Available online at www.ijpe-online.com

International Journal of Performability Engineering

vol. 15, no. 8, August 2019, pp. 2062-2070
DOI: 10.23940/ijpe.19.08.p5.20622070

Reliability Analysis of Cold Standby Systems based on Supplement Events using Multiple-Valued Decision Diagrams

Ting Zeng*

Periodicals Press of Jianghan University, Jianghan University, Wuhan, 430056, China

Abstract

Multiple-valued decision diagrams (MDDs) have been widely used in reliability analysis for non-repairable fault-tolerant systems. Typically, multiple-valued decision diagrams are used to analyze the static fault tree (SFT) model since basic events in the SFT are non-sequence-dependent. However, a cold standby system belongs to the dynamic fault tree model; the traditional MDD cannot be directly applied in this system. Hence, a novel analytical method based on conditional events using multiple-valued decision diagrams for combinatorial reliability analysis of non-repairable cold standby systems is proposed. In the proposed method, a set of fault events with specific supplement events replaces cut sequences for qualitative analysis. The sets are converted to algebraic structure-based models for quantitative analysis. Additionally, the system MDD model and reliability evaluation expression can be reused for reliability analysis with different component failure parameters.

Keywords: reliability; fault tree analysis (FTA); multiple-valued decision diagram (MDD)

(Submitted on May 27, 2019; Revised on July 5, 2019; Accepted on August 12, 2019)

1. Introduction

A cold standby system is an application of fault-tolerant technology that involves having one or more systems as a backup for another identical primary system. The cold standby system is called upon only on failure of the primary system. Cold standby systems are turned on once to install and configure the system and data and then turned off until needed. Thereafter, they are employed only when updating noncritical data, which is done infrequently, or when the primary system fails. Fault tree analysis is a traditional reliability analysis method that is suitable for the system. The dynamic fault tree (DFT) [1] was first proposed in 1992. It is an extension of the static fault tree (SFT) [2], which involves functional-dependency failure and sequence-dependent failure. Thus, aside from static gates (OR gate, AND gate, etc.), several dynamic gates such as functional-dependency (FDEP) gates, spare gates, priority-and (PAND) gates, sequence-enforcing (SEQ) gates, cold spare (CSP) gates, warm spare (WSP) gates, and hot spare (HSP) gates have been introduced via DFTs. Among these dynamic gates, the CSP gate model is mainly used for reliability analysis of cold standby systems, as shown in Figure 1. The CSP gate is the most economical, since its spare component is always in an un-power state before the cold spare component is activated by the current primary component failure. The cold spare component requires a long time to replace the faulty component in the CSP. Hence, the CSP is typically applied in places where power shortages occur, such as satellites and conventional submarines. A CSP gate, however, models sequential dependence between the online component and standby units. In particular, a cold spare component can start to work and then fail only after the online component has failed. Thus, a cold spare component has sequence-dependent failure behaviors and two different failure probabilities in different states. Current approaches to reliability analysis of DFTs with SP gates are Markov-based methods [3], simulation-based methods [4-5], Bayesian network-based methods [6-7], and algebraic-structure-based methods [8]. The above methods have some problems. The state-space-based methods easily suffer from the state-space explosion problem and typically are limited to the exponential time-to-failure distribution for the system components. The simulation-based methods, e.g., Monte Carlo simulations, do not offer exact results and often involve long computational times. Furthermore, in the simulation-based methods, a completely new simulation must be performed whenever the input failure parameter values change.

* Corresponding author.
E-mail address: flora198234@jhun.edu.cn

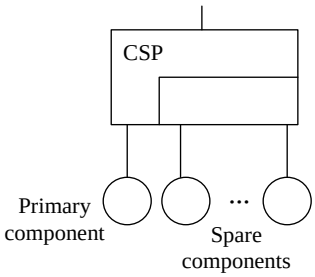


Figure 1. Cold spare (CSP) gate

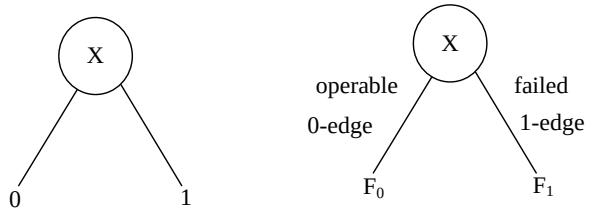
In this paper, an analytical and combinatorial method is proposed for reliability analysis of cold standby systems while addressing the problems of the existing approaches. The proposed method is based on multiple-valued decision diagrams (MDD), an extended version of the binary decision diagram (BDD) [9-10].

2. Binary Decision Diagram

BDD is a graphical representation of a Boolean logic function that uses a directed acyclic tree graph to represent Boolean logic functions *f*: {0, 1}*n* → {0, 1}. It mainly consists of a top node, the internal nodes, and the terminal nodes. The two BDD terminal nodes are logic value '0' and value '1'. A BDD is a directed acyclic graph (DAG) based on Shannon decomposition, as shown in Equation (1).

$$f = x \cdot f_{x=1} + \neg x \cdot f_{x=0} = ite(x, F_1, F_2) \tag{1}$$

The general BDD node and prime BDD node are shown in Figure 2.



(a) The prime BDD node (b) The general BDD node

Figure 2. Two BDD nodes: a) the prime BDD node; b) the general BDD node

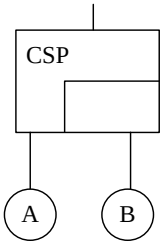
The bottom-up method generally traverses the fault tree to generate the corresponding BDD, which requires the constant recursive to call off the operation, as shown below.

$$\begin{aligned} G \lozenge H &= ite(x, G_1, G_0) \lozenge ite(y, H_1, H_0) \\ &= \begin{cases} ite(x, G_1 \lozenge H_1, G_0 \lozenge H_0) & index(x) = index(y); \\ ite(x, G_1 \lozenge H, G_0 \lozenge H) & index(x) < index(y); \\ ite(y, G \lozenge H_1, G \lozenge H_0) & index(x) > index(y); \end{cases} \end{aligned} \tag{2}$$

Where G and H represent two Boolean expressions corresponding to the traversed sub-trees. G_i is a sub-expression of G, and H_i is a sub-expression of H. "*index*" indicates the order between variables defined in the input variable list. $\lozenge$ represents a logical operation, such as AND and OR.

3. Minimal Cut Set with Supplement Events

The spare gate belongs to dynamic fault trees, and the CPS gate is a kind of spare gate. The CSP gate fails only when basic events in the CSP gate fail in a specific sequence. Due to algebraic-structure-based methods [8], a symbol $\lhd$ denotes a sequential relationship "before"; for example, $A \lhd B$ denotes event *A* fails before event *B*. The simple CSP gate shown in Figure 3 fails under the condition $(A \lhd B) \cdot B$.



$$(A \lhd B) \cdot B$$

Figure 3. Sequence-dependent failure behaviors for the CSP gate

However, the component in spare gates has an activated state besides type, operable, and failure. The activated state denotes the spare component is activated by the current primary component failure. In other words, the current primary component fails before the activated spare component failure. The un-activated spare component in CSP gates cannot fail before the current primary component failure. As a result, the combination of component static states can be considered to replace the sequence-dependent failure behaviors between components.

Our supplement events are regarded as static states and may also involve s-dependent events, but they are mutually exclusive. Sa_x is a supplement event that denotes the activated state of a spare component *x*. *Type* denotes a WSP gate with value 1 and a CPS gate with value 0. The MCS with supplement events of simple cold spare gates failure can be expressed as shown in Figure 4. Considering the two states of the cold spare component, stopping state and working state, which are denoted by α and λ respectively, the static state transform in Figure 4 is changed as follows:

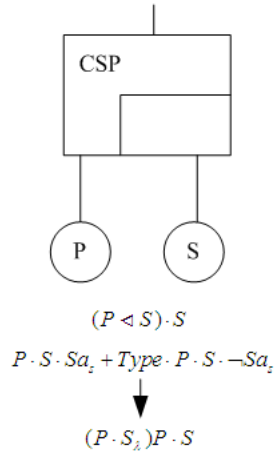


Figure 4. MCS with supplement events of simple spare gates failure

$$(P \cdot S_\lambda) P \cdot S + Type \cdot (P \cdot S_\alpha) P \cdot S \tag{3}$$

The parenthesis is a set of supplement events; for example, $(P \cdot S_\lambda)$ denotes the spare component *S* is activated by the current primary component *P*, and $(P \cdot S_\alpha)$ denotes the spare component *S* is un-activated by the current primary component *P*. Note that the supplement events does not indicate component failure. Unlike basic fault events, supplement events do not participate in reliability computation directly. When the cold spare gate model is confirmed, the value of type is 0. The MCS with supplement events of general cold spare gate (one primary component with n spare components) failure is as follows:

$$(P \cdot S_{1_\lambda} \cdots\cdots S_{n_\lambda}) P \cdot S_1 \cdots S_n \tag{4}$$

The event *P* denotes the primary component in spare gates. S_1 to S_n denote the spare components in spare gates from left to right in the sequence. Compared with the sequence-dependent failure behavior, the static transform expression of spare gates can describe where the fault events are and which failure states fault events are in. Thus, it can be returned to the DFT model of spare gates. Furthermore, a static state combination including fault events and condition events instead of fault event permutation describes DFTs with spare gate dynamic failure behaviors.

In this paper, the supplement events only refer to the relationship between primary components and spare components in spare gates. The spare component only can be activated by the current faulty primary component, which is on the left. The current primary component failure can only activate one spare component, and a spare component can only be activated by one primary component failure. Hence, at least two supplement events (one primary component event and one spare component event) are in the supplement set.

The MCS with supplement events is further explained via a cold standby system example of a processor subsystem in a computer system with three processors (*A*, *B*, and *S*). Processors *A* and *B* share the same cold-standby processor *S*. The first processor to fail is replaced by *C*; the standby processor is then unavailable when the other processor fails. The processor subsystem functions as long as at least one of the three processors is operating correctly, as shown in Figure 5. The DFT describes two shared CSP gates connected to the logic OR gate. The two shared CSP gates have three components: the primary component *A*, the primary component *B*, and a shared spare component *S*. The sequence-dependent failure model of the DFT in Figure 5 is $A \cdot B + (A \triangleleft S) \cdot S + (B \triangleleft S) \cdot S$. It can be converted into a static transform expression with MCS and supplement events as follows:

$$(A \cdot S_\lambda) A \cdot B + (B \cdot S_\lambda) A \cdot B + (A \cdot S_\lambda) A \cdot S + (B \cdot S_\lambda) B \cdot S = A \cdot B + (A \cdot S_\lambda) A \cdot S + (B \cdot S_\lambda) B \cdot S \tag{5}$$

MCSs with supplement events are $A \cdot B$, $(A \cdot S_\lambda) A \cdot S$, and $(B \cdot S_\lambda) B \cdot S$ (B ·S λ)B ·S. $(A \cdot S_\lambda) A \cdot B$ and $(B \cdot S_\lambda) A \cdot B$ have a set of two supplement events, but $A \cdot B$ has no set (the number of supplement events is zero). Hence, with the same fault events, $A \cdot B$ is minimal.

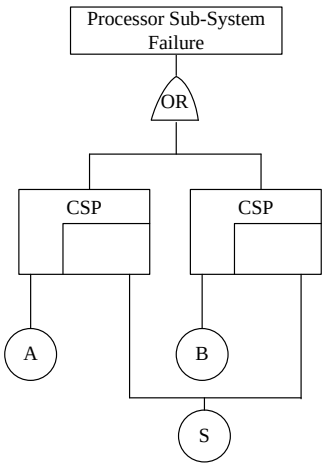
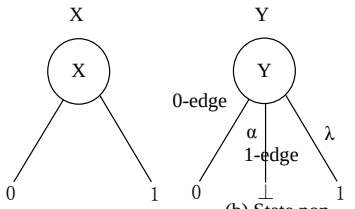


Figure 5. A DFT model of cold standby systems [11]

4. Multiple-Valued Decision Diagram

A DFT with spare gates has two types of logic gates: static gates and dynamic gates. Therefore, it also has two kinds of basic events: non-sequence events and sequence events. Note that, in this paper, the sequence event only refers to the spare component in spare gates. The prime non-sequence event is described as a normal binary node like the prime BDD node, as shown in Figure 6(a), but the prime sequence event is different. The sequence event can be converted into the conditional non-sequence event via the static state transformation. The failure state in the spare component can be divided into two parts based on the static state transformation. Hence, the ternary node is proposed to describe the state non-sequence events via the three mutual exclusive outgoing edges: 0-edge, 1-edge, and 2-edge, respectively representing the operable state, failure in the dormant state, and failure in the activated state. However, there is no dormant state in the CSP gate since the spare component has no power before it is activated. It is a stopping state that is a non-operable state. Thus, the ternary node has three terminal nodes, labeled by logic values '0', '1', and '⊥', respectively representing the system operational state, the system failure state, and the cold spare component stopping state. The existence of a stopping state implies it cannot be interactive with other states including itself. In other words, in the MDD, the result is also '⊥' when any other logic value performs a logical operation with '⊥'. The prime state non-sequence nodes (prime ternary nodes) are shown in Figure 6(b). Similar to the if-then-else (*ite*) format for expressing Boolean expressions in the BDD operation and the case format in the TDD [12-13], the MDD is expressed in a combinatorial *ite−case* format, and Boolean logic operations are used for the MDD. However, '⊥' is a special case where the result is '⊥' when '⊥' is operating with any Boolean variables.



(a) Non-sequence node (b) State non-sequence node in the CSP

Figure 6. Two prime nodes in the MDD

Sub_MDDs can be created and then combined based on the current static gate (or conditional static gate) logic operation. Let the *ite−case* format for the CMDD with binary-valued or ternary expressions *G* and *H*, representing the two Sub_MDDs, be the following:

$$
\begin{aligned}
&\textit{if } G \textit{ or } H \textit{ is binary node, then}\\
&G = ite(X, G_0, G_1)\\
&H = ite(Y, H_0, H_1)\\
&\textit{if } G \textit{ or } H \textit{ is ternary node, then}\\
&G = case(X, G_0, G_1, G_2)\\
&H = case(Y, H_0, H_1, H_2)
\end{aligned}
$$

Let $\Diamond$ represent any logic operation (AND/OR), and then we have $G \Diamond H =$

$$
\begin{aligned}
&\textit{if } G \textit{ and } H \textit{ are binary nodes, then}\\
&\begin{cases}
ite(X, G_0 \Diamond H_0, G_1 \Diamond H_1) & index(X) = index(Y);\\
ite(X, G_0 \Diamond H, G_1 \Diamond H) & index(X) < index(Y);\\
ite(Y, G \Diamond H_0, G \Diamond H_1) & index(X) > index(Y);
\end{cases}\\
&\textit{if } G \textit{ and } H \textit{ are ternary nodes, then}\\
&\begin{cases}
case(X, G_0 \Diamond H_0, G_1 \Diamond H_1, G_2 \Diamond H_2) & index(X) = index(Y);\\
case(X, G_0 \Diamond H, G_1 \Diamond H, G_2 \Diamond H) & index(X) < index(Y);\\
case(Y, G \Diamond H_0, G \Diamond H_1, G \Diamond H_2) & index(X) > index(Y);
\end{cases}\\
&\textit{if } G \textit{ is a bianry node and } H \textit{ is a ternary node, then}\\
&\begin{cases}
\perp & index(X) = index(Y);\\
ite(X, G_0 \Diamond H, G_1 \Diamond H) & index(X) < index(Y);\\
case(Y, G \Diamond H_0, G \Diamond H_1, G \Diamond H_2) & index(X) > index(Y);
\end{cases}
\end{aligned}
$$

Where *index*(*X*) and *index*(*Y*) denote the index of *X* and *Y* in the sequence of two Sub_CMDDs nodes, respectively. All nodes are Boolean variables that come from a Boolean function converted from a DFT with spare gates. Similar to the BDD, the size of an MDD also depends greatly on the order chosen for Boolean variables. To simplify the MDD computation, we adopt a solution in [14] to sort the Boolean variables. Then, a recursive operation can be used until one of them becomes a terminal node.

It is worth noting that it is impossible for both a sequence node and a non-sequence node to have a same variable index, since a component cannot be in the working state and non-working state simultaneously. This rule can handle certain contradictions that building SBDD [14-16] cannot avoid directly, such as an SBDD node $(A \lhd B) \cdot B$ logic AND the other SBDD $(B \lhd A) \cdot A$, in a DFT with spare gates. It is clear that event *A* or event *B* is both a binary node and a ternary node in the MDD. According to the *ite−case* format, the result of operating node A and node B is '$\perp$' (false).

The general MDD node is either a binary node (like BDD) or a binary combining ternary node. The general MDD can be composed of general non-sequence nodes, as shown in Figure 7(a), and general conditional non-sequence nodes, as

shown in Figure 7(b). Strictly speaking, a conditional non-sequence node cannot be used alone because a spare component is impossible to utilize without the primary component failure in the spare gate initially. Hence, a conditional non-sequence node connects to at least one non-sequence node in the MDD. The static state transformation (SST) of a simple cold spare gate is shown in Figure 8. Based on the SST, an MDD of the cold spare gate is generated by the Boolean logic operation rules, as shown in Figure 9. It is specified that the binary node (non-sequence node) variable index is less than the ternary node (conditional non-sequence node) variable index when they are in the same spare gate. The less ternary node variable index is closer to the primary component among the ternary nodes in the same spare gate. Additionally, ternary node index order is required to accord with the order of spare components (from left to right) in the spare gate. The cold spare gate with n spare components in MDDs is shown in Figure 10.

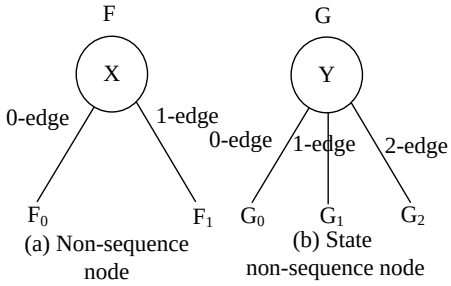


Figure 7. General sub-nodes in the MDD

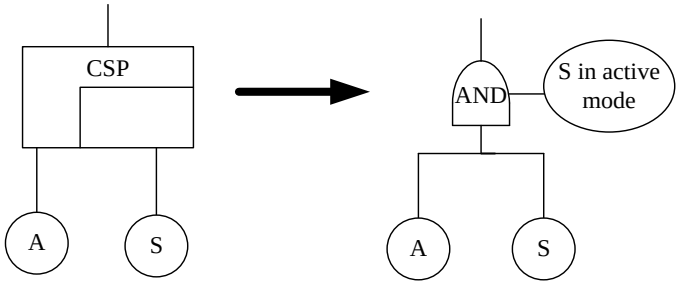


Figure 8. Static state transformations

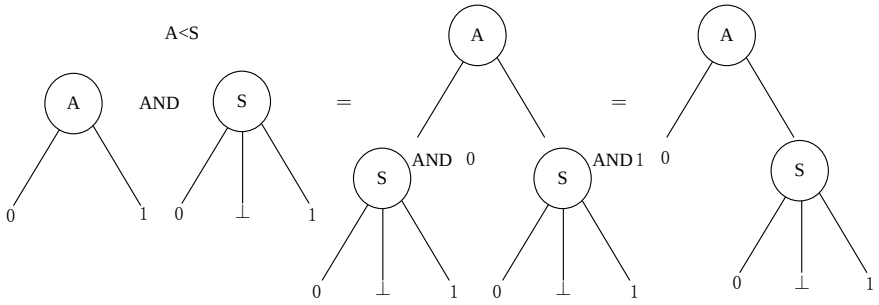


Figure 9. Generating a cold spare gates in the MDD

Spare gate components are usually seen as an entirety when they are in the same spare gate. Thus, all node variable indexes of the spare gate are represented as a whole index when they combine with other Sub_MDD nodes. For example, a logic OR operation between a binary node and a CSP gate MDD is shown in Figure 11.

The node variable superscript denotes which node variables are in the same spare gate, for example, X_1 and Y_1 are in the same spare gate. However, it is unnecessary to mark superscripts in a single spare gate or other easily distinguishable cases. (X^1Y^1) is a whole node index and can be seen as $X^1 < Y^1$. Similarly, $(X_1^1Y_1^1) < (X_2^1Y_2^1)$ equals $(X_1^1 < Y_1^1 < X_2^1 < Y_2^1)$. Nevertheless, the whole node index cannot be separated by any other node index except for in the shared spare component situation. In the shared spare gate condition, although there are multiple spare gates with different primary components, the spare component is shared by them. Hence, in the general case, they are treated as a whole spare gate.

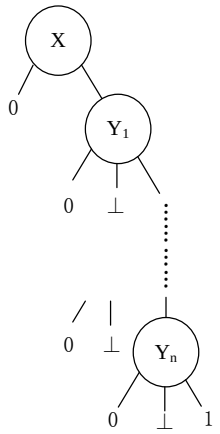


Figure 10. A cold spare gate with n spares

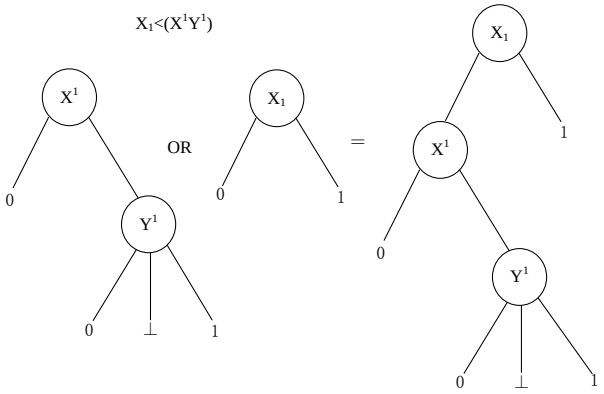


Figure 11. Logic OR operation between a binary node and a CSP gate MDD

5. Study Case

A cold standby system that is a part of HECS (hypothetical example computer system) [15-16] is shown in Figure 12. It includes two CSP gates that share a spare component and are connected by a logical AND gate. After conditional state transformation, the system MDD is built as shown in Figure 13. The order (*AS*) < (*BS*) can be interpreted as (*A*<*B*<*S*). The cut set is $(ABS_\lambda)ABS$ and equals two MCSs, $(AS_\lambda)ABS$ and $(BS_\lambda)ABS$. $(AS_\lambda)ABS$ denotes *S* is activated by *A* and all of them are failed at last. It can deduce two MCSs, $(A \lhd S \lhd B)\cdot B$ and $(A \lhd B \lhd S)\cdot S$. Similarly, $(BS_\lambda)ABS$ can deduce two MCSs, $(B \lhd S \lhd A)\cdot A$ and $(B \lhd A \lhd S)\cdot S$. In most cases, the standby system fails since all the spare components fail, but the system can also fail due to no spare components being available. For example, one of two CSP gates will fail if both the primary component *A* and the primary component *B* fail in the DFT, as shown in Figure 12.

To verify the correctness of our approach, we compare the reliability analysis results of the HECS using the proposed approach with those obtained using the traditional Markov-based method and SBDD method [17]. Because the Markov-based method is limited to the exponential component time-to-failure distribution, we set the following parameters: primary component *A* fails with a constant rate of λ_A = 0.001/day, *B* fails with a constant rate of λ_B = 0.003/day, and *S* fails with a constant rate of λ_S = 0.0025/day after activation [17].

$$\begin{aligned}&\Pr\{(AS_\lambda)ABS + (BS_\lambda)ABS\} = \Pr\{(A \lhd S \lhd B)\cdot B + (A \lhd B \lhd S)\cdot S + (B \lhd S \lhd A)\cdot A + (B \lhd A \lhd S)\cdot S\} \\ &= \Pr\{(A \lhd S \lhd B)\cdot B\} + \Pr\{(A \lhd B \lhd S)\cdot S\} + \Pr\{(B \lhd S \lhd A)\cdot A\} + \Pr\{(B \lhd A \lhd S)\cdot S\}\end{aligned} \tag{6}$$

The results obtained from the Markov analysis match those obtained using our proposed method in three different mission times (*t* = 300, 600, and 1000), as shown in Table 1. Since MCSs with supplement events can be converted to the algebraic structure as shown in Equation (6), our proposed method can be suitable for an arbitrary distribution.

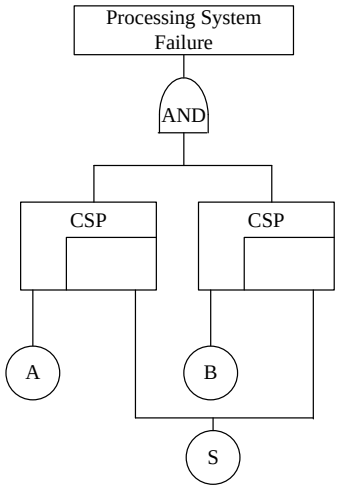


Figure 12. HECS [15-16]

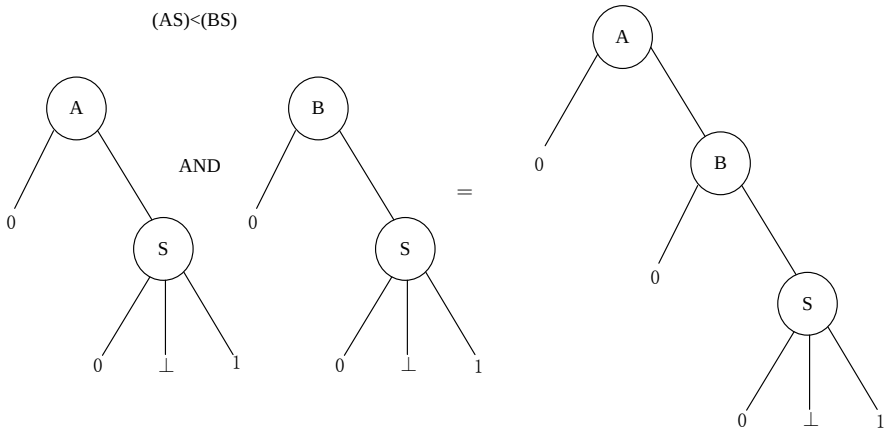


Figure 13. Generating MDD of the HECS

Table 1. Reliability results for the HECS

t (days)	Markov-based method	SBDD method	MDD
300	0.06257	0.06257	0.06257
600	0.24983	0.24983	0.24983
1000	0.51137	0.51137	0.51137

6. Conclusions

In this paper, we have presented a novel reliability analysis method of cold standby systems based on the MDD via static state transformation. The MDD can be applicable to both cold standby systems (DFTs model with cold spare gates) and static systems (SFT models) with any arbitrary component time-to-failure distribution and different component failure parameter values. In our proposed method, some specific states in cold spare gates replace sequence-dependent relations to describe dynamic fault behaviors in cold standby systems. Additionally, we have also proven that our MCSs with supplement events can be explained by sequence-dependent failure behaviors used in algebraic-structure-based methods. The quantitative analysis based on the MDD has been presented by means of a hypothetical example computer system.

References

1. J. B. Dugan, S. J. Bavuso, and M. A. Boyd, "Dynamic Fault-Tree Models for Fault-Tolerant Computer Systems," *IEEE Transactions on Reliability*, Vol. 41, No. 3, pp. 363-377, 1992
2. W. Vesely, M. Stamatelatos, J. Dugan, J. Fragola, J. Minarick III, and J. Railsback, "Fault Tree Handbook with Aerospace Applications Version 1.1," NASA Office of Safety and Mission Assurance, NASA HQ, 2002
3. K. B. Misra, "Handbook of Performability Engineering," Springer Science & Business Media, 2008
4. W. Long, "On the Quantitative Analysis of Sequential Failure Logic using Monte Carlo Method for Different Distributions," in *Proceedings of Probabilistic Safety Assessment and Management*, pp. 391-396, 2002

5. G. Merle, J. -M. Roussel, J. -J. Lesage, V. Perchet, and N. Vayatis, "Quantitative Analysis of Dynamic Fault Trees based on the Coupling of Structure Functions and Monte Carlo Simulation," *Quality and Reliability Engineering International*, Vol. 32, No. 1, pp. 7-18, 2016
6. H. Boudali and J. B. Dugan, "A Discrete-Time Bayesian Network Reliability Modeling and Analysis Framework," *Reliability Engineering & System Safety*, Vol. 87, No. 3, pp. 337-349, 2005
7. S. Kabir, M. Walker, and Y. Papadopoulos, "Reliability Analysis of Dynamic Systems by Translating Temporal Fault Trees into Bayesian Networks," in Model-based Safety and Assessment, pp. 96-109, Springer, 2014
8. G. Merle, J. M. Roussel, and J. J. Lesage, "Algebraic Determination of the Structure Function of Dynamic Fault Trees," *Reliability Engineering & System Safety*, Vol. 96, No. 2, pp. 267-277, 2011
9. R. E. Bryant, "Graph-based Algorithms for Boolean Function Manipulation," *IEEE Transactions on Computers*, Vol. 100, No. 8, pp. 677-691, 1986
10. A. Rauzy, "New Algorithms for Fault Trees Analysis," *Reliability Engineering & System Safety*, Vol. 40, No. 3, pp. 203-211, 1993
11. L. D. Xing, A. Shrestha, L. Meshkat, and W. D. Wang, "Incorporating Common-Cause Failures into the Modular Hierarchical Systems Analysis," *IEEE Transactions on Reliability*, Vol. 58, No. 1, pp. 10-19, 2009
12. L. Xing and Y. Dai, "A New Decision-Diagram-based Method for Efficient Analysis on Multistate Systems," *IEEE Transactions on Dependable and Secure Computing*, Vol. 6, No. 3, pp. 161-174, 2009
13. L. Xing and J. B. Dugan, "A Separable Ternary Decision Diagram based Analysis of Generalized Phased-Mission Reliability," *IEEE Transactions on Reliability*, Vol. 53, No. 2, pp. 174-184, 2004
14. T. Sasao, "Ternary Decision Diagrams: Survey," in *Proceedings of International Symposium on Multiple-Valued Logic*, pp. 241, 1997
15. L. Xing, A. Shrestha, L. Meshkat, and W. Wang, "Incorporating Common-Cause Failures into the Modular Hierarchical Systems Analysis," *IEEE Transactions on Reliability*, Vol. 58, No. 1, pp. 10-19, 2009
16. J. B. Dugan, B. Venkataraman, and R. Gulati, "Diftree: A Software Package for the Analysis of Dynamic Fault Tree Models," in *Proceedings of Symposium on Reliability and Maintainability*, pp. 64-70, 1997
17. L. Xing, O. Tannous, and J. B. Dugan, "Reliability Analysis of Nonrepairable Cold-Standby Systems using Sequential Binary Decision Diagrams," *IEEE Transactions on Systems, Man, and Cybernetics-Part A*: S*ystems and Humans*, Vol. 42, No. 3, pp. 715-726, 2011