Available online at www.sciencedirect.com
SCIENCE DIRECT®
Reliability Engineering and System Safety 83 (2004) 369–374
RELIABILITY ENGINEERING & SYSTEM SAFETY
www.elsevier.com/locate/ress

A fast BDD algorithm for large coherent fault trees analysis

Woo Sik Jung*, Sang Hoon Han, Jaejoo Ha

Korea Atomic Energy Research Institute, P.O. Box 105, Yuseong, Daejeon 305-600, South Korea

Received 4 June 2003; accepted 20 October 2003

Abstract

Although a binary decision diagram (BDD) algorithm has been tried to solve large fault trees until quite recently, they are not efficiently solved in a short time since the size of a BDD structure exponentially increases according to the number of variables. Furthermore, the truncation of If–Then–Else (ITE) connectives by the probability or size limit and the subsuming to delete subsets could not be directly applied to the intermediate BDD structure under construction. This is the motivation for this work.

This paper presents an efficient BDD algorithm for large coherent systems (coherent BDD algorithm) by which the truncation and subsuming could be performed in the progress of the construction of the BDD structure. A set of new formulae developed in this study for AND or OR operation between two ITE connectives of a coherent system makes it possible to delete subsets and truncate ITE connectives with a probability or size limit in the intermediate BDD structure under construction. By means of the truncation and subsuming in every step of the calculation, large fault trees for coherent systems (coherent fault trees) are efficiently solved in a short time using less memory. Furthermore, the coherent BDD algorithm from the aspect of the size of a BDD structure is much less sensitive to variable ordering than the conventional BDD algorithm.

Keywords: Binary decision diagram; Coherent fault tree

1. Introduction

Lee [1] and Akers [2] introduced a binary decision diagram (BDD) by representing Boolean functions as decision graphs. Bryant [3] popularized the use of the BDD by introducing a set of algorithms for the efficient construction and manipulation of the BDD structure. The BDD is used in a wide range of areas [4] including digital system design and reliability analysis. Coudert and Madre [5] and Rauzy [6] initiated the BDD application to the reliability analysis. Furthermore, the BDD has been investigated to solve large fault trees [7,8]. Way and Hsia [9] developed a special method to build a BDD encoding fault tree in a hybrid way. The size of a BDD structure depends critically on variable ordering and the determination of an appropriate variable ordering has a highly heuristic nature [10,11]. Rauzy [12] proposed an algebraic framework to perform approximate computations of minimal cut sets (MCSs) of fault tree.

The Shannon decomposition is succinctly defined in terms of the ternary If–Then–Else (ITE) connective

$$f = \mathrm{ite}(x, f_1, f_0) = xf_1 + \bar{x}f_0, \tag{1}$$

where x is one of decision variables. The functions f_1 and f_0 are Boolean functions evaluated at $x = 1$ and $x = 0$, respectively. The two resultant terms in the RHS of Eq. (1) are mutually exclusive. The recursive use of ITE connectives is the core of the BDD algorithm that provides an important alternative way of representing fault trees.

The Shannon decomposition in the form of the ITE connective is applicable to coherent or non-coherent systems. A system of components (a fault tree) is coherent if (a) its structure function (Boolean function) is increasing (non-decreasing) and (b) each component (basic event) is relevant ($f_1 \neq f_0$ for some x) [13]. A system is non-coherent if its Boolean function does not conform to the requirements of the coherency.

For example, the Boolean function for coherent or non-coherent systems can be represented as a linear function of an arbitrary variable x

$$f = a\bar{x} + bx + c, \tag{2}$$

where a, b, and c are Boolean functions of the other variables and some of them could be empty Boolean functions. If a is always an empty Boolean function for an arbitrary variable x, the resulting Boolean function

* Corresponding author. Tel.: +82-42-868-2764; fax: +82-42-861-2574.
E-mail address: woosjung@kaeri.re.kr (W.S. Jung).

0951-8320/$ - see front matter © 2003 Elsevier Ltd. All rights reserved.
doi:10.1016/j.ress.2003.10.009

$f = bx + c$ is for coherent systems. The ITE connective in Eq. (1) for the Boolean function in Eq. (2) becomes

$$\text{ite}(x, f_1, f_0) = x(b + c) + \bar{x}(a + c). \tag{3}$$

It could be easily proved that the decomposed Boolean function in Eq. (3) is identical to the original one in Eq. (2).

Let us explain the operation between two ITE connectives. If x and y are two variables with a variable ordering $x < y$, then the following equalities hold (conventional BDD algorithm in Ref. [6])

$$\text{ite}(x, G_1, G_2)\text{ite}(x, H_1, H_2) = \text{ite}(x, G_1H_1, G_2H_2) \tag{4}$$

$$\text{ite}(x, G_1, G_2) + \text{ite}(x, H_1, H_2) = \text{ite}(x, (G_1 + H_1), (G_2 + H_2)) \tag{5}$$

$$\text{ite}(x, G_1, G_2)\text{ite}(y, H_1, H_2) = \text{ite}(x, G_1h, G_2h) \tag{6}$$

$$\text{ite}(x, G_1, G_2) + \text{ite}(y, H_1, H_2) = \text{ite}(x, (G_1 + h), (G_2 + h)) \tag{7}$$

where $h = \text{ite}(y, H_1, H_2)$.

A BDD structure can be constructed using recursive ITE connectives for any fault tree, once a total ordering of variables such as basic events is selected (see the examples in Sections 2 and 3). The evaluation of the top event probability of a BDD structure does not require the preliminary search for MCSs. However, since MCSs provide valuable qualitative and quantitative information to the reliability analyst, they should be extracted from the BDD structure. Rauzy [6] proposed an efficient subsuming method to transform or minimize the original BDD structure into a new BDD structure representing MCSs (see Appendix A).

The size of a BDD structure exponentially increases with respect to the number of variables. This is due to the intrinsic nature of the Boolean function being represented. Another problem is that the size of the final BDD structure is drastically dependent on the choice of variable ordering for the BDD construction. Furthermore, the truncation of ITE connectives by the probability or size limit and the subsuming to delete subsets could not be directly applied to the intermediate BDD structure under construction. The truncation and subsuming should be performed on the final BDD structure. It inherently makes the BDD algorithm very slow and results in a huge memory usage.

The objective of this paper is to develop an efficient BDD algorithm for large coherent systems (coherent BDD algorithm) by which the truncation and subsuming could be performed in the progress of the construction of the BDD structure. The coherent BDD algorithm developed in this study is explained in Section 2. The conventional BDD and coherent BDD algorithms are compared in Section 3. Benchmark tests were performed to show the efficiency of the coherent BDD algorithm for large coherent fault trees and the results are described in Section 4. Conclusions of this study are provided in Section 5.

2. BDD algorithm for coherent systems

The general Shannon decomposition defined in terms of the ITE connective in Eq. (1) could be simplified for the coherent systems as follows:

$$f = \text{ite}(x, f_1, f_0) = xf_1 + f_0. \tag{8}$$

It could be proved that the simplified decomposition holds for the coherent system that has a Boolean function $f = bx + c$ as follows

$$\text{ite}(x, f_1, f_0) = x(b + c) + c = bx + cx + c \Rightarrow bx + c = f$$

where $\Rightarrow$ is an equality operator for the subsuming operation. The term cx is deleted since it is a subset of c (subsuming).

A set of new formulae was developed in this study for the operation between two ITE connectives of a coherent fault tree using the simplified Shannon decomposition in Eq. (8). If x and y are two variables with a variable ordering $x < y$, then the following equalities hold for coherent systems

$$\begin{aligned}\text{ite}(x, G_1, G_2)\text{ite}(x, H_1, H_2) &= (xG_1 + G_2)(xH_1 + H_2)\\ &= x(G_1H_1 + G_1H_2 + G_2H_1) + G_2H_2\\ &= \text{ite}(x, (G_1H_1 + G_1H_2 + G_2H_1), G_2H_2)\end{aligned} \tag{9}$$

$$\begin{aligned}\text{ite}(x, G_1, G_2) + \text{ite}(x, H_1, H_2)\\ &= (xG_1 + G_2) + (xH_1 + H_2)\\ &= x(G_1 + H_1) + (G_2 + H_2)\\ &= \text{ite}(x, (G_1 + H_1), (G_2 + H_2))\end{aligned} \tag{10}$$

$$\begin{aligned}\text{ite}(x, G_1, G_2)\text{ite}(y, H_1, H_2) &= (xG_1 + G_2)h\\ &= xG_1h + G_2h = \text{ite}(x, G_1h, G_2h)\end{aligned} \tag{11}$$

$$\begin{aligned}\text{ite}(x, G_1, G_2) + \text{ite}(y, H_1, H_2) &= (xG_1 + G_2) + h\\ &= xG_1 + (G_2 + h) = \text{ite}(x, G_1, (G_2 + h))\end{aligned} \tag{12}$$

where $h = \text{ite}(y, H_1, H_2)$. Here, please note that Eqs. (9) and (12) differ from Eqs. (4) and (7), respectively.

When $x = y$, the subsuming in Appendix A could be simplified to $\text{ite}(x, G_1 \backslash H_1, G_2 \backslash H_2)$ in the conventional BDD algorithm [6]. However, the simplification could not be applied to the coherent BDD algorithm, since the ITE operation in Eq. (9) produces more ITE connectives $G_1H_2 + G_2H_1$ than the ITE operation in Eq. (4).

Let us solve a simple example fault tree in Fig. 1 by the coherent BDD algorithm in Eqs. (9)–(12) with an alphabetical variable ordering $a < b < c < d$ as

$$\begin{aligned}\text{TOP} &= (a + b)(a + c) + d\\ &= \text{ite}(a, 1, \text{ite}(b, \text{ite}(c, 1, 0), \text{ite}(d, 1, 0))).\end{aligned} \tag{13}$$

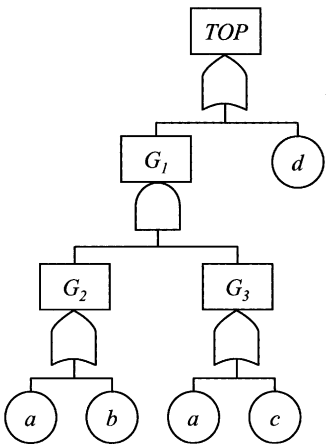


Fig. 1. Example 1.

As shown in Fig. 2(b), the BDD structure in Eq. (13) has three paths (cut sets) from the root to leaf 1 $\{a, bc, d\}$ that are MCSs of the fault tree in Fig. 1. The conventional BDD algorithm in Eqs. (4)–(7) in Section 1 generates the same MCSs as follows

$$\begin{aligned}\text{TOP} &= (a+b)(a+c)+d \\ &= \text{ite}(a,1,\text{ite}(b,\text{ite}(c,1,\text{ite}(d,1,0)),\text{ite}(d,1,0))) \\ &\Rightarrow \text{ite}(a,1,\text{ite}(b,\text{ite}(c,1,0),\text{ite}(d,1,0))) \end{aligned} \quad (14)$$

where $\Rightarrow$ is the equality operator for the subsuming. As shown in Fig. 2(a), there are four paths from the root to leaf 1 $\{a, bc, bd, d\}$. After deleting the subset $\{bd\}$ of $\{a\}$, the BDD structure is reduced to the one in Fig. 2(b).

As shown in Eqs. (13) and (14), the conventional BDD and coherent BDD algorithms generate the same MCSs when the subsuming is performed on the final BDD structure.

3. Comparison of two BDD algorithms

In this section, it is illustrated that the coherent BDD algorithm in Eqs. (9)–(12) makes it possible to delete subset in the intermediate BDD structure under construction. However, the conventional BDD algorithm generates inappropriate MCSs if the subsuming is applied to the intermediate BDD structure under construction.

3.1. Conventional BDD algorithm

Let us solve the fault tree in Fig. 3 by using the conventional BDD algorithm in Eqs. (4)–(7) with an alphabetical variable ordering $a < b < c < d < e$. The BDD structure is constructed in a bottom-up way as follows

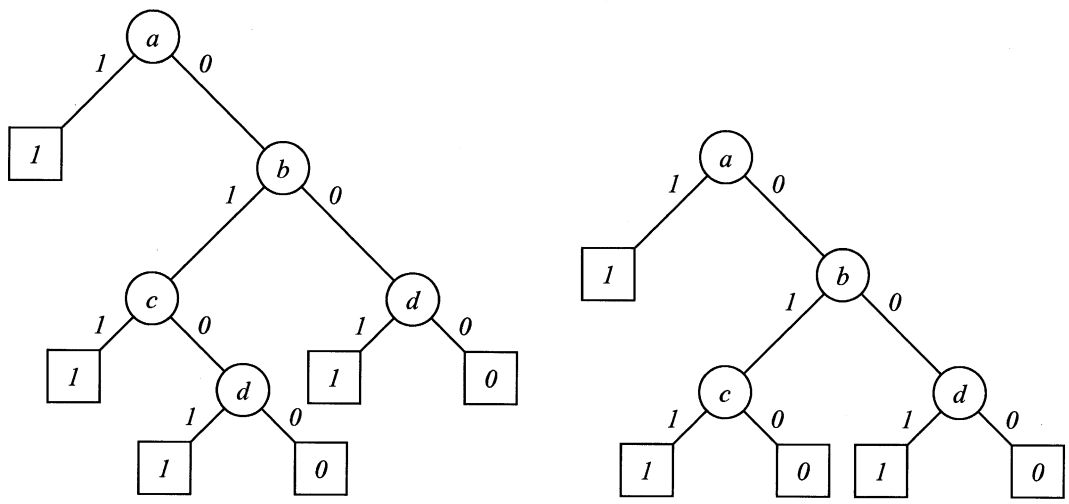
$$G_4 = d + e = \text{ite}(d,1,\text{ite}(e,1,0))$$

$$G_3 = aG_4 = \text{ite}(a,\text{ite}(d,1,\text{ite}(e,1,0)),0)$$

$$G_2 = a + b = \text{ite}(a,1,\text{ite}(b,1,0))$$

$$\begin{aligned} G_1 &= b + c + d + G_3 \\ &= \text{ite}(a,\text{ite}(b,1,\text{ite}(c,1,\text{ite}(d,1,\text{ite}(e,1,0)))), \\ &\quad \text{ite}(b,1,\text{ite}(c,1,\text{ite}(d,1,0)))) \end{aligned}$$

$$\begin{aligned} \text{TOP} &= G_1 G_2 \\ &= \text{ite}(a,\text{ite}(b,1,\text{ite}(c,1,\text{ite}(d,1,\text{ite}(e,1,0)))), \\ &\quad \text{ite}(b,1,0)) \\ &\Rightarrow \text{ite}(a,\text{ite}(c,1,\text{ite}(d,1,\text{ite}(e,1,0))),\text{ite}(b,1,0)) \end{aligned} \quad (15)$$



(a) Before subsuming (b) After subsuming

Fig. 2. BDD structure of example 1.

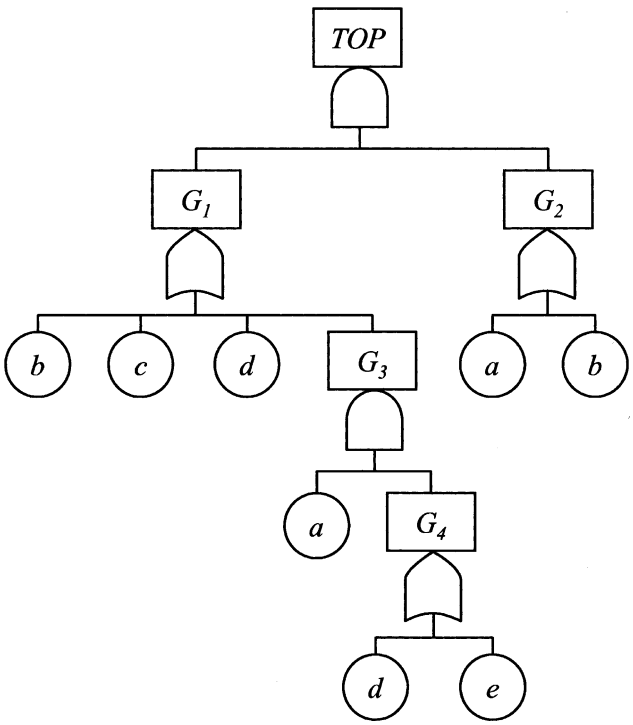


Fig. 3. Example 2.

where cut sets $\{ab, ac, ad, ae, b\}$ are reduced to the final MCSs $\{ac, ad, ae, b\}$ since ab is a subset of b. The final BDD structure is depicted in Fig. 4.

Let us show that the subsuming could not be applied to the intermediate BDD structure under construction when using the conventional BDD algorithm. If the subsuming is performed on the BDD structure of gate G_1, the BDD expression becomes

$$G_1 \Rightarrow \text{ite}(a, \text{ite}(e, 1, 0), \text{ite}(b, 1, \text{ite}(c, 1, \text{ite}(d, 1, 0))))$$

where cut sets $\{ab, ac, ad, ae, b, c, d\}$ are reduced to $\{ae, b, c, d\}$. Then, the BDD structure for the top event is

$$\text{TOP} = \text{ite}(a, \text{ite}(e, 1, 0), \text{ite}(b, 1, 0)). \quad (16)$$

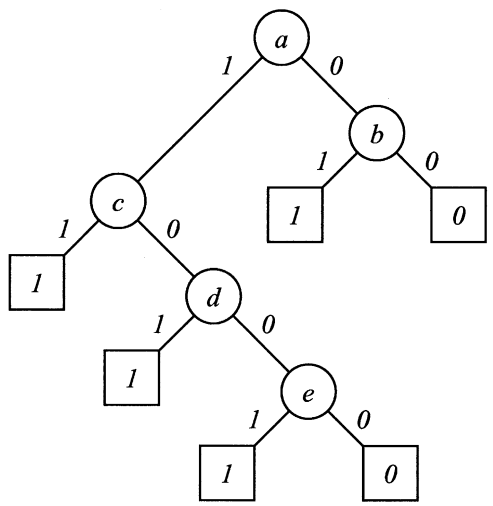


Fig. 4. BDD structure of example 2.

It generates a solution $\{ae, b\}$ where the two MCSs $\{ac, ad\}$ are lost compared with the exact MCSs $\{ac, ad, ae, b\}$ in Eq. (15).

3.2. Coherent BDD algorithm

Let us develop the BDD structure for the same fault tree in Section 3.1 by using the coherent BDD algorithm in Eqs. (9)–(12) with an alphabetical variable ordering $a < b < c < d < e$. When the coherent BDD algorithm is applied to the sub-trees of gates G_2, G_3, and G_4, the resultant BDD structures for each gate are identical to those in Section 3.1. The BDD structure for gate G_1 is further reduced as follows:

$$G_1 = \text{ite}(a, \text{ite}(d, 1, \text{ite}(e, 1, 0)), \text{ite}(b, 1, \text{ite}(c, 1, \text{ite}(d, 1, 0))))$$

$$\Rightarrow \text{ite}(a, \text{ite}(e, 1, 0), \text{ite}(b, 1, \text{ite}(c, 1, \text{ite}(d, 1, 0)))).$$

Here, cut sets $\{ad, ae, b, c, d\}$ are reduced to $\{ae, b, c, d\}$. The BDD structure for the top event becomes

$$\text{TOP} = \text{ite}(a, \text{ite}(b, 1, \text{ite}(c, 1, \text{ite}(d, 1, \text{ite}(e, 1, 0)))), \text{ite}(b, 1, 0))$$

$$\Rightarrow \text{ite}(a, \text{ite}(c, 1, \text{ite}(d, 1, \text{ite}(e, 1, 0))), \text{ite}(b, 1, 0)). \quad (17)$$

In Eq. (17), cut sets $\{ab, ac, ad, ae, b\}$ are reduced to the same correct MCSs $\{ac, ad, ae, b\}$ as those in Eq. (15). As shown in this example, the subsuming can be directly applied to the intermediate BDD structure under construction when using the coherent BDD algorithm. This results in the fast computation of MCSs without consuming a huge memory.

As illustrated in this section, the subsuming could be performed on the intermediate BDD structure under construction. Furthermore, it is easily shown in a similar way that the intermediate BDD structure could be truncated with a probability or MCS size limit.

4. Application to large fault trees

Since the least size of the BDD structure is maintained during the calculation by the subsuming and truncation, the coherent BDD algorithm does not employ any hash function [6] that is for reusing the repeated ITE connectives to save memory.

The coherent BDD algorithm is tested with a fault tree restructuring method [14,15] and a module identification method [16,17]. An adapted pre-processing algorithm for the restructuring and the identification of modules was programmed, which is similar to those in Refs. [14–17]. A typical quantification procedure is as follows

1. Restructure a fault tree and identify modules,
2. Solve each module and assign the maximum MCS probability to the module,

3. Solve the fault tree where modules are treated as basic event,
4. Substitute modules in the final MCSs with their MCSs, and
5. Calculate the top event probability using MCSs.

Here, the term 'solve' in Steps 2 and 3 denotes 'build a BDD structure in a bottom-up way, subsume subsets, and extract MCSs'.

Before building a BDD structure, a fault tree is restructured to get the largest independent sub-trees that could be treated as basic events in the process of quantification of a fault tree. Due to the pre-processing of a fault tree, the quantification of fault trees could be accelerated.

A depth-first traversal ordering of basic events is employed for the benchmark tests. A typical recursive depth-first BDD algorithm is illustrated in Table 1. Starting at the top event and navigating the fault tree in a depth-first manner where basic events of a given gate are visited before traversing any sub-gates, the gates and basic events are visited in the order shown in Table 1. The variable orderings for the two examples in Figs. 1 and 3 are $d < a < b < c$ and $b < c < d < a < e$, respectively.

The coherent BDD algorithm is applied to the large coherent fault trees in Table 2 that could not be solved by the conventional BDD algorithm in a reasonable time and memory usage. As shown in Table 2, the coherent BDD algorithm showed a desirable performance. The coherent BDD algorithm and KIRAP [18] generate the same MCSs. However, the coherent BDD algorithm is much faster than KIRAP. The truncation and subsuming in the progress of the construction of the BDD structure resulted in the fast computation of MCSs and the least memory usage such as the fast fault tree quantifier FORTE [19] that is based on Boolean algebra.

Table 1
Depth-first traversal ordering

Example 1[a]		Example 2[b]	
Nodes[c]	Order	Nodes	Order
TOP	1	TOP	1
d	2	G_1	2
G_1	3	b	3
G_2	4	c	4
a	5	d	5
b	6	G_3	6
G_3	7	a	7
a	NA[d]	G_4	8
c	4	d	NA
		e	9
		G_2	10
		a	NA
		b	NA

[a] Example 1 in Fig. 1.
[b] Example 2 in Fig. 3.
[c] Visited gates or basic events.
[d] Not applicable since the order is already assigned.

Table 2
Benchmark tests of large fault trees

Fault trees	Number of gates	Number of basic events	Truncation limit	Number of MCSs	Run time (s)[a]	
					Coherent BDD[b]	KIRAP[c]
FT1	813	667	1.0×10^{-13}	26,632	1	2
			1.0×10^{-14}	55,708	2	5
			1.0×10^{-15}	119,277	4	14
FT2	1037	705	1.0×10^{-13}	32,217	1	9
			1.0×10^{-14}	64,724	1	23
			1.0×10^{-15}	134,950	2	71
FT3	1587	1761	1.0×10^{-10}	5292	2	10
			1.0×10^{-11}	18,671	5	33
			1.0×10^{-12}	58,543	12	114
FT4	4762	2825	1.0×10^{-9}	24,332	1	35
			1.0×10^{-10}	69,323	2	83
			1.0×10^{-11}	179,618	4	237
FT5	5148	2558	1.0×10^{-9}	5,611	3	18
			1.0×10^{-10}	34,107	7	43
			1.0×10^{-11}	188,854	21	151

[a] Pentium IV 2.56 GHz CPU, 512 MB RAM, Windows XP.
[b] Coherent BDD algorithm presented in this paper.
[c] Fault tree quantifier in Ref. [18].

5. Conclusions

Although it is easy to implement the BDD algorithm to solve fault trees, there have been some limitations. The size of a BDD structure exponentially increases according to the number of variables since the truncation of ITE connectives by the probability or size limit and the subsuming could not be directly applied to the intermediate BDD structure under construction. Furthermore, the size of a BDD structure depends critically on variable ordering and the determination of an appropriate variable ordering has a highly heuristic nature.

This paper presents an efficient BDD algorithm for large coherent systems that overcomes the limitations. A set of new formulae for the operation between two ITE connectives of a coherent fault tree makes it possible to subsume subset and truncate ITE connectives with a probability or size limit in the intermediate BDD structure under construction. The truncation and subsuming in the progress of the construction of the BDD structure is the key to the fast quantification of large coherent fault trees using less memory. Since the truncation and subsuming is performed on the intermediate BDD structure, the least size of the BDD structure is maintained during the calculation. Thus, the coherent BDD algorithm is much less sensitive to variable ordering than the conventional BDD algorithm.

Appendix A. Subsuming of BDD algorithm

In order to get minimal solutions of a BDD structure, the subsuming is recursively performed from the root ITE to the child ITE connectives by comparing left and right ITE connectives. Let us consider recursive ITE connectives $F = \text{ite}(t, G, H)$, $G = \text{ite}(x, G_1, G_2)$, and $H = \text{ite}(y, H_1, H_2)$. In order to get MCSs of F, each cut set in G is tested and deleted if it is a subset of a cut set in H (subsuming operation $G\backslash H$). Rauzy [6] proposed an efficient subsuming operation:

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

The term $G_1\backslash(H_1$ or $H_2)$ in the last case denotes that each cut set in G_1 is tested and deleted if it is a subset of a cut set in H_1 or H_2.

References

[1] Lee CY. Representation of switching circuits by binary-decision programs. Bell Syst Tech J 1959;38:985–99.

[2] Akers B. Binary decision diagrams. IEEE Trans Comput 1978; C-27(6):509–16.

[3] Bryant R. Graph based algorithms for Boolean function manipulation. IEEE Trans Comput 1986;C-35(8):677–91.

[4] Bryant R. Symbolic Boolean manipulation with ordered binary decision diagrams. ACM Comput Surv 1992;24:293–318.

[5] Coudert O, Madre JC. Implicit and incremental computation of primes and essential primes of Boolean functions. Proceedings of the 29th ACM/IEEE Design Automation Conference, DAC'92, June. 1992.

[6] Rauzy A. New algorithms for fault trees analysis. Reliab Engng Syst Saf 1993;40:203–11.

[7] Coudert O, Madre JC. Fault tree analysis: 10^{20} prime implicants and beyond. Proceedings of the Annual Reliability and Maintainability Symposium, Atlanta, NC, USA, January. 1993.

[8] Rauzy A, Dutuit Y. Exact and truncated computations of prime implicants of coherent and non-coherent fault trees within aralia. Reliab Engng Syst Saf 1997;58:127–44.

[9] Way YS, Hsia DY. A simple component-connection method for building binary decision diagrams encoding a fault tree. Reliab Engng Syst Saf 2000;70:59–70.

[10] Sinnamon RM, Andrews JD. New approaches to evaluating fault trees. Reliab Engng Syst Saf 1997;58:89–96.

[11] Bartlett LM, Andrews JD. An ordering heuristic to develop the binary decision diagram based on structural importance. Reliab Engng Syst Saf 2001;72:31–8.

[12] Rauzy A. Mathematical foundations of minimal cutsets. IEEE Trans Reliab 2001;50(4):389–96.

[13] Barlow RE, Proschan F. Statistical theory of reliability and life testing. Holt: Rinehart and Winston, Inc.; 1975.

[14] Niemelä I. On simplification of large fault trees. Reliab Engng Syst Saf 1994;44:135–8.

[15] Reay KA, Andrews JD. A fault tree analysis strategy using binary decision diagrams. Reliab Engng Syst Saf 2002;78:45–56.

[16] Han SH, Kim TW, Yoo KJ, Development of an integrated fault tree analysis computer code MODULE by modularization technique. Reliab Engng Syst Saf 1988;21:145–54.

[17] Dutuit Y, Rauzy A. A linear-time algorithm to find modules of fault trees. IEEE Trans Reliab 1996;45:422–5.

[18] Han SH. PC-Workstation Based Level 1 PRA Code Package-KIRAP. Reliab Engng Syst Saf 1990;30:313–22.

[19] Jung WS, Kim DK. FORTE: a fast new algorithm for risk monitors and PSA. Proceedings of the Fourth International Conference on Probabilistic Safety Assessment and Management, September, New York, USA. 1998. p. 1221.