Transactions of the Korean Nuclear Society Autumn Meeting
PyeongChang, Korea, October 25-26, 2007

How to incorporate a truncation limit into BDD

Woo Sik Jung, Sang Hoon Han, and Joon-Eon Yang
Korea Atomic Energy Research Institute, P.O. Box 105, Yuseong, Daejeon, 305-600, Korea

1. Introduction

A Binary Decision Diagram (BDD) is a graph based data structure and it has become a very popular method to calculate the exact top event probability (TEP) of a small or intermediate size reliability problem. In order to solve a large problem, this study presents an efficient method to maintain a BDD size within computational resources. The fast calculation was accomplished by making it possible to truncate BDDs when a Boolean operation between two BDDs is performed.

1.1 BDD Algorithm

A BDD [1-3] is a graph based data structure which allows efficient representation and manipulation of Boolean formulae, and it was proved that the BDD is effective in diverse fields of computer science and reliability [4]. Bryant [3] popularized the use of the BDD by developing a set of algorithms for the efficient construction and manipulation of BDDs. The BDD was applied to the reliability analysis [5,6] and has been investigated to solve large fault trees and importance measures [7-10].

1.2 ZBDD Algorithm

ZBDD (Zero-suppressed BDD) that encodes minimal cut sets (MCSs) is an important variation of the BDD [11]. By developing special formulae for the Boolean operations on two ZBDDs, it was shown that the operation could be performed with a truncation limit [12]. Due to the nature of the ZBDD, MCSs could be easily calculated with a given truncation limit. Both in computation time and memory usage, ZBDD algorithm is more efficient than MCS algorithms that are based on the classical Boolean algebra [12-14].

1.3 Variable Ordering of BDD Algorithm

The BDD algorithm generates a BDD structure from a fault tree and calculates the exact TEP. It is well known that the BDD algorithm is highly memory consuming.

In order to solve a large reliability problem within limited computational resources, lots of efforts have been done to minimize a BDD size. The size of a BDD structure (measured in the number of nodes) is drastically dependent on the choice of the variable ordering for the BDD construction. Finding the optimal variable ordering is an NP-hard problem. Bryant [3] has shown the importance of a good variable ordering may lead to a small size of a BDD structure. All known methods for finding a better variable ordering are based on static and dynamic variable ordering heuristics. Dynamic variable reordering heuristics that are based on a variable sifting are considered as a significant improvement of the BDD technology [15]. But unfortunately sifting is very time-consuming for large functions and the dynamic variable reordering method is still inefficient to solve large problems. Please note that most heuristics are based on decision of trade-off between fast run-time and small BDDs.

2. Methods and Results

The method to truncate a BDD and its test results are explained in this section.

2.1 BDD Algorithm

The conventional Shannon decomposition is succinctly defined in terms of the ternary If-Then-Else (ITE) connectives as

$$\begin{aligned} F &= ite(x, F_1, F_0) = xF_1 + \bar{x}F_0 \\ G &= ite(y, G_1, G_0) = yG_1 + \bar{y}G_0 \end{aligned} \qquad (1)$$

where x and y are two variables with a variable ordering $x < y$. BDD starts from a single initial node, two children nodes are connected to the parent node with edges labeled 0 and 1, and the final nodes are always one of two leaf nodes labeled 0 or 1. BDD operation is recursively performed on variable x that has a higher priority as

$$H = F <> G = \begin{cases} ite(x, F_1 <> G_1, F_0 <> G_0) & if\ x = y \\ ite(x, F_1 <> G,\ F_0 <> G) & if\ x < y \end{cases} \qquad (2)$$

where <> is *AND* or *OR* Boolean operator.

In order to save memory usage by maintaining a unique $ite(x, F_1, F_0)$, $\{hash\_key(x, F_1, F_0), F\}$ is stored and retrieved to and from a 'ITE hash table'. Here, $hash\_key(x, F_1, F_0)$ is a hash function that maps a triple into a node index.

For suppressing the repetition of the same operation $H=F<>G$, the BDD operation results $\{hash\_key(<>, F, G), H\}$ are stored into an 'operation hash table'. Please note that it was well known that the BDD truncation can not be used together with a hash table since it gives wrong answers.

2.2 BDD Algorithm with Truncation

In this study, the method to incorporate the truncation limit into an 'operation hash table' was developed. Whenever a BDD operation is recursively performed on variable x, $p(x)$ or 1.0- $p(x)$ is multiplied to the upper probability p.

(1) If the probability p is less than the truncation limit, the operation is stopped and returns 0.
(2) Before the operation, if {*hash_key*(<>, *F*, *G*), *H*, *q*} is in the hash table and $q > p$, the operation returns *H*.
(3) After the operation, $H=F<>G$ is stored in the hash table. If {*hash_key*(<>, *F*, *G*), *T*, *q*} is in the hash table and $q < p$, *T* and *q* are replaced with *H* and *p*. Else, {*hash_key*(<>, *F*, *G*), *H*, *p*} is stored in the hash table.

Table 1. Benchmark test A

Fault tree = CEA9601
http://iml.univ-mrs.fr/~arauzy/aralia/benchmark.html
201 gates, 186 events, 26 negates, 4 complemented events
All event probabilities = 0.001
Without fault tree restructuring and modules

Truncation	TEP	Run time (seconds)	BDD node number
1.00E-11	1.059240E-06	0.88	12,349
1.00E-12	1.092776E-06	1.47	19,090
1.00E-13	1.176633E-06	1.77	54,280
1.00E-14	1.180200E-06	3.11	154,728
1.00E-15	1.181040E-06	4.36	200,157
1.00E-16	1.182503E-06	4.67	320,805
1.00E-17	1.182611E-06	7.77	703,816
1.00E-18	1.182618E-06	9.94	811,113
Exact TEP	1.182622E-06	6.22	1,250,725

Table 2. Benchmark test B

Fault tree = HPSI3.FTP
571 gates, 421 events, 0 negates, 0 complemented events
With fault tree restructuring and modules

Truncation	TEP	Run time (seconds)	BDD node number
1.00E-11	1.076139E-03	2.15	130,013
1.00E-12	1.076293E-03	3.78	274,187
1.00E-13	1.076325E-03	6.76	573,908
1.00E-14	1.076332E-03	12.60	1,131,067
1.00E-15	1.076334E-03	21.87	2,051,615
Exact TEP	1.076334E-03	21.31	2,497,172

2.3 Test Results

Two Benchmark problems were solved and their results are listed in Tables 1 and 2. The TEP rapidly converges to an exact value. Furthermore, this method is very fast and uses much less memory (nodes).

3. Conclusion

In order to solve a large reliability problem within limited computational resources, lots of efforts have been done to minimize a BDD size. The method is to find the optimal variable ordering by some heuristics. This paper explains another efficient method that provides an accurate TEP in a reasonably short time. It was accomplished by making it possible to truncate BDDs when a Boolean operation between two BDDs is performed.

REFERENCES

[1] C.Y. Lee, "Representation of switching circuits by binary-decision programs," Bell System Technical Journal, 38, pp. 985-999, 1959.
[2] B. Akers, "Binary Decision Diagrams," IEEE Transactions on Computers, C-27(6), pp. 509-516, 1978.
[3] R. Bryant, "Graph Based Algorithms for Boolean Function Manipulation," IEEE Transactions on Computers, C-35(8), pp. 677-691, August, 1986.
[4] R. Bryant, "Symbolic Boolean Manipulation with Ordered Binary Decision Diagrams," ACM Computing Surveys, 24, pp. 293-318, September 1992.
[5] O. Coudert and J.C. Madre, "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] A. Rauzy, "New Algorithms for Fault Trees Analysis," Reliability Engineering and System Safety, 40, pp. 203-211, 1993.
[7] O. Coudert and J.C. Madre, "Fault Tree Analysis: 1020 Prime Implicants and Beyond," Proceedings of the Annual Reliability and Maintainability Symposium, Atlanta, NC, USA, January 1993.
[8] A. Rauzy and Y. Dutuit, "Exact and Truncated Computations of Prime Implicants of Coherent and Non-coherent Fault Trees Within Aralia," Reliability Engineering and System Safety, 58, pp. 127-144, 1997.
[9] Y. Dutuit and A. Rauzy, "Efficient Algorithms to Assess Component And Gate Importance in Fault Tree Analysis," Reliability Engineering & System Safety, Volume 72, pp. 213-222, May 2001.
[10] S. Epstein, A. Rauzy, "Can we trust PRA," Reliability Engineering & System Safety, Volume 88, pp. 195-205, June 2005.
[11] S. Minato., "Zero-suppressed BDDs for set manipulation in combinatorial problems," Proc. of the 30th Int'l Conf. on Design Automation, pp. 272-277, 1993.
[12] W.S. Jung, S.H. Han, J.J. Ha, "A Fast BDD Algorithm for Large Coherent Fault Trees Analysis," Reliability Engineering and System Safety, Vol. 83, pp. 369–374, 2004.
[13] W.S. Jung, S.H. Han, J.J. Ha, "Development of an Efficient BDD Algorithm to Solve Large Fault Trees," Proceedings of the 7th International Conference on Probabilistic Safety Assessment and Management, June, Berlin, Germany, 2004.
[14] W.S. Jung, S.H. Han, J.J. Ha, "An Overview of the Fault Tree Solver FTREX," 13th International Conference on Nuclear Engineering, Beijing, China, May 16-20, 2005.
[15] R. Rudell, "Dynamic variable ordering for ordered binary decision diagrams," International Conference on Computer Aided Design, pp. 42-47, November 1993.