c}.C + \underline{A_b.A_{b+c}} .(B + C) + A_b.A_{b.c}.B.C).[B \text{ or } C]$$

(the underlined terms become Φ by eqn (2))

$$= A_b.B.[B \text{ or } C]$$

$$\therefore \quad A_b.[B \text{ or } C].A = A_b.[B \text{ or } C].B$$

The above Boolean equation is valid only for the following three cases since we assume that A and B are different systems:

i) $$A_b = \Phi, \quad (7)$$

ii) $$[B \text{ or } C] = \Phi \text{ and} \quad (8)$$

iii) $$A_b.[B \text{ or } C] = \Phi \text{ or } A_i.[B \text{ or } C].A = \Phi. \quad (9)$$

For the first and second cases, these terms would not be generated during the expansion. Therefore, if we get these kinds of terms while we expand the above Boolean equations, only the relation of the third case is correct. So, in such a case, we do not need to expand these terms, we just need to delete them.

Boolean equations for m support systems can be expressed as follows:

$$n_i = n_i^* + \Sigma_j \text{Cond}_i(j) f_{ij}(n_1, n_2, \ldots, n_{i-1}, n_{i+1}, \ldots, n_m) \quad (10)$$

$$n_j = n_j^* + \Sigma_k \text{Cond}_j(k) f_{jk}(n_1, n_2, \ldots, n_{j-1}, n_{j+1}, \ldots, n_m) \quad (11)$$

where

n_i	a set representing the failure of the i-th support system,
n_i^*	a basic event set of the i-th support system,
$\text{Cond}_i(j)$	the j-th conditional event set of the i-th support system and
$f_{ij}(n_1, n_2, \ldots, n_{i-1}, n_{i+1}, \ldots, n_m)$	the combination of n_1, n_2, ..., n_{i-1}, n_{i+1}, ..., n_m corresponding to the j-th conditional event set.

Since the intersection of each conditional event is an empty set as shown in the above example, when we expand the above equations for n_i, we will get terms like the following:

$$\text{Cond}_i(k).[\text{terms of } n_{j \neq i}].n_i \quad (12)$$

$$= \text{Cond}_i(k).[\text{terms of } n_{j \neq i}].\{n^*_i + \Sigma_j \text{Cond}_i(j) f_{ij}(n_1, n_2, \ldots, n_{i-1}, n_{i+1}, \ldots, n_m)\}$$

$$= \text{Cond}_i(k).[\text{terms of } n_{j \neq i}].n_k. \quad (13)$$

The term $\text{Cond}_i(k).[\text{terms of } n_{j \neq i}].n_i$ should be an empty set by the same reason as above.

The logic of the original fault tree is expanded by using a top-down approach. During the expansion, we can use eqn (13) as the criteria to decide which terms cause the logical loops and should be deleted. By expanding and deleting the logic of the fault trees, we will get the new logic of the fault trees without the logical loops while maintaining the proper relations among the support systems. These new fault trees are used to generate the minimal cut sets for the support systems. All minimal cut sets should include at least one of the basic event set and a combination of the conditional event set.

2.3 Implementation of developed method in KIRAP

The developed method is implemented in KIRAP. An example used to get the new logic of the fault tree is shown in Figs 2 and 3. Let us think about a case of three support systems. In this example, the relations among them are represented by the following Boolean equations that are the simplified forms of eqns (1), (3) and (4).

$$A = A_a + A_b.B + A_c.C, \quad (14)$$

$$B = B_b + B_a.A + B_c.C, \quad (15)$$

$$C = C_c + C_a.A + C_b.B. \quad (16)$$

In the developed method, a fault tree is expanded by the top-down approach. During the expansion, it traces the combinations of events and deletes any combinations that cause the logical loops. Figure 2 shows how to expand a fault tree for a gate A that has the logical loops.

In Fig. 2, a combination of events that includes an octagonal event causes the logical loop. There are 6 combinations that cause the logical loops; A-B-C-A, A-B-C-B, A-B-A, A-C-B-C, A-C-B-A, and A-C-A (the underlined parts cause the logical loops). Those parts are to be deleted in the fault tree, and the fault tree is restructured as shown in Fig. 3. By solving this restructured fault tree, we will get the minimal cut sets for the system A.

3 COMPARISON BETWEEN THE CONVENTIONAL AND DEVELOPED METHODS

When support systems are few, there would be no difference between the results obtained by the

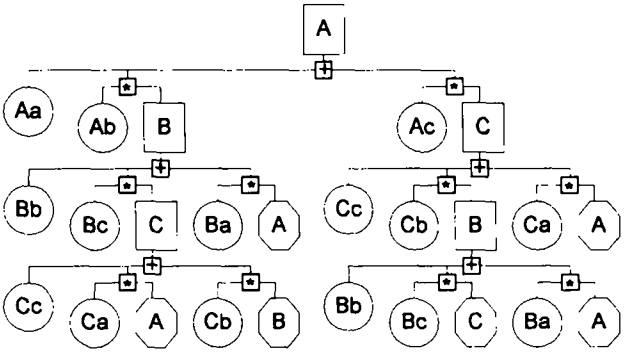

Fig. 2. Fault tree expansion to solve a fault tree with the logical loops.

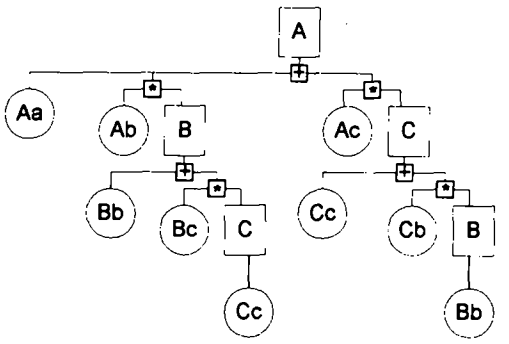

Fig. 3. The fault tree without the logical loops.

conventional and developed methods; we will get the same results by using both methods. If the number of support systems, however, becomes greater than some criteria e.g., three systems in this example, then there are some missed terms in the result obtained by the conventional method. So let us think about four support systems that have a relation like that shown in Fig. 4, that is, the most general relations among four systems.

For simplicity, let's assume that there are no parts like A_{b+c} or $A_{b.c}$. The relations among the support systems, then, can be represented by Boolean equations as follows:

$$A = A_a + A_b.B + A_c.C + A_d.D \tag{17}$$

$$B = B_b + B_a.A + B_c.C + B_d.D \tag{18}$$

$$C = C_c + C_a.A + C_b.B + C_d.D \tag{19}$$

$$D = D_d + D_a.A + D_b.B + D_c.C \tag{20}$$

where

A_a, B_b, C_c and D_d the basic event set of support systems A, B, C and D, respectively,

Xy the conditional event set that causes the failure of the system X with the simultaneous failure of system Y ($X \neq Y$).

We will solve this problem in two ways: the conventional and developed methods.

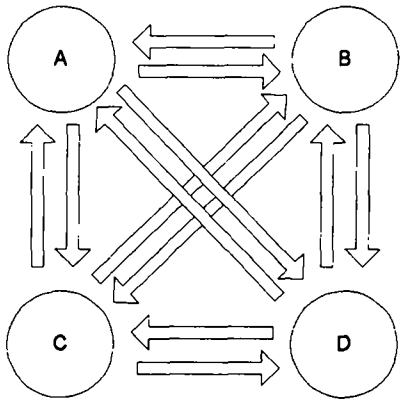

Fig. 4. An example of general logical loop among four support systems; A, B, C and D.

3.1 The conventional method

In the conventional method, the logical loop should be broken at some points. In this case, all four systems support each other, therefore, it should be broken at four points. Boolean equations without logical loops for four systems can be written as:

$$A^* = A_a \tag{21}$$

$$B^* = B_b \tag{22}$$

$$C^* = C_c \tag{23}$$

$$D^* = D_d. \tag{24}$$

Then we develop new Boolean equations, that is, the new fault trees that will be used in ASQ, without the logical loops, by using the above equations:

$$\begin{aligned} A_{bk} &= A_a + A_b.B^* + A_c.C^* + A_d.D^* \\ &= A_a + A_b.B_b + A_c.C_c + A_d.D_d \end{aligned} \tag{25}$$

$$\begin{aligned} B_{bk} &= B_b + B_a.A^* + B_c.C^* + B_d.D^* \\ &= B_b + B_a.A_a + B_c.C_c + B_d.D_d \end{aligned} \tag{26}$$

$$\begin{aligned} C_{bk} &= C_c + C_a.A^* + C_b.B^* + C_d.D^* \\ &= C_c + C_a.A_a + C_b.B_b + C_d.D_d \end{aligned} \tag{27}$$

$$\begin{aligned} D_{bk} &= D_d + D_a.A^* + D_b.B^* + D_c.C^* \\ &= D_d + D_a.A_a + D_b.B_b + D_c.C_c. \end{aligned} \tag{28}$$

From the above equations, we will get the new Boolean equation for the system A:

$$\begin{aligned} A &= A_a + A_b.B_{bk} + A_c.C_{bk} + A_d.D_{bk} \\ &= A_a + A_b.(B_b + B_a.A_a + B_c.C_c + B_d.D_d) \\ &\quad + A_c.(C_c + C_a.A_a + C_b.B_b + C_d.D_d) \\ &\quad + A_d.(D_d + D_a.A_a + D_b.B_b + D_c.C_c) \\ &= A_a + A_b.B_b + A_c.C_c + A_d.D_d \\ &\quad + A_b.(B_a.A_a + B_c.C_c + B_d.D_d) \\ &\quad + A_c.(C_a.A_a + C_b.B_b + C_d.D_d) \\ &\quad + A_d.(D_a.A_a + D_b.B_b + D_c.C_c) \\ &= A_a + A_b.B_b + A_c.C_c + A_d.D_d + A_b.B_c.C_c \\ &\quad + A_b.B_d.D_d + A_c.C_b.B_b + A_c.C_d.D_d \\ &\quad + A_d.D_b.B_b + A_d.D_c.C_c \end{aligned} \tag{29}$$

A Boolean reduction rule, $X + X.Y = X$, is applied in the above process. Each term of eqn (29) represents the minimal cut sets for the system A generated by the conventional method.

3.2 The analytic method

In the developed method, the Boolean eqn (17) is expanded by a top-down approach, that is, at first, B is expanded:

$$\begin{aligned} A &= A_a + A_b.(B_b + B_a.A + B_c.C + B_d.D) + A_c.C \\ &\quad + A_d.D = A_a + (A_b.B_b + \underline{A_b.B_a.A} + A_b.B_c.C \\ &\quad + A_b.B_d.D) + A_c.C + A_d.D. \end{aligned} \tag{30}$$

By eqn (9), the third term (the underlined one), $\underline{A_b.B_a.A}$, is deleted, after that, the fourth term, $\underline{A_b.B_c.C}$, is expanded:

$$\begin{aligned} A &= A_a + A_b.B_b + A_b.B_c.(C_c + C_a.A + C_b.B + C_d.D) \\ &\quad + A_b.B_d.D + A_c.C + A_d.D. \\ &= A_a + A_b.B_b + A_b.B_c.C_c + \underline{A_b.B_c.C_a.A} \\ &\quad + \underline{A_b.B_c.C_b.B} + A_b.B_c.C_d.D + A_b.B_d.D \\ &\quad + A_c.C + A_d.D. \end{aligned} \tag{31}$$

The underlined fourth and fifth terms are deleted and the sixth term will be expanded. By repeating this procedure, finally, we will get the following result:

$$\begin{aligned} A &= A_a + A_b.B_b + A_c.C_c + A_d.D_d + (A_c.C_b + A_d.D_b) \\ &\quad .B_b + (A_b.B_c + A_d.D_c).C_c + (A_b.B_d + A_c.C_d).D_d \\ &\quad + (A_d.D_c.C_b + A_c.C_d.D_b).B_b \\ &\quad + (A_d.D_b.B_c + A_b.B_d.D_c).C_c \\ &\quad + (A_c.C_b.B_d + A_b.B_c.C_d).D_d \end{aligned} \tag{32}$$

The same Boolean reduction rule is applied in the above process as in the conventional method.

Comparing eqns (29) and (32), we find that following six terms are not shown in eqn (29):

$$A_d.D_c.C_b.B_b, A_c.C_d.D_b.B_b, A_d.D_b.B_c.C_c, A_b.B_d.D_c.C_c,$$
$$A_c.C_b.B_d.D_d, A_b.B_c.C_d.D_d.$$

These terms represent the missed minimal cut set from the result obtained by the conventional method due to the arbitrary break of dependencies among the support systems. These terms can cause the quantitative difference in the ASQ result. The number of differences, however, depends upon the value of each term. Generally, these terms are the multiplication of several events, so the value would be very small.

4 DISCUSSIONS AND CONCLUSIONS

KAERI has developed a method to break the logical loops automatically and to generate new fault trees without logical loops. A top-down approach is used to automatically generate new logic of fault trees without logical loops. The terms that cause the logical loops are identified during the expansion and are deleted.

Basically, the conventional and developed methods are the same in that both methods delete some parts of fault tree logic in order to break the logical loops. The developed method, however, gives us the exact criteria, which are based on Boolean equations, to identify parts of fault trees that cause the logical loops.

Since the new logic of the fault trees is generated automatically by using this method, we can, therefore, reduce man-power to review the dependencies among the support systems and to develop the new fault trees without the logical loops. This method also enables us to get complete minimal cut sets from the fault trees of support systems without paying attention to breaking the logical loops among these systems.

The missed terms may not affect the reliability values of support systems greatly since these terms are the multiplication of several events, so the values of these terms would be negligible. We might, however, miss some terms that show a critical relation among some support systems. For instance, if we assume that A_c, A_d and B_d of eqn (29) are empty sets, the result obtained by the conventional method becomes:

$$A = A_a + A_b.B_b + A_b.B_c.C_c. \tag{33}$$

Under the same assumptions, the results by the developed method, i.e., eqn (30) becomes:

$$A = A_a + A_b.B_b + A_b.B_c.C_c + A_b.B_c.C_d.D_d. \tag{34}$$

Comparing the above results, we can see that the conventional method can not show the relation between the systems A and D, i.e., we might miss the fact that the failure of the system D can cause the failure of the system A when we use the conventional method. In this case, even if we use the conventional method, $A_b.B_c.C_d.D_d$ could be generated if we break the logical loops between the systems A, B, C and D differently from the way of Section 3.1. However, if we use the developed method, we do not need to pay our attention to this kind of problem.

The drawback of the developed method is that the size of the restructured fault tree logic becomes larger than that of the conventional method. It, therefore, requires more computational time.

This method is implemented in KIRAP that is developed and used in KAERI for the level 1 PSA. The developed method is tested for some real fault trees with the logical loops. The results show that the developed method gives us the expected results.

REFERENCES

1. USNRC, Interim Reliability Evaluation Program Procedure Guide, NUREG/CR-1278, 1984.
2. Coles, G. A. and Powers, T. B., Breaking the logic loop to complete the probabilistic risk assessment. In *Proceedings of PSA* 89: *International Topical Meeting Probability, Reliability, and Safety Assessment*, Pennsylvania, USA, x-y month 1989, pp. 1155–1160.
3. Han, Sang-Hoon, Kim, T. W., Jeong, K. S. and Yoo, K. J., PC-workstation based level 1 PRA code package—KIRAP. *Reliability and System Engineering*, 1990, **30**, 313–322.
4. McCormick, N. J., *Reliability and Risk Analysis: Methods and Nuclear Power Applications*. Academic Press, NY, 1981.