Strength of ZBDD Algorithm for the Post Processing of Huge Cutsets in Probabilistic Safety Assessment

Woo Sik Jung[a] and Jeff Riley[b]
[a] Sejong University, 209 Neungdong-Ro, Gwangjin-Gu, Seoul, South Korea
[b]Electric Power Research Institute, 3420 Hillview Avenue, Palo Alto, CA, USA

Abstract: Zero-suppressed Binary Decision Diagram (ZBDD) algorithm was an important variation of a BDD algorithm, since it quickly solves a very large fault tree with a very low truncation limit when performing a Probabilistic Safety Assessment (PSA) of a nuclear power plant. The ZBDD algorithm can generate huge cutsets with a very small truncation limit for the accurate core damage frequency (CDF) calculation. Furthermore, the ZBDD algorithm can perform cutset post-processing that is called as a cutset recovery in a PSA.

This paper explains the strength of the ZBDD algorithm for the efficient cutset post-processing. The post-processing of huge cutsets was performed and compared by FTREX and QRECOVER. Test results show that the ZBDD algorithm can play an important role for performing an efficient PSA by providing significantly reduced cutset quantification and processing time.

Keywords: PSA, Cutset post-processing, Zero-suppressed Binary Decision Diagram

1. INTRODUCTION

1.1. Cutset Post-processing in a PSA

The accident sequence cutsets that is called as Minimal Cut Sets (MCSs) can be post-processed for a realistic analysis of the accident sequences. This cutset post-processing is manually or automatically performed in order (1) to delete a physically impossible cutset that has mutually exclusive events and (2) to take into account recovery actions in a cutset and dependencies among the recovery actions. Generally, a rule-based cutset post-processed is performed to handle many cutsets. The cutset post-processing is automatically performed by applying the set of rules to cutsets with a specialized tool.

Two or more events are said to be mutually exclusive if the occurrence of any one of them excludes the occurrence of the others. The term mutually exclusive events refer to two or more basic events that appear in a single cutset which logically should not appear together. Generally, mutually exclusive events should not appear together in a single cutset for one of two reasons. First, plant operating restrictions such as Technical Specifications may prevent two components from being out of service at the same time. Second, other general logic modeling concerns may lead the analyst to remove specific combinations of events. During the PSA logic modeling phase, the analyst may or may not recognize that certain combinations of mutually exclusive events will appear in the same cutset. A PSA analyst could refine a fault tree model so that mutually exclusive events will not appear in the same cutset. However, some unrecognized mutually exclusive events may not be evident until the analyst generates and evaluates the system or sequence cutsets. Thus, the post-processing to delete physically impossible cutsets is inevitable.

In order to model the PSA accident sequences as accurately as practical, operator actions that could prevent the accident sequence are reflected to cutsets by using recovery events. The recovery events represent the probability that the operator or operators fail to successfully prevent the accident by restoring one or more of the failed components in the cutsets. The recovery events are frequently called as non-recovery probability events. In a PSA, operator actions that could prevent an accident sequence may not be specifically included in the logic models. To model the accident sequences as accurately as practically possible, a reliability analyst applies recovery events to the appropriate cutsets. These recovery events denote the failure of a recovery action.

1.2. Problems in the Cutset Post-processing

The cutset post-processing is usually performed with a separate tool after the cutset generation from the fault tree. Most PSA tools perform post-processing of cutsets that are generated by the independent fault tree solves. However, the required time for the huge cutset post-processing frequently exceeds the cutset generation time and, even worse, the processing of millions of cutsets frequently fails due to lack of memory.

In order to reduce long cutset post-processing time and to overcome the failure of huge cutset post-processing, the ZBDD-based cutset post-processing method was developed [1,2] and implemented into FTREX (Fault Tree Reliability Evaluation eXpert) [3]. FTREX had been developed using ZBDD algorithm that is for accomplishing the fast cutset generation. Furthermore, FTREX is being constantly revised and updated in order to support diverse cutset post-processing rules.

2. ZBDD ALGORITHM

2.1. ZBDD Algorithm Development

The ZBDD algorithm was a successful variation of the BDD algorithm and it quickly solves a large fault tree with a truncation limit. The ZBDD algorithm [3] and its software FTREX were developed to overcome a long run time and reduce huge memory requirements. In view of the short computation time and small memory usage of a ZBDD algorithm, the ZBDD algorithm is much more efficient than the cutset-based algorithms that are based on the traditional Boolean algebra.

The ZBDD is an efficient data structure that encodes cutsets as a factorized form of cutsets. The ZBDD is a Boolean structure that consists of recursively connected If-Then-Else (ITE) connectives. The ZBDD ITE is interpreted as

$$f = ite(x, f_1, f_0) = x \cdot f_1 + f_0 \ . \tag{1}$$

By optimally choosing the factorization order, that is, a ZBDD variable ordering, the ZBDD size can be minimized significantly.

A set of the special formulae for Boolean operations for two ZBDDs was developed by Jung [3]. A Boolean operation for two ZBDDs is performed by using the formulae in Eq. (2) below. If x and y are two variables with a variable ordering $x < y$, then the following equalities hold for coherent fault trees.

$$\begin{aligned}
&ite(x, L_1, R_1) \cdot ite(x, L_2, R_2) = ite(x, (L_1L_2 + L_1R_2 + R_1L_2), R_1R_2) \\
&ite(x, L_1, R_1) + ite(x, L_2, R_2) = ite(x, (L_1 + L_2), (R_1 + R_2)) \\
&ite(x, L_1, R_1) \cdot ite(y, L_2, R_2) = ite(x, L_1h, R_1h) \\
&ite(x, L_1, R_1) + ite(y, L_2, R_2) = ite(x, L_1, (R_1 + h))
\end{aligned} \tag{2}$$

where $h=ite(y,L_2,R_2)$. The ZBDD algorithm in Eq. (2) is for performing a Boolean operation of two ZBDDs.

In order to maintain minimal solutions in a ZBDD structure, that is, in order to maintain MCSs in a ZBDD, a subsuming is performed whenever a gate is solved by Eq. (2). The subsuming is recursively performed from the root ITE to the child ITE connectives by comparing the left and right ITE connectives. Let us consider recursive ITE connectives

$$\begin{aligned}
&F=ite(t, G, H) = t \cdot G + H \\
&G=ite(x, G_1, G_2) \\
&H=ite(y, H_1, H_2) \ .
\end{aligned} \tag{3}$$

In order to delete non-minimal cutsets in F, a subsuming operation $G\backslash H$ is performed. A subset is always located in the left ITE connectives due to the ZBDD ITE definition in Eq. (1). A cutset in G is deleted if H has its superset. Rauzy [4] proposed an efficient subsuming operation in Eq. (4).

$$\text{Subsume}(G,H) = G \setminus H = \begin{cases} ite(x, G_1 \setminus H, G_2 \setminus H) & , x < y \\ G \setminus H_2 & , x > y \\ ite(x, G_1 \setminus (H_1 \text{ or } H_2), G_2 \setminus H_2) & , x = y \end{cases} . \quad (4)$$

The term $G_1\backslash(H_1$ or $H_2)$ in the last case denotes that each cutset in G_1 is tested and deleted if H_1 or H_2 has its superset.

A coherent fault tree is solved with a truncation limit in a bottom-up way by using Eqs. (2) and (4). The sum of cutset probabilities is calculated by recursively calculating the following equation in a bottom-up way in the ZBDD.

$$p_x \times p(f_1) + p(f_0). \quad (5)$$

If a cutset probability is less than the truncation limit, it is deleted during the ZBDD calculation.

2.2. ZBDD Algorithm Implementation

Boolean operations with expanded cutsets in a traditional Boolean algebra when (1) solving a fault tree, (2) performing a delete-term operation to handle negates, and (3) performing a cutset post-processing take a very long time, since all cutsets should be examined one by one and lots of comparisons should be performed.

However, FTREX is based on this ZBDD algorithm that is a replacement of the traditional Boolean algebra in the traditional fault tree solvers. The ZBDD can be interpreted as a factorized structure of cutsets. Since the ZBDD algorithm is based on the factorized form of cutsets, it uses much less memory than the cutset-based algorithm. By using the ZBDD algorithm in FTREX, a long run time for (1) solving a fault tree, (2) performing a delete-term operation to handle negates, and (3) performing a cutset post-processing of cutsets could be significantly reduced.

Due to the small memory requirement of the ZBDD algorithm in FTREX from solving a fault tree to performing a cutset post-processing, a much smaller truncation limit can be used than that in the cutset-based algorithm. By lowering the truncation limit, accurate PSA results such as a core damage frequency and importance measures could be calculated by the ZBDD algorithm in FTREX.

FTREX provides a significant improvement in the quantification speed for large PSA fault trees even with a small size memory. As an independent fault tree solver, FTREX can be used together with a number of PSA software packages and online risk monitoring systems. Currently, FTREX has an interface with two PSA tools AIMS [5] and EPRI R&R Workstation tools [2].

The following are important FTREX calculation procedures to solve a fault tree that has negates and to perform a cutset post-processing. Please note that a ZBDD structure is maintained during the whole calculation and it is expanded into the final cutsets at the last stage.

(1) Solve a fault tree in a bottom-up way. Here, negates are treated as basic events.
(2) Solve a top event with Eqs. (2) and (4). Here, negates are treated as basic events.
(3) Perform a delete-term procedure to the cutsets of top event with Eq. (4) in order to simulate negates.
(4) Perform a cutset post-processing with Eq. (4).
(5) Calculate a top event probability with Eq. (5).
(6) Transform the final ZBDD structure of a top event into expanded cutsets.

3. CUTSET POST-PROCESSING

3.1. Cutset Post-processing by ZBDD Algorithm

In a PSA, new cutsets that satisfy some conditions and exceptions are extracted from given cutsets, and then recovery actions are performed on the extracted cutsets. The typical recovery actions are an addition of recovery events, a replacement of some events, and an elimination of mutually exclusive events.

Let us illustrate a cutset post-processing with sample Boolean equations in Eqs. (6) to (8). They are general expressions for the cutset recovery in a PSA.

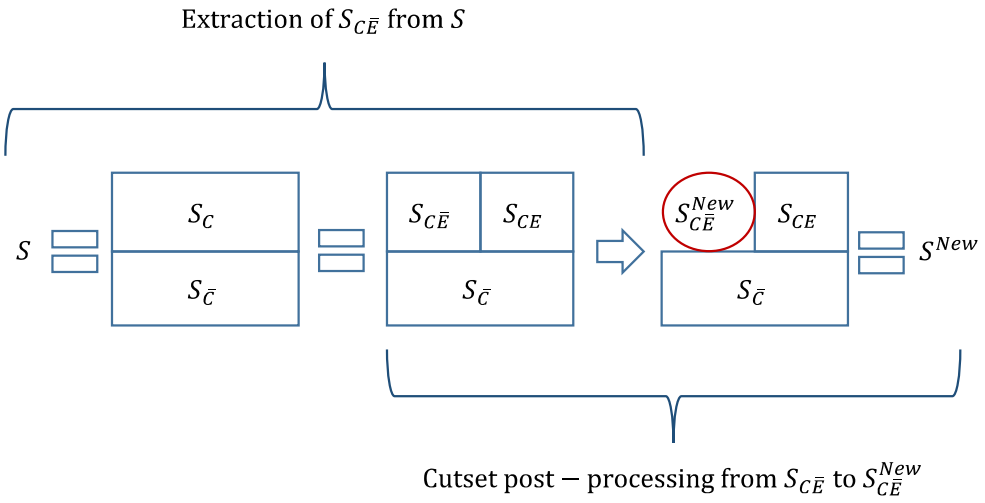
Cutset S (6)
Condition C (7)
. Exception E . (8)

In order to perform the cutset post-processing, a new Boolean equation $S_{C\bar{E}}$ that satisfies the conditions C and $\bar{E}$ in Eqs. (7) and (8) should be extracted from S in Eq. (6)

$$\text{Extracted Cutstes} = S_{C\bar{E}} \ . \tag{9}$$

If the Boolean equations in Eqs. (6) to (8) are expanded cutsets, all cutsets should be examined one by one and lots of comparisons should be performed in order to extract $S_{C\bar{E}}$ in Eq. (9). Thus, the actual extraction of $S_{C\bar{E}}$ in Eq. (9) takes a very long time. However, in this study, $S_{C\bar{E}}$ is extracted from S with Eqs. (6) to (8) in the form ZBDDs. Furthermore, the cutset post-processing is performed to the extracted cutsets $S_{C\bar{E}}$ with these equations.

A cutset post-processing should be performed with a new ZBDD that satisfies the conditions C and $\bar{E}$. The study by Jung [1] introduces a very simple cutset extraction method that is based on the ZBDD algorithm. If the cutset *S*, condition *C*, and exception *E* are encoded in the form of ZBDDs, the extraction of $S_{C\bar{E}}$ from S can be accomplished in a simple way by the ZBDD subsuming operation in Eq. (4). The extraction procedure is depicted in Fig. 1.



Cutset post-processing is performed on $S_{C\bar{E}}$

Figure 1. Cutset post processing scheme

First, the ZBDD structure S_C that satisfies the condition in Eq. (7) can be calculated by performing the subsuming operation in Eq. (4) twice

$$S_{\overline{C}} = S \setminus C \tag{10}$$

$$S_C = S \setminus S_{\overline{C}}. \tag{11}$$

Second, the ZBDD S_C that satisfies Eq. (7) could be further divided into two ZBDDs $S_{C\bar{E}}$ and S_{CE} by reflecting the exception in Eq. (8). A ZBDD $S_{C\bar{E}}$ that satisfies C and $\bar{E}$ is obtained by performing the subsuming operation $S_C \setminus E$ as

$$S_{C\overline{E}} = S_C \setminus E \tag{12}$$

$$S_{CE} = S_C \setminus S_{C\overline{E}}\ . \tag{13}$$

Since $S = S_C + S_{\overline{C}}$ and $S_C = S_{CE} + S_{C\overline{E}}$, the main ZBDD S in Eq. (6) can be expanded as

$$S = S_{\overline{C}} + \left(S_{C\overline{E}} + S_{CE}\right) = S_{C\overline{E}} + \left(S_{\overline{C}} + S_{CE}\right). \tag{14}$$

Third, a new ZBDD $S_{C\bar{E}}^{New}$ is calculated by performing cutset post-processing operations on $S_{C\bar{E}}$. Here, typical post-processing examples for $S_{C\bar{E}}^{New}$ and S^{New} are an addition of recovery events, a replacement of some events, and an elimination of mutually exclusive events [1].

Finally, a new S^{New} is obtained as a post-processed ZBDD.

$$S^{New} = S_{C\overline{E}}^{New} + \left(S_{\overline{C}} + S_{CE}\right)\ . \tag{15}$$

3.2. FTREX Cutset Post-processing

The advantage of FTREX recovery is reduced calculation time and minimized memory requirement since FTREX performs all calculations in a single run with a ZBDD that is a factored form of cutsets. Furthermore, FTREX does not write and read cutsets between cutset generation and cutset post-processing. As shown in Fig. 2, FTREX can generate cutstes or perform cutset post-processing in three ways [2]:

(1) FTREX generates cutsets from a fault tree and third party tool performs cutset post-processing,
(2) FTREX generates cutsets from a fault tree and directly performs cutset post-processing, or
(3) FTREX reads existing cutsets and performs cutset post-processing.

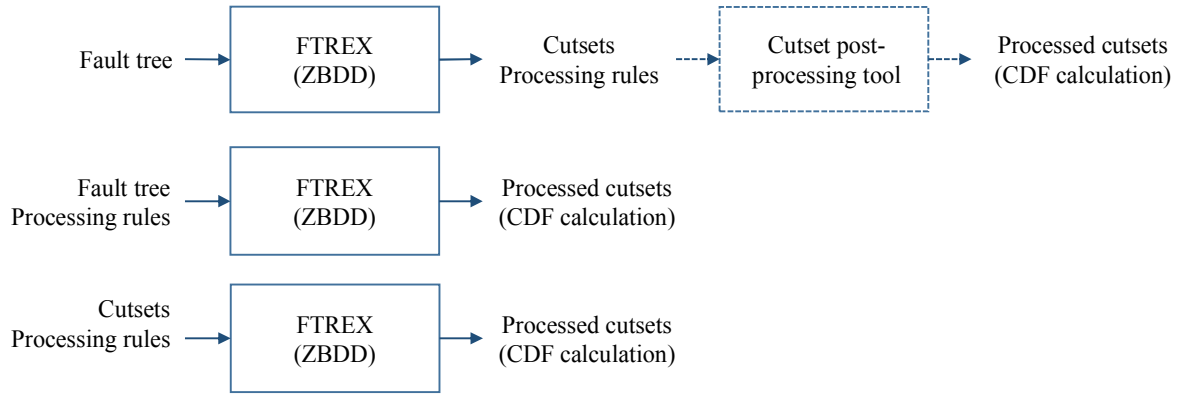


Figure 2. Three methods of cutset post-processing by FTREX

4. BENCHMARK TESTS

FTREX can perform cutset post-processing with diverse cutset post-processing rules in PRAQUANT, AIMS, SAREX, and SAPHIRE. In order to demonstrate the efficiency of the cutset post-processing by the ZBDD algorithm, (1) two PSA models were solved by FTREX, (2) cutset post-processing was performed by FTREX and QRECOVER, and (3) the cutset post-processing capabilities of FTREX and QRECOVER were compared. The test was performed with a personal computer (i3-4130 CPU, 9GB memory, and MS Windows 7 operating system).

As listed in Table 1, PSA1 and PSA2 models have 2,612 and 1,154 rules for the cutset post-processing, respectively. Since some Boolean components of the rules are complex Boolean equations, lots of cutsets are affected by the rules. Some recovery events have probabilities larger than one.

Table 1: Input models of PSA1 and PSA2

	PSA1	PSA2
Gates	4,555	5,586
Events	3,608	4,682
Negates	17	25
Complemented events	21	0
Initiators	66	59
Rules	2,612	1,154

The calculation results with various truncation limits are summarized in Tables 2 and 3 and depicted in Figs. 3 and 4. After performing the cutset post-processing, processed cutsets are truncated with given truncation limit. As shown in Tables 2 and 3, some processed cutsets are missing in QRECOVER calculation when these missing cutsets have event probabilities larger than one.

As shown in the results, the running time of FTREX for the cutset recovery is much smaller than that of QRECOVER. Furthermore, FTREX performance is much greater than that of QRECOVER in case of the low truncation limit, that is, huge cutsets. The high performance of FTREX has been accomplished by the ZBDD algorithm nature that maintains the small memory size by using factored cutstes.

Table 2: Cutset post-processing for PSA1 by FTREX and QRECOVER

Truncation limit	Cutset number	Cutset number after cutset post-processing		Cutset post-processing time (sec)	
		FTREX	QRECOVER	FTREX	QRECOVER
1.0E-09	4,358	631	630	0.8	2.3
1.0E-10	26,628	3,789	3,780	1.4	4.2
1.0E-11	75,117	17,849	17,804	4.3	12.5
1.0E-12	268,488	66,223	66,117	14.1	49.1
1.0E-13	897,077	233,984	233,579	50.5	224.0

Table 3: Cutset post-processing for PSA2 by FTREX and QRECOVER

Truncation limit	Cutset number	Cutset number after cutset post-processing		Cutset post-processing time (sec)	
		FTREX	QRECOVER	FTREX	QRECOVER
1.0E-08	160	100	100	0.7	0.9
1.0E-09	1,865	938	934	1.6	2.0
1.0E-10	17,136	7,233	7,198	7.4	12.6
1.0E-11	125,883	40,403	40,331	43.6	87.0
1.0E-12	914,237	238,596	238,247	238.4	646.8

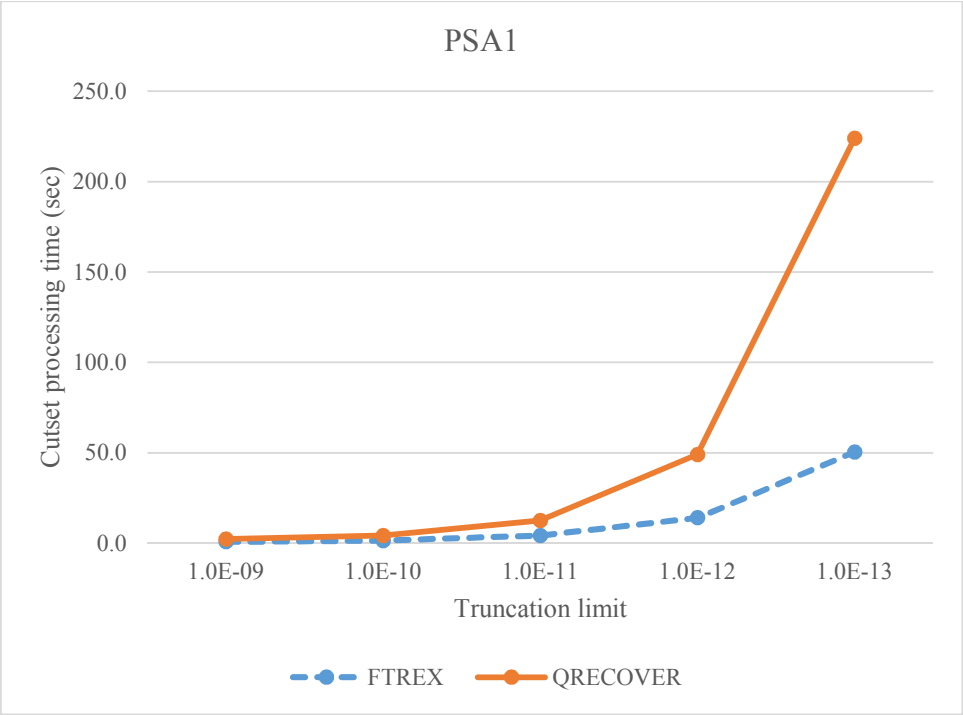


Figure 3. Cutset post processing for PSA1 by FTREX and QRECOVER

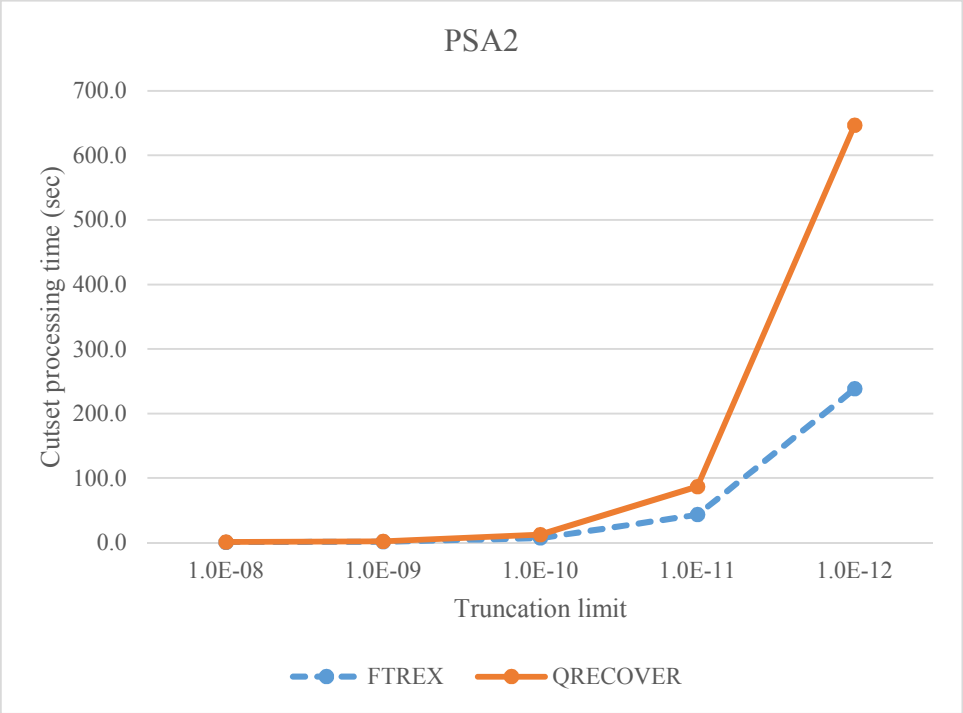


Figure 4. Cutset post processing for PSA2 by FTREX and QRECOVER

5. CONCLUSION

Boolean operations with expanded cutsets in the cutset-based algorithm take a very long time, since all cutsets should be examined one by one and lots of comparisons should be performed. On the other hand, since the ZBDD algorithm is based on the factorized form of cutsets, it consumes much less memory for the cutset generation and cutset post-processing than the traditional method that is using expanded

cutsets. Please note that the most efficient way to save memory is to maintain a ZBDD structure during the whole calculation stages and expand it just one time into the final cutsets at the last stage.

The cutset processing time of FTREX is much smaller than that of QRECOVER. Furthermore, FTREX performance is much greater than that of QRECOVER in case of huge cutsets. It results from the strength of the ZBDD algorithm programmed in FTREX.

The ZBDD algorithm nature and features that are explained in this study could be excellent solutions to overcome a long run time of the cutset-based algorithm for all calculation phases when (1) solving a fault tree, (2) performing a delete-term operation to simulate negates, and (3) performing a cutset post-processing of cutsets. Thus, it is strongly recommended to use the ZBDD algorithm or FTREX for generating cutsets and performing a cutset post-processing of cutsets.

References

[1] W.S. Jung, "ZBDD Algorithm Features for an Efficient Probabilistic Safety Assessment," Nuclear Engineering and Design, 239, 2085-2092, 2009.
[2] W.S. Jung, FTREX User Manual Version 1.7, Electric Power Research Institute, October 2013.
[3] W.S. Jung, S.H. Han, and J.J. Ha, "A Fast BDD Algorithm for Large Coherent Fault Trees Analysis," Reliability Engineering and System Safety, 83, 369–374, 2004.
[4] A. Rauzy, "New Algorithms for Fault Trees Analysis," Reliability Engineering and System Safety, 40, 203-211, 1993.
[5] S.H. Han, H.G. Lim, and J.E. Yang, “AIMS-PSA: A Software for Integrating Various Types of PSAs,” International Probabilistic Safety Assessment and Management Conference, Hong Kong, China, May 18-23.