Reliability Engineering and System Safety 66 (1999) 157–170

RELIABILITY ENGINEERING & SYSTEM SAFETY

www.elsevier.com/locate/ress

An algorithm for reliability analysis of phased-mission systems

Y. Ma[a,*], K.S. Trivedi[b]

[a]Center for Advanced Computing and Communication (CACC), Department of Computer Science, Duke University, Box 90129, Durham, NC 27708-0129, USA
[b]Center for Advanced Computing and Communication (CACC), Department of Electrical and Computer Engineering, Duke University, Box 90291, Durham, NC 27708-0291, USA

Received 2 November 1998; accepted 13 February 1999

Abstract

The purpose of this paper is to describe an efficient Boolean algebraic algorithm that provides exact solution to the unreliability of a multi-phase mission system where the configurations are described through fault trees. The algorithm extends and improves the Boolean method originally proposed by Somani and Trivedi. By using the Boolean algebraic method, we provide an efficient modeling approach which avoids the state space explosion and the mapping problems that are encountered by the Markov chain approach. To calculate the exact solution of the phased-mission system with deterministic phase durations, we introduce the sum of disjoint phase products (SDPP) formula, which is a phased-extension of the sum of disjoint products (SDP) formula. Computationally, the algorithm is quite efficient because it calls an SDP generation algorithm in the early stage of the SDPP computation. In this way, the phase products generated in the early stage of the SDPP formula are guaranteed to be disjoint. Consequently, the number of the intermediate phase products is greatly reduced. In this paper, we also consider the transient analysis of the phased-mission system. Special care is needed to account for the possible latent failures at the mission phase change times. If there are more stringent success criteria just after a mission phase change time, an unreliability jump would occur at that time. Finally, the algorithm has been implemented in the software package SHARPE. With SHARPE, the complexities of the phased-mission system is made transparent to the potential users. The user can conveniently specify a phased-mission model at a high level (through fault trees) and analyze the system quantitatively. © 1999 Elsevier Science Ltd. All rights reserved.

Keywords: Binary decision diagrams (BDD); Boolean algebraic methods; Cold/hot spares; Fault trees; Latent failure; Phased-mission systems; SHARPE; Software reliability; Software reusability; Transient analysis

Nomenclature

CFC	common failure combinations
DPC	disjoint phase constituent
DPP	disjoint phase product
MPCT	mission phase change time
PFC	phase failure combinations
PMS	phased-mission system
SDP	sum of disjoint products
SDPP	sum of disjoint phase products
C_{ji}	the event that component C_j fails during a time period of duration T_i
PE_i	the event describing the failure combinations (a set of mincuts) for Phase i, $1 \le i \le p$
T_i	duration of Phase i
UR	unreliability
p	total number of phases
p_e	index of the last phase for evaluation, $1 \le p_e \le p$
t_e	evaluation time within Phase p_e, $0 \le t_e \le T_{p_e}$

1. Introduction

Most reliability techniques and tools generally assume that the systems being analyzed perform a single phased-mission. With the increased use of automation in industries such as aerospace, chemical, communication networks, electronics, transportation and nuclear, phased-mission system (PMS) analysis is being recognized as an appropriate reliability analysis method for a large number of problems [1]. Many systems perform a mission which can be divided into consecutive time periods (phases). In each phase, the system needs to accomplish a specific task. The system configuration (the logic model), the phase duration, and the failure rates of the components often vary from phase to phase. We are interested in finding out the reliability/unreliability of the system either at the end of the mission or at any moment within the mission period.

* Corresponding author.
E-mail addresses: yuem@cs.duke.edu (Y. Ma), kst@ee.duke.edu (K.S. Trivedi)

The existence of more than one phase in a phased-mission system leads to some complexities which do not occur in a single phased-system. The problem arises because the models of different phases are dependent. The techniques used to reflect this dependence distinguish different approaches [2–10] employed for the analysis of PMS. For example, in the Markov chain approach [2], the dependence is reflected by assigning the state probabilities at the end of one phase as the initial state probabilities for the immediately following phase. In the combinatorial approach [4], the dependence can be accounted for by replacing a component C_j in Phase i with a series of s-independent components $C_{j,1}, C_{j,2}, \ldots$, and $C_{j,i}$. Then the reliability block diagrams (RBD) of different phases are connected in series to get the equivalent single phased-system. Although the original paper used RBDs, an equivalent fault tree approach can be envisaged [1].

Both of the above approaches have advantages and drawbacks. The advantage of the Markov chain approach is that it can reflect the dynamic behavior such as transient fault recovery. However, with the Markov approach, there is generally a state space explosion problem. For a system composed of n components, we may need up to 2^n states to represent each phase. In addition, since configurations are generally different from phase to phase, special care is needed to map the up states from one phase to the up states in the immediately following phase. The combinatorial approach is conceptually simple, but its size grows with the number of phases. This is due to the need to represent the same component many times. An efficient way to reduce the combinatorial explosion is to encode the Boolean expressions of the fault trees by means of binary decision diagrams (BDD) [11–13]. Recently, a PMS algorithm based on BDD was proposed [14]. Considerable reduction in computing and storage requirements is achieved through this algorithm.

In this paper, we propose an algorithm to determine the reliability of a PMS with deterministic phase durations. The algorithm is based on a Boolean method originally proposed by Somani and Trivedi [15] (ST algorithm). Our approach involves the solution of multiple single-phased fault trees. This is actually a divide and conquer strategy which is computationally more efficient than the combinatorial approach that combines the fault trees of all the phases into a single fault tree with repeated events. Our methodology also avoids the problems (state space explosion and mapping) faced by the Markov approach. In addition, the algorithm is general enough to apply to a wide range of problems. For example, the algorithm can handle repeated components, dormant components or *k-out-of-n* gates in some phases. These features are very important for the reliability analysis of ultra-reliable systems. For convenience and clear reference, we refer to our algorithm as the MT algorithm.

To calculate the unreliability of a PMS, we introduce the sum of disjoint phase products (SDPP) formula, which is a phased-extension of the sum of disjoint products (SDP) formula. The MT algorithm is based on the SDPP formula. We show that the original ST algorithm is also based on SDPP. Consequently, the proof of correctness of the ST algorithm follows as well.

To improve the computational efficiency over the ST algorithm, the MT algorithm calls an SDP generation algorithm in the early stage of the SDPP formula. In this way, the phase products generated in the early stage of the SDPP computation are guaranteed to be disjoint. As a result, the number of the intermediate phase products is greatly reduced.

In KITT-1 [1], which is a computer program using the combinatorial approach for the PMS analysis, when a component is dormant in a phase, it is handled as a hot spare. That is, under the constant failure rate assumption, the component failure rates are assumed to be the same, whether the component is dormant or not. In a real system, the failure rate of a dormant component is generally much less than the failure rate of an active component. It is a well accepted practice to approximate the failure rate of a dormant component to be zero. Therefore, in the MT algorithm, a dormant component in a phase is regarded as a cold spare and its default failure rate is assumed to be zero.

The transient analysis of a PMS is also considered in this paper. Special attention needs to be paid to possible latent failures at the mission phase change times (MPCT). If there are more stringent success criteria just after an MPCT, an unreliability jump would occur at that time. Finally, the MT algorithm is implemented into the software package SHARPE (Symbolic Hierarchical Automated Reliability and Performance Evaluator) [16]. With SHARPE, we can automatically analyze the unreliability of a multi-phase mission system, both at the end of the whole mission and at any time in between. Several examples that illustrate the MT algorithm are provided.

The rest of the paper is organized as follows. Section 2 presents some of the key concepts that will be used in the MT algorithm. In Section 3, the SDPP formula is first introduced. Then we describe and compare the ST and MT algorithms. A comparative example is also given there. The transient analysis of the PMS is described in Section 4. The SHARPE implementation is introduced in Section 5. In Section 6, more examples of the PMS analysis are given. Experimental results of MT as implemented in SHARPE are reported. We conclude the paper in Section 7. The Appendix shows the errors we found in Ref. [1] together with the corrections.

2. Preliminary concepts

2.1. Distribution functions with mass at origin

One of the key concepts we will use in the MT algorithm is the cumulative distribution functions with a mass at the

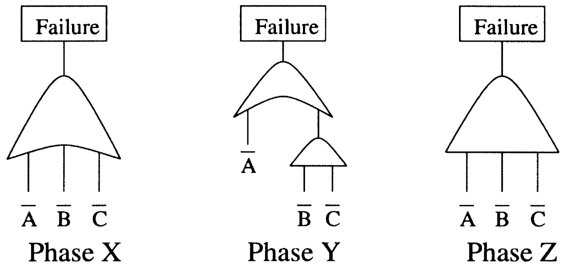

Fig. 1. The failure criteria for a system with three phases.

origin. Consider a random variable C_{jp} with cumulative distribution function (CDF) [16] given by

$$F_{C_{jp}}(t) = (1 - e^{-\lambda_{j1}T_1}) + e^{-\lambda_{j1}T_1}(1 - e^{-\lambda_{j2}T_2}) + \cdots + e^{-\lambda_{j1}T_1}e^{-\lambda_{j2}T_2}\cdots e^{-\lambda_{j(p-1)}T_{(p-1)}}(1 - e^{-\lambda_{jp}t}), \quad (1)$$

where λ_{ji} is the failure rate for component C_j in Phase i, T_i is the phase duration for Phase i, $t \in [0, T_p]$ and $1 \le i \le p$. The time origin for each phase is *reinitialized* to zero at the start of the phase. The above function has a mass at the origin given by $F_{C_{jp}}(0) = 1 - e^{-\sum_{i=1}^{p-1}\lambda_{ji}T_i}$. The last term of CDF (1) represents the continuous part of the distribution function. After simplification, CDF (1) turns out to be:

$$F_{C_{jp}}(t) = 1 - \left[e^{-\sum_{i=1}^{p-1}\lambda_{ji}T_i} \right] e^{-\lambda_{jp}t}. \quad (2)$$

In order to illustrate the use of such a CDF, consider a PMS that has just completed the first $(p - 1)$ phases and is currently in the pth phase. The above CDF can be assigned as the time to failure distribution function of component C_j in the pth phase. Recall that $F_{C_{jp}}(t)$ is the probability that component C_j fails at a time point τ, where $0 \le \tau \le t + \sum_{i=1}^{p-1} T_i$. The probability mass at the origin is the probability that the component has already failed at a time point within the first $(p - 1)$ phases. The factor in the square brackets of CDF (2) is the probability that component C_j has survived in the first $(p - 1)$ phases. We will use the distribution function of form (2) to represent the failure CDF of individual components in different phases.

2.2. Phase manipulation

To describe the MT algorithm, we use a three-component system (Fig. 1) as a sample example. To show the effect of the PMS analysis, we will consider all the six permutations of phases X, Y and Z. That is, the mission can go through all the three phases in any order.

Table 1
The effect of mincut cancellation

Before cancellation	After cancellation
$PE_{1X} = \overline{A_1} + \overline{B_1} + \overline{C_1}$	$PE_{1X} = \overline{B_1} + \overline{C_1}$
$PE_{2Z} = \overline{A_2}\ \overline{B_2}\ \overline{C_2}$	$PE_{2Z} = \varnothing$
$PE_{3Y} = \overline{A_3} + \overline{B_3}\ \overline{C_3}$	$PE_{3Y} = \overline{A_3} + \overline{B_3}\ \overline{C_3}$

Unless otherwise specified, the sequence number of a phase is represented by a lower case letter, and the name of a phase is denoted by an upper case letter. Let $A_i = 1$ denote the event that component A is up in phase number i. Then the Boolean expressions for the phases with names X, Y and Z are:

$$PE_{iX} = \bar{A}_i + \bar{B}_i + \bar{C}_i,$$

$$PE_{iY} = \bar{A}_i + \bar{B}_i\bar{C}_i,$$

$$PE_{iZ} = \bar{A}_i\bar{B}_i\bar{C}_i.$$

Before applying any algorithm to analyze a PMS, it is generally a good practice to simplify the configurations of the PMS by applying the *mincut cancellation rule* [4]:

A mincut for a phase can be cancelled from the list of mincuts for that phase if it contains a mincut of a later phase.

The rule can be proved by applying the law of absorption. Unless otherwise stated, we assume that we are interested in finding the unreliability of a PMS at the end of the mission and not at a time point before the mission completion. Table 1 shows the effect of applying the mincut cancellation rule for a PMS with a phase sequence of X, Z and Y. Mincut $\{\bar{A}\}$ is cancelled from PE_{1X} because it contains the mincut $\{\bar{A}\}$ of PE_{3Y}. Similarly, mincut $\{\bar{A}, \bar{B}, \bar{C}\}$ is cancelled from PE_{2Z}.

When an expression for an SDPP is simplified, we need to merge different combinations of phase products. This could be a little tricky and needs special treatment. Let i and j be two phase numbers and $i \le j$. The reduction rules [15] in Table 2 can be used to simplify the logic expressions.

Note that for $i < j$, two kinds of combination of phase products cannot be simplified.

1. Event $A_i\bar{A}_j$ describes the fact that component A is operational until the end of Phase i and fails sometime between the end of Phase i and the end of Phase j. Using CDF (2),

$$\Pr[A_i\bar{A}_j] = e^{-\sum_{m=1}^{i}(\lambda_m T_m)}\left(1 - e^{-\sum_{n=i+1}^{j-1}(\lambda_n T_n)} e^{-\lambda_j t}\right), \quad (3)$$

where $0 \le t \le T_j$. The first term $(e^{-\sum_{m=1}^{i}(\lambda_m T_m)})$ of Eq. (3) represents that component A is up in the first i phases. The second term represents that component A is down sometime between Phases i and j. Since the first i phases have already been considered in the first term, in the second term, the failure rate λ_n is accumulated starting from Phase $i + 1$.
2. Event $\bar{A}_i + A_j$ represents the fact that either component A is down at the end of Phase i or it is still functioning at the end of Phase j. Notice that event $\bar{A}_i + A_j$ is a disjoint union.

Table 2
Reduction rules

AND	OR
$A_iA_j \rightarrow A_j$	$\overline{A_i} + \overline{A_j} \rightarrow \overline{A_j}$
$\overline{A_i}\,\overline{A_j} \rightarrow \overline{A_i}$	$A_i + A_j \rightarrow A_i$
$\overline{A_i}A_j \rightarrow 0$	$A_i + \overline{A_j} \rightarrow 1$

3. Algorithm

3.1. Sum of disjoint products and its phased-extension: sum of disjoint phase products

The sum of disjoint products (SDP) formula is one of the techniques [17,18] that is used to compute the probability of a union of a set of events in a single-phased system. Since this paper uses fault trees to specify the system configurations, we will use the mincuts to illustrate how SDP works. Let E_i be the event that all the components in the mincut MC_i fail. In other words, the event E_i is a Boolean expression describing a *single* mincut MC_i. The SDP formula for calculating the unreliability of the system is:

$$\mathrm{UR}(S) = \Pr\left[\bigcup_{i=1}^{n} E_i\right] = \Pr\Big[E_1 \bigcup (\bar{E}_1E_2) \bigcup (\bar{E}_1\bar{E}_2E_3) \bigcup \cdots \bigcup (\bar{E}_1\bar{E}_2 \cdots \overline{E_{n-1}}E_n)\Big], \tag{4}$$

where n is the total number of mincuts. Define the constituent $CS_1 = E_1$ and in general, $CS_i = \bar{E}_2\bar{E}_2 \cdots \overline{E_{i-1}}E_i$ where $1 \le i \le n$. Since the constituents CS_i in Eq. (4) are disjoint from each other, the final SDP formula for calculating the unreliability of the system is:

$$\mathrm{UR}(S) = \sum_{i=1}^{n} \Pr(CS_i). \tag{5}$$

The crux of the SDP formula is to obtain the disjoint constituent CS_i, for $i > 1$. Several algorithms have been published for this calculation. A survey that compares these algorithms can be found in Ref. [19].

To calculate the unreliability of a PMS, we extend the sum of disjoint products formula into the sum of disjoint phase products (SDPP) formula. Let PE_i be the event that a PMS is down in Phase i. The SDPP formula for calculating the unreliability of the PMS is:

$$\mathrm{UR}(PMS) = \Pr\left[\bigcup_{i=1}^{p} PE_i\right] \tag{6}$$

$$= \Pr\Big[PE_1 \bigcup (\overline{PE_1}\ PE_2 \bigcup (\overline{PE_1}\ \overline{PE_2}PE_3) \bigcup \cdots \bigcup (\overline{PE_1}\ \overline{PE_2} \cdot \cdots \cdot \overline{PE_{p-1}}PE_p)\Big], \tag{7}$$

where p is the total number of phases for the PMS. In Eq. (4), event E_i represents one *single* mincut. In Eqs. (6) and (7), event PE_i represents a *set* of mincuts, in which the mincuts are generally non-disjoint. The complement of PE_i is normally a set of non-disjoint phase products as well. Define the phase constituent $PC_1 = PE_1$ and in general, $PC_i = \overline{PE_1}\ \overline{PE_2} \cdots \overline{PE_{i-1}}PE_i$, where $1 < i \le p$. Generally, the phase products in each PC_i are non-disjoint. If the phase products in a PC_i are mutually disjoint, the PC_i is defined as a *disjoint phase constituent*, denoted by DPC_i. One of the challenges in using the SDPP formula is to change the PC_i into DPC_i. In the following sections, we will explain in detail how this problem is solved in the ST and MT algorithms. Once the DPC_i are found, the final SDPP formula for calculating the unreliability of the PMS is:

$$\mathrm{UR}(PMS) = \sum_{i=1}^{p} \Pr[DPC_i]. \tag{8}$$

In ST [15], the unreliability of a PMS is given by

$$\mathrm{UR}(PMS) = \Pr[CFC] + \sum_{i=1}^{p-1} \Pr[PFC_i], \tag{9}$$

where $CFC = PE_p$ and is referred as the *Common Failure Combinations* (CFC), $PFC_i = PE_i\overline{PE_{i+1}} \cdots \overline{PE_{(p-1)}}\ \overline{PE_p}$, and is referred as a *Phase Failure Combination* (PFC).

Proposition 1. The ST algorithm, which is based on Eq. (9), is the right to left (RL-SDPP) evaluation of the SDPP formula (6).

Proof. Straight forward by comparing Eqs. (6) and (9). □

In MT, we use the SDPP formula as the core of the algorithm. In addition, the SDPP is evaluated from left to right (LR-SDPP), as it is conventionally done in evaluating SDP. Furthermore, we use mincut cancellation in MT while this was not done in ST.

In the following sections, we describe the main operations that are used in the ST and MT algorithms. We first present the ST algorithm in Section 3.2. Then we show the MT algorithm and its improvements over the ST algorithm

```
1. $find\_mincuts(PMS)$;
2. $CFC = find\_SDP(PE_p)$;
3. $add\_phase\_numbers(CFC)$;
4. $UR(PMS) = Pr[CFC]$;
5. for $i = 1, 2, \cdots, p-1$ do
       {
           $PFC_i = find\_PFC(i)$;
           $UR(PMS) ++ = Pr[PFC_i]$;
       }
```

Fig. 2. The core of the ST algorithm.

```
1. $PFC_i = PE_i$;
2. $add\_phase\_numbers(PFC_i)$;
3. for $j = (i+1), (i+2), \cdots, p$ do
     {
       $G_j = inverse(PE_j)$;
       $add\_phase\_numbers(G_j)$;
       $PFC_i = multiply\_PFC(PFC_i, G_j)$;
     }
4. $PFC_i = find\_SDPP(PFC_i)$;
5. return($PFC_i$);
```

Fig. 3. Routine *find_PFC*(*i*).

in Section 3.3. In Section 3.4, an example is given which illustrates the improvements in MT by using a detailed comparative listing of the steps in applying MT and ST.

3.2. The ST algorithm

In the ST algorithm (Fig. 2), the mincuts of each phase are first found in Step 1 by applying one of the mincut generation algorithms [20–22]. Since PE_p is a union of the mincuts for the last phase, the SDP for *CFC* is calculated by calling routine *find_SDP* which applies one of the SDP generation algorithms [23–28]. The subscript *p* in PE_p at Step 2 is used for the convenience of identification. No phase numbers have been added to the expression yet. In Step 3, routine *add_phase_numbers* includes the phase information to the SDP of *CFC*. In Step 4, the probability for *CFC* is calculated by using the distribution functions with mass at origin, which is discussed in Section 2.1. In Step 5, each PFC_i is found and the system unreliability is updated. Fig. 3 shows routine *find_PFC,* which is used to generate an SDPP for $PFC_i = PE_i\overline{PE_{i+1}}\cdots\overline{PE_{(p-1)}}\ \overline{PE_p}$, where $1 \le i < p$.

In routine *find_PFC,* the complement of PE_j is obtained through *routine inverse* and is assigned to G_j. In routine *multiply_PFC* (Fig. 4), list *l* is first initialized to be empty. In Steps 2 and 3, the first phase products of PFC_i and G_j are assigned to *u* and *v* respectively. Routine *multiply* combines one phase product of PFC_i and one phase product of G_j into a new phase product *tmp*. The reduction rules as shown in Table 2 are applied for the combination. Since the phase products in PFC_i and G_j are generally not disjoint, we need to check if the newly generated phase product *tmp* is a subset or a superset of one of the phase products in the list *l*. This is done by calling routine *compare*. Three scenarios might happen:

1. If *tmp* is a subset, then nothing is done for *l* in the routine *sort_and_insert*.
2. If *tmp* is a superset, then the phased products which are subsets of *tmp* are eliminated from *l*. And routine *sort_and_insert* adds *tmp* in *l* to its proper place. The size of a phase product is defined as its *cardinality*. Notice that the phase products in *l* are sorted (through routine *sort_and_insert*) according to the increasing cardinality of the phase products. This is done to make routine *find_SDPP* (used in routine *find_PFC*) more efficient. Similar implementations of *compare* and *sort_and_insert* in SDP algorithms have been reported in Refs. [23,26,27].
3. If *tmp* is neither a subset nor a superset, routine *sort_and_insert* adds *tmp* in *l* according to *tmp*'s cardinality.

```
1. $l = \varnothing$;
2. $u = PFC_i \to head$;
3. $v = G_j \to head$;
4. while $u \neq NIL$ do
     {
       while $v \neq NIL$ do
         {
           $tmp = multiply(u, v)$;
           $compare(l, tmp)$;
           $sort\_and\_insert(l, tmp)$;
           $v = v \to next$;
         }
       $u = u \to next$;
     }
5. return($l$);
```

Fig. 4. Routine *multiply_PFC*(PFC_i, G_j).

In *multiply_PFC*, since the phase products in PFC_i and G_j are generally not disjoint, the resulting phase products for *multiply_PFC* are normally not disjoint as well. This is the reason why routine *find_SDPP* is needed in routine *find_PFC*. Routine *find_SDPP* applies formula (7) to get an SDPP for PFC_i. Notice that when using the SDPP formula (7) for routine *find_SDPP*, event PE_i is no longer the *set* of mincuts for Phase *i*. Instead, it represents a *single* phase product in PFC_i. Again the procedures *compare* and *sort_and_insert* are needed in *find_SDPP*.

3.3. The MT algorithm

From the analysis in the above section, it is seen that the complexities of the ST algorithm arise mainly from three routines: *find_SDPP*, *compare* and the sorting process in *sort_and_insert*. In the MT algorithm (Fig. 5), two major steps are taken to improve the computational efficiency:

- First, the logic expressions for a PMS are simplified by applying the mincut cancellation rule (Step 2 in Fig. 5).
- Second, by using routine *find_SDP* in Steps 1 and 3 in *find_DPC* (Fig. 6), which is the counterpart of *find_PFC* in the MT algorithm, the phased products generated in the early stage of the SDPP computation are guaranteed to be disjoint. Consequently, the routines *find_SDPP*, *compare* and the sorting process in *sort_and_insert* are no longer necessary in the MT algorithm.

Because of the above modifications, the MT algorithm is computationally more efficient than the ST algorithm.

In MT, Steps 3 and 4 are similar to the Steps 2–4 in the ST (Fig. 2). The only difference is that routine *add_phase_numbers* does not need to be applied to PE_1. This is because PE_1 describes the failure combinations for Phase 1, no

```
1. $find\_mincuts(PMS)$;
2. $mincut\_cancellation(PMS)$;
3. $DPC_1 = find\_SDP(PE_1)$;
4. $UR(PMS) = Pr[DPC_1]$;
5. for $i = 2, \cdots, p$ do
     {
       $DPC_i = find\_DPC(i)$;
       $UR(PMS) ++ = Pr[DPC_i]$;
     }
```

Fig. 5. The core of the MT algorithm.

previous phase information is needed for the evaluation of Pr[DPC_1]. Routine *find_DPC* (Fig. 6) is used to generate an SDPP for $DPC_i = \overline{PE_1}\,\overline{PE_2}\cdots\overline{PE_{i-1}}PE_i$, where $1 < i \leq p$. In the first for loop in routine *find_DPC*, after routine *find_SDP* is called, each of the operands DPC_i and G_1 is an SDP. With the phase information being added by calling the routine *add_phase_numbers*, each of the operands DPC_i and G_1 turns out to be an SDPP. By applying the reduction rules in Table 2, routine *multiply_DPC* (Fig. 7) combines two SDPP into a single SDPP. Because the product of two disjoint products is always disjoint, after the first for loop in routine *find_DPC*, DPC_i is an SDPP. Then DPC_i is recursively updated by the multiplication of two SDPP, thus after Step 3, DPC_i is already an SDPP. This implies that routine *find_SDPP*, which is used in *find_PFC*, is no longer needed in *find_DPC*. As will be shown in Section 3.4, this would greatly improve the efficiency of the calculation by reducing the number of intermediate phase products.

Comparing routines *multiply_PFC* and *multiply_DPC*, we can find two major improvements in *multiply_DPC*:

1. Due to the *find_SDP* routine in *find_DPC*, the operands DPC_i and G_j are always disjoint with each other in routine *multiply_DPC*. As a result, the newly generated phase product *tmp* is always disjoint from any phase product in the list *l*. Thus, routine *compare* in *multiply_PFC* is no longer necessary in *multiply_DPC*.
2. Because routine *find_SDPP* is not needed in *find_DPC*, the sorting process in *sort_and_insert* is no longer necessary for *multiply_DPC*. Thus routine *sort_and_insert* is replaced by a simple routine *insert*, where *tmp* is just added at the end of list *l*. With respect to cardinality, the list *l* in *multiply_DPC* is a random set of the phased products, rather than a sorted one in routine *multiply_PFC*. However, this would not make any difference for the final calculation of Pr[DPC_i].

```
1. $DPC_i = find\_SDP(PE_i)$;
2. $add\_phase\_numbers(DPC_i)$;
3. for $j = 1, 2, \cdots, i-1$ do
     {
       $G_j = inverse(PE_j)$;
       $G_j = find\_SDP(G_j)$;
       $add\_phase\_numbers(G_j)$;
       $DPC_i = multiply\_DPC(DPC_i,\ G_j)$;
     }
4. return($DPC_i$);
```

Fig. 6. Routine *find_DPC*(*i*).

```
1. $l = \varnothing$;
2. $u = DPC_i \rightarrow head$;
3. $v = G_j \rightarrow head$;
4. while $u \neq NIL$ do
     {
       while $v \neq NIL$ do
         {
           $tmp = multiply(u,\ v)$;
           $insert(l, tmp)$;
           $v = G_j \rightarrow next$;
         }
       $u = DPC_i \rightarrow next$;
     }
5. return($l$);
```

Fig. 7. Routine *multiply_DPC*(DPC_i, G_j).

The above two changes, which are made possible by applying an SDP generation algorithm in the earlier stage (Steps 1 and 3 in routine *find_DPC*) of the SDPP formula, greatly improve the efficiency of routine *multiply_DPC*.

As we mentioned earlier in Section 3.2, in an SDP generation algorithm, there are also routines for the *compare* and *sort* mechanisms. However, these are used only for a single phased-system, which is more efficient than for multi-phase systems. The strategy we use here is to *compare* and *sort* in an earlier stage where only a single phase is involved. In ST, this is done in a later and more complex stage, i.e. after the phase products are entangled with information from multiple phases. The effect of these two approaches can be shown in the following example.

3.4. Example 1: an illustration

In this example, we apply the MT algorithm to find the unreliability of a PMS with configurations shown in Fig. 1. The permutation of X, Y and Z is considered. A detailed comparison between the ST and the MT is also shown here. Only the logical expressions are given in this section. The final numerical results are given in Sections 5 and 6.

Using the MT algorithm:

$DPC_1 = PE_{1X}$

(after the mincut cancellation)
$= \overline{B_1} + \overline{C_1}$
(after calling an SDP generation algorithm)
$= \overline{B_1} + B_1\overline{C_1}$,

$DPC_2 = \overline{PE_{1X}}PE_{2Y}$

$= (\overline{\overline{B_1} + \overline{C_1}})(\overline{A_2} + \overline{B_2}\;\overline{C_2})$
(before applying *multiply_DPC* in *find_DPC*(2))
$= B_1C_1(\overline{A_2} + A_2\overline{B_2}\;\overline{C_2})$
(after the for loop in *find_DPC*(2))
$= \overline{A_2}B_1C_1 + A_2B_1\overline{B_2}C_1\overline{C_2}$,

$DPC_3 = \overline{PE_{1X}}\,\overline{PE_{2Y}}PE_{2Z}$

$= \overline{(\overline{B_1} + \overline{C_1})(\overline{A_2} + \overline{B_2}\,\overline{C_2})}\,\overline{A_3}\,\overline{B_3}\,\overline{C_3}$

(before applying *multiply_DPC* in *find_DPC*(3) when $j = 1$)

$= B_1C_1\overline{(\overline{A_2} + \overline{B_2}\,\overline{C_2})}\,\overline{A_3}\,\overline{B_3}\,\overline{C_3}$

(after the first for loop in *find_DPC*(3))

$= \overline{A_3}B_1\overline{B_3}C_1\overline{C_3}\,\overline{(\overline{A_2} + \overline{B_2}\,\overline{C_2})}$

(before applying *multiply_DPC* in *find_DPC*(3) when $j = 2$)

$= \overline{A_3}B_1\overline{B_3}C_1\overline{C_3}(A_2B_2 + A_2\overline{B_2}C_2)$

(after the second for loop in *find_DPC*(3))

$= A_2\overline{A_3}B_2\overline{B_3}C_1\overline{C_3} + A_2\overline{A_3}B_1\overline{B_2}C_2\overline{C_3}.$

The final total number of disjoint phase products (DPP) for the MT algorithm is 6.

Using the ST algorithm:

$CFC = PE_{3Z}$

$= \overline{A_3}\,\overline{B_3}\,\overline{C_3},$

$PFC_1 = PE_{1X}\overline{PE_{2Y}}\,\overline{PE_{3Z}}$

(no mincut cancellation is applied in ST)

$= (\overline{A_1} + \overline{B_1} + \overline{C_1})\overline{(\overline{A_2} + \overline{B_2}\,\overline{C_2})}\,\overline{\overline{A_3}}\,\overline{\overline{B_3}}\,\overline{\overline{C_3}}$

(before applying *multiply_PFC* in *find_PFC*(1) when $j = 2$)

$= (\overline{A_1} + \overline{B_1} + \overline{C_1})(A_2B_2 + A_2C_2)\overline{\overline{A_3}\,\overline{B_3}\,\overline{C_3}}$

(after the first for loop in *find_PFC*(1))

$= (A_2B_2\overline{C_1} + A_2\overline{B_1}C_2)\overline{\overline{A_3}\,\overline{B_3}\,\overline{C_3}}$

(before applying *multiply_PFC* in *find_PFC* when $j = 3$)

$= (A_2B_2\overline{C_1} + A_2\overline{B_1}C_2)(A_3 + B_3 + C_3)$

(after the second for loop in *find_PFC*(1))

$= A_3B_2\overline{C_1} + A_3\overline{B_1}C_2 + A_2B_3\overline{C_1} + A_2\overline{B_1}C_3$

(now start the routine *find_SDPP*(PFC_1))

$= A_3B_2\overline{C_1} + (A_3\overline{B_1}C_2 + A_2B_3\overline{C_1} + A_2\overline{B_1}C_3)(\overline{A_3} + \overline{B_2} + C_1)$

(after routines *compare* and *sort_and_insert*)

$= A_3B_2\overline{C_1} + A_3\overline{B_1}C_2 + A_2\overline{B_1}C_3 + A_2\overline{A_3}B_3\overline{C_1}$

$= A_3B_2\overline{C_1} + A_3\overline{B_1}C_2 + (A_2\overline{B_1}C_3 + A_2\overline{A_3}B_3\overline{C_1})(\overline{A_3} + B_1 + \overline{C_2})$

(after routines *compare* and *sort_and_insert*)

$= A_3B_2\overline{C_1} + A_3\overline{B_1}C_2 + A_2\overline{A_3}\,\overline{B_1}C_3 + A_2\overline{A_3}B_3\overline{C_1}(\overline{A_2} + A_3 + B_1 + \overline{C_3})$

(after routines *compare* and *sort_and_insert*, now PFC_1 is disjoint)

$= A_3B_2\overline{C_1} + A_3\overline{B_1}C_2 + A_2\overline{A_3}\,\overline{B_1}C_3 + A_2\overline{A_3}B_3\overline{C_1},$

$PFC_2 = PE_{2Y}\overline{PE_{3Z}}$

$= (\overline{A_2} + \overline{B_2}\,\overline{C_2})\overline{\overline{A_3}\,\overline{B_3}\,\overline{C_3}}$

(before applying *multiply_PFC* in *find_PFC*(2) when $j = 3$)

$= (\overline{A_2} + \overline{B_2}\,\overline{C_2})(A_3 + B_3 + C_3)$

(after the for loop in *find_PFC*(2))

$= \overline{A_2}B_3 + \overline{A_2}C_3 + A_3\overline{B_2}\,\overline{C_2}$

(now start the routine *find_SDPP*(PFC_2))

$= \overline{A_2}B_3 + (\overline{A_2}C_3 + A_3\overline{B_2}\,\overline{C_2})(A_2 + \overline{B_3})$

(after routines *compare* and *sort_and_insert*)

$= \overline{A_2}B_3 + A_3\overline{B_2}\,\overline{C_2} + \overline{A_2}\,\overline{B_3}C_3(\overline{A_3} + B_2 + C_2)$

(after routines *compare* and *sort_and_insert*, now PFC_2 is disjoint)

$= \overline{A_2}B_3 + A_3\overline{B_2}\,\overline{C_2} + \overline{A_2}\,\overline{B_3}C_3.$

The final total number of DPP for the ST algorithm is 8. Comparing the two algorithms, we notice that the final total number of DPP for the MT algorithm is less than that of ST algorithm. More importantly, however, MT generates fewer *intermediate* phase products than ST. This is due to the embedded mincut cancellation rule and the SDP generation algorithm in the early stage of the SDPP computation.

4. Transient analysis

In the previous section, we have shown how the MT algorithm computes the unreliability of a PMS at the end of the whole mission. In other words, $\mathrm{UR}(PMS) = \sum_{i=1}^{p} \Pr[DPC_i]$ gives us the unreliability of the system at time point t, where $t = \sum_{i=1}^{p} T_i$. Sometimes, we might be interested in getting the unreliability information at any time point between the mission start time and the mission end time. Among these time points, special attention needs to be given to some certain time points: mission phase change times (MPCT). At the transition point between two phases, the unreliability of the PMS may rise suddenly. This would happen if more stringent criteria apply to the next phase than to the current phase. This is defined as the *latent failure* [10,29] because the system would fail instantly at MPCT. For example, when an aircraft is flying, it does not matter whether the landing gear is operational or not. However, as soon as the landing phase begins, if the landing gear has failed in an earlier phase, the system fails immediately.

Suppose that before an MPCT, the system is in Phase i, and after it, the system is in Phase $i + 1$. We assume that the transition time from one phase to another is so negligible that it is taken to be zero. If we wish to compute the unreliability of the system at MPCT, then the index of the last phase for evaluation, denoted by p_e, is $i + 1$. The evaluation time (t_e) for Phase p_e is 0. That is, it is the failure criteria in the later phase that account for the possible latent failure, not the duration of the later phase. In addition, for transient analysis of a PMS, the mincut cancellation applies only up to the first p_e phases.

As an example, consider a PMS (Fig. 1) with permutation Y, X and Z. If the phase durations are given as T_Y, T_X and T_Z, then the time points $t_1 = T_Y$ and $t_2 = T_Y + T_X$ are MPCT for the PMS. In the following we demonstrate how the unreliabilities of the PMS at MPCT are computed.

At t_1, the index of the last phase for evaluation is 2, $\mathrm{UR}(PMS)_{t_1} = \sum_{i=1}^{2} \Pr[DPC_i]$, where

$DPC_1 = PE_{1Y}$

Table 3
The effect of latent failure at MPCT

Time (hours)	System	UR $\times 10^{-3}$	System	UR $\times 10^{-3}$	UR jump $\times 10^{-3}$
10	PMS_{YXZ}	2.99550450	PMS_{Y}	1.00049817	1.99500633
20	PMS_{YXZ}	5.98203595	PMS_{YX}	5.98203595	0

(due to the mincut cancellation rule)
$= \emptyset$,

$$DPC_2 = \overline{PE_{1Y}}PE_{2X} = \overline{\emptyset}\cdot(\overline{A_2} + \overline{B_2} + \overline{C_2}) = \overline{A_2} + A_2\overline{B_2} + A_2B_2\overline{C_2},$$

$$\mathrm{UR}(PMS)_{t_1} = \Pr[DPC_2]$$

$$= \Pr[\overline{A_2}] + \Pr[A_2\overline{B_2}] + \Pr[A_2B_2\overline{C_2}]$$

$$= \Pr[\overline{A_1}] + \Pr[A_1\overline{B_1}] + \Pr[A_1B_1\overline{C_1}], \quad \text{since } t_e = 0. \quad (10)$$

At t_2, the index of the last phase for evaluation is 3, after applying mincut cancellation, $DPC_1 = \emptyset, DPC_2 = \overline{A_2} + A_2\overline{B_2} + A_2B_2\overline{C_2}$ and $DPC_3 = A_2\overline{A_3}B_2\overline{B_3}C_2\overline{C_3}$. Because the valuation time for Phase 3 is zero, $\Pr[DPC_3] = 0$. Thus

$$\mathrm{UR}(PMS)_{t_2} = \Pr[DPC_2]$$

$$= \Pr[\overline{A_2}] + \Pr[A_2\overline{B_2}] + \Pr[A_2B_2\overline{C_2}]. \quad (11)$$

Comparing Eqs. (10) and (11), we notice that the Boolean expressions for DPC_2 are the same at both time points t_1 and t_2. However, since the phase durations for the two cases are different, the final numerical values for $\Pr[DPC_2]$ are not the same.

Using the above analysis, we obtained the unreliabilities at MPCT for PMS_{YXZ}. In Table 3, the unreliabilities at the end of a single and double phased-systems are also presented. From these, we can find the possible unreliability jump due to the latent failure. If the failure criteria in a later

```
format 8

ftree X
basic a exp(a_x)
basic b exp(b_x)
basic c exp(c_x)
or top a b c
end

ftree Y
basic a exp(a_y)
basic b exp(b_y)
basic c exp(c_y)
and BC b c
or top BC a
end

ftree Z
basic a exp(a_z)
basic b exp(b_z)
basic c exp(c_z)
and ABC a b c
end

bind
a_x      0.0001
a_y      0.0001
a_z      0.0001
b_x      0.0001
b_y      0.0001
b_z      0.0001
c_x      0.0001
c_y      0.0001
c_z      0.0001
T_x      10
T_y      10
T_z      10
end

pms XYZ
phase-1  X  time(T_x)
phase-2  Y  time(T_y)
phase-3  Z  time(T_z)
end

pms XZY
phase-1  X  time(T_x)
phase-2  Z  time(T_z)
phase-3  Y  time(T_y)
end

pms YXZ
phase-1  Y  time(T_y)
phase-2  X  time(T_x)
phase-3  Z  time(T_z)
end

pms YZX
phase-1  Y  time(T_y)
phase-2  Z  time(T_z)
phase-3  X  time(T_x)
end

pms ZXY
phase-1  Z  time(T_z)
phase-2  X  time(T_x)
phase-3  Y  time(T_y)
end

pms ZYX
phase-1  Z  time(T_z)
phase-2  Y  time(T_y)
phase-3  X  time(T_x)
end

pms YX
phase-1 Y time(T_y)
phase-2 X time(T_x)
end

expr tvalue(10; YXZ)

expr tvalue(10; Y)

expr tvalue(20; YXZ)

expr tvalue(20; YX)

loop t, 0, 30, 2
  expr tvalue(t; XYZ)
  expr tvalue(t; XZY)
  expr tvalue(t; YXZ)
  expr tvalue(t; YZX)
  expr tvalue(t; ZXY)
  expr tvalue(t; ZYX)
end

end
```

Fig. 8. A SHARPE input file for PMS analysis.

```
  value(10; YXZ):    2.99550450e-03     |                ...
---------------------------------------|                ...
  value(10; Y):      1.00049817e-03     | t=30.000000
---------------------------------------|     tvalue(t; XYZ):    3.99300865e-03
  value(20; YXZ):    5.98203595e-03     |     tvalue(t; XZY):    4.99149291e-03
---------------------------------------|     tvalue(t; YXZ):    5.98203694e-03
  value(20; YX):     5.98203595e-03     |     tvalue(t; YZX):    8.95962123e-03
---------------------------------------|     tvalue(t; ZXY):    6.97654910e-03
                ...                     |     tvalue(t; ZYX):    8.95962123e-03
```

Fig. 9. Part of the SHARPE output for Fig. 8.

phase are more relaxed, then at MPCT, no latent failure would occur. This is true when the PMS_{YXZ} transfers from the second phase X to the third phase Z. There is no unreliability jump at MPCT t_2. All the phase durations in Table 3 are 10 h and the failure rate for each component is 0.0001/h.

If the evaluation time τ is between two MPCT, i.e. $\mathrm{MPCT}_i < \mathrm{MPCT}_{i+1}$, where $MPCT_i = \sum_{m=1}^{i} T_m$ and $1 \le i < p$, then the index of the last phase (denoted by p_e) for evaluation is $i + 1$. The evaluation time (t_e) for that phase is ($\tau - \mathrm{MPCT}_i$). The unreliability of the PMS at τ is evaluated according to $\mathrm{UR}(PMS)_\tau = \sum_{i=1}^{p_e} \Pr[DPC_i]$, where $T_{p_e} = t_e$.

5. The SHARPE implementation

The MT algorithm has been implemented in SHARPE [16], a software package that analyzes stochastic models. SHARPE was developed for modeling complex real-time systems. It has been used in over 220 academic institutions and industrial laboratories. The package provides a specification language and a wide variety of algorithms for analyzing reliability, availability, performance and performability models. In this section, we present a particular example which shows how to use SHARPE for end-of-mission and transient analysis of the PMS as shown in Fig. 1. All the examples mentioned in this paper can be handled by SHARPE, and all the numerical results presented in this paper were calculated by SHARPE.

Fig. 8 presents a SHARPE input file for end-of-mission and transient analysis of the PMS as shown in Fig. 1. Line numbers are included only for the sake of explanation. Line 1 specifies the number of digits to be printed in the results after the decimal point. On lines 3 through 23, the configurations for different phases are described by fault trees. In line 3, the definition of event $\overline{A}$ starts with the keyword *basic* because the event appears only once in the fault tree. Otherwise, the event would be specified by the keyword *repeat*. An example with repeated (shared) event will be presented in the next section. The exponential time to failure distribution is assigned to event $\overline{A}$ through the built-in function *exp* in line 3. The failure rate is *a_x*. Line 7 is a structure-defining line, defining the OR gate combining the events $\overline{A}$, $\overline{B}$ and $\overline{C}$. The gate is defined by using the keyword *or*, followed by the name of the gate (top) and the inputs (a, b and c) to the gate.

Lines 25 through 38 assign the failure rates for the components and the phase durations. In this example, all the phase durations are 10 hours and the failure rates are of the same value, 0.0001/h. Thus the input data do not skew results in any direction as all the components are similar and the phase durations are the same. The results are only affected by the sequence of the phases and the phase configurations. Lines 40 through 74 define the six permutations of the three phases. In line 41, the definition of the first phase starts with the keyword *phase-1*, followed by the name of the fault tree (X) for the first phase. The time duration for the first phase is assigned as *T_x* by using the keyword *time*. Lines 76 through 79 define a two phased system which will be used to check if an unreliability jump would occur at MPCT. The unreliabilities at time points 10 and 20 are evaluated for PMS_{YXZ}, the fault tree Y and PMS_{YX} from line 81 to line 87. The transient analysis of the six permutations is obtained from line 89 to line 96. From Fig. 8, it can be noticed that SHARPE supports the reusability [30] of the phase specification. For example, Phase Y is specified for once, but it can be used in different PMS. More importantly, the SHARPE implementation of the MT algorithm reuses the GKG_VT [26] as the SDP generation algorithm. The GKG_VT algorithm was implemented in SHARPE in 1989. Since then, it has been used successfully for the SDP generation. There are several advantages [30,31] to reuse software components in the software systems. Two related ones are listed below:

- Fewer errors will occur by reusing the already tested and used software components than developing new code from scratch. Consequently, by reusing the software components, the development and maintenance costs can be reduced, and the overall software reliability is enhanced.
- If components with sophisticated, efficient algorithms can be reused (in this case, the GKG_VT algorithm), the overall software product (in this case, the MT algorithm) will be more efficient. This is shown in Section 3.

Part of the output file for Fig. 8 is shown in Fig. 9. The transient results are shown in Fig. 10. From Fig. 10, we can see how the unreliability of the system is affected by

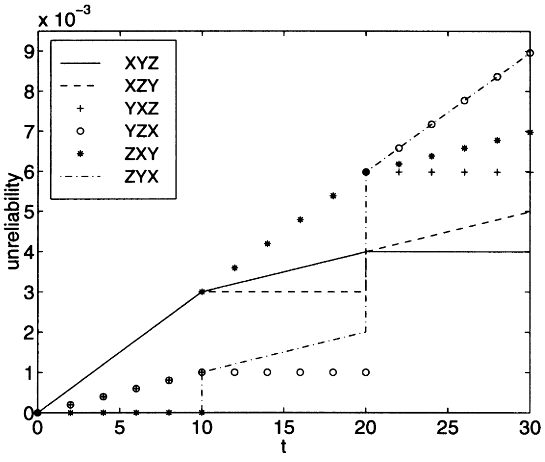

Fig. 10. The unreliability values for the six permutations.

the sequence of different phases. Unreliability jumps are observed at MPCT for all but one permutation (PMS_{XYZ}). Low unreliabilities in earlier phases are not predictive of low unreliabilities in later phases. For PMS_{YZX} and PMS_{ZYX}, the systems experience high unreliability jumps once they enter Phase 3. This is due to the stringent success criteria in Phase X. Armed with the transient analysis of PMS, the system designer can gain insights among alternative designs and select the best available scheme to enhance the robustness of the PMS.

6. Experimental results

In this section, we first describe five more examples that are used to test the MT algorithm. Then the execution times are given for the examples presented in this paper.

6.1. More examples to illustrate MT

6.1.1. Example 2: a PMS with shared components in some phases

In Fig. 11, component *B* is shared by two AND gates in Phase *R*. In the SHARPE input file (which is not shown in this paper), the keyword *repeat* instead of *basic* is used to indicate that event $\overline{B}$ appears more than once in the fault tree but represents only one physical component.

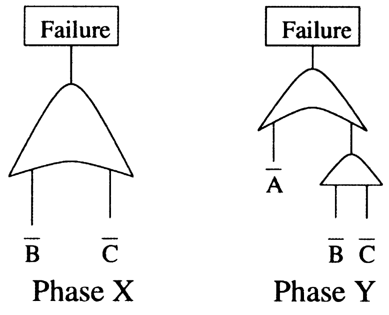

Fig. 12. A PMS where component *A* is dormant in Phase X.

6.1.2. Example 3: a PMS with dormant components in some phases

In this example, we show how the MT algorithm handles the PMS with dormant components in some phases. Suppose that we have a simple two-phased mission system as shown in Fig. 12. The phase sequence is assumed to be XY. Component *A* is dormant in Phase 1 (named as *X*). In KITT-1 [1], which is a computer program using the combinatorial approach for the PMS analysis, when a component is dormant in a phase, it is handled as a hot spare. That is, under the constant failure rate assumption, the component failure rates are assumed to be the same, whether the component is dormant or not. In a real system, the failure rate of a dormant component is generally much less than the failure rate of the component when it is active. It is a well accepted practice to approximate the failure rate of a dormant component to be zero. Therefore, we think it is more appropriate to consider a dormant component in a phase as a cold spare. Consequently, the default failure rate of a dormant component is assumed to be zero in the MT algorithm. In addition, in our approach, it is also possible to assign a non-zero failure rate to a cold spare.

Now, consider another two phased mission system as shown in Fig. 13, whose phase sequence is also XY. The only difference between the two PMS (Figs. 12 and 13) is that in Fig. 13, component *A* is being used in Phase 1 as well. However, after applying the mincut cancellation rule, Fig. 13 can be represented by Fig. 12. Although component *A* is not physically shown in Phase X in Fig. 12, when the MT algorithm uses Fig. 12 as a simplification for Fig. 13, MT

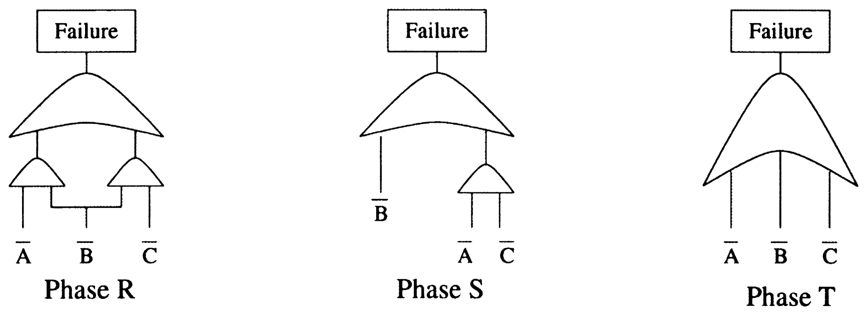

Fig. 11. A PMS with shared components in Phase R.

Table 4
Notation simplification

	Notation							
In Ref. [1]	HPCS	ADS	LPCI-A	LPCI-B	LPCI-C	LPCS	HX-A	HX-B
In Fig. 15	$\overline{A}$	$\overline{B}$	$\overline{C}$	$\overline{D}$	$\overline{E}$	$\overline{F}$	$\overline{G}$	$\overline{H}$

will automatically retrieve the failure rate of component *A* for Phase 1, instead of regarding it as 0. This is achieved by maintaining two lists for each phase in the MT algorithm. One list is to record the mincuts. The other is the component list. For both Figs. 12 and 13, the mincuts for the two PMS are the same. However, their component lists for Phase X are different. Consequently, the unreliabilities are different (please refer to Table 6) for the phased-mission systems in Figs. 12 and 13.

6.1.3. Example 4: the tutorial example in Ref. [1]

Fig. 14 shows the PMS taken from the tutorial example in Ref. [1]. For the two different sets of failure rates provided in Ref. [1], the MT algorithm obtains the same exact solutions as in Ref. [1].

6.1.4. Example 5: boiling water reactor (BWR) system

In Fig. 15, the emergency core cooling system (ECCS) of a BWR is considered [1]. A detailed physical description of this PMS can be found in Ref. [1]. The notations of Ref. [1] are simplified as in Table 4. The mincuts for Phase 3 are listed two times (before and after the mincut cancellation) in Ref. [1]. They are not consistent with each other and none of them is correct. The errors and the correction are shown in the Appendix.

As mentioned in Example 3, in Ref. [1], components *G* and *H* are considered as hot spares in Phase 1. Under this assumption, MT yields the correct result for the unreliability of the PMS, which is $5.22048061 \times 10^{-4}$. In Ref. [1], the result was calculated as 5.22046×10^{-4}, a smidgen lower than the correct answer. When components *G* and *H* are regarded as cold spares in Phase 1, the unreliability is calculated to be $5.22038735 \times 10^{-4}$ by using the MT algorithm as implemented in SHARPE.

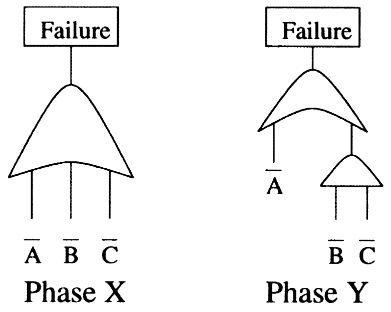
Failure
Failure
$\overline{A}$ $\overline{B}$ $\overline{C}$
Phase X
$\overline{A}$
$\overline{B}$ $\overline{C}$
Phase Y

Fig. 13. A PMS where all the components are active.

6.1.5. Example 6: a distributed computer system

In Fig. 16, we consider a modified version of a distributed computer system modeled in Ref. [9]. The system consists of four types of redundancy nodes:

- Node P has triple modular redundancy (Pa, Pb and Pc). For the PMS to function correctly, at least two of them must be working in Phase A. This is represented by the 2-*out-of*-3 gate.
- Node Q has quadruple modular redundancy (Qa, Qb, Qc and Qd). Different phase has different redundancy requirement for node Q.
- Node M has dual redundancy (Ma and Mb).
- Node R has triple modular redundancy (Ra, Rb and Rc).

The input parameters for each phase are shown in Table 5.

6.2. Execution time

In order to test the MT algorithm given in this paper, we measure the execution times for a number of phased-mission systems. Since there is no implementation for the ST algorithm, the execution times cannot be compared between ST and MT. However, by including the execution times taken by MT as implemented in SHARPE, we have provided some information for the reader to judge the efficiency of the algorithm.

Table 6 represents the experimental results by running SHARPE for a number of examples presented in this paper. The clock starts to tick when the routine *find_mincuts* is called in Fig. 5. The clock stops when the unreliability of the PMS at the end of the mission is found. The results are generated by using a SUN Ultra 1 workstation. Unless otherwise specified, the following assumptions are made:

- the permutation order of the PMS is the same as that shown in the related figure;
- the failure rate of each component is constant and the value is 0.0001 per unit time; the time duration for each phase is constant and the value is 10 time units;
- when a component is dormant in a phase, it is considered

Table 5
Input parameters for the PMS in Fig. 16

Phase	λ_P [a]	λ_Q [a]	λ_M [a]	λ_R [a]	Duration
1	110	120	14	0	2
2	–	310	5.1	0	8
3	–	220	4.1	0	9
4	–	37	22	140	12

[a] The unit of the failure rates is 10^{-6}/unit time.

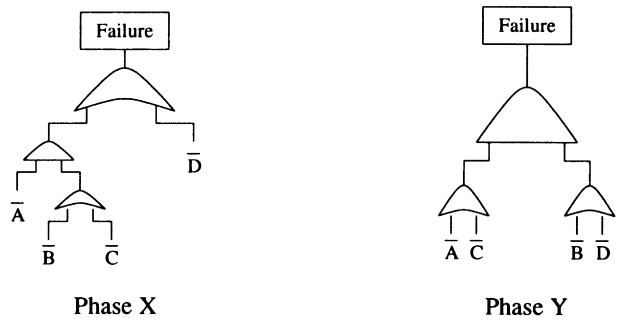

Fig. 14. The tutorial example of Ref. [1].

as a cold spare, i.e. the failure rate of the component in that phase is assumed to be zero.

7. Conclusion

In this paper, we extend the sum of disjoint products formula into the sum of disjoint phase products (SDPP) formula. An algorithm based on SDPP to analyze the unreliability of a PMS has been developed. The algorithm yields exact results and is simple in concept and computation. Both end-of-mission and transient analysis of a PMS can be carried out by the algorithm. The algorithm has also been incorporated in the software package SHARPE. With SHARPE, the complexity of the algorithm is hidden from the user. A PMS can be described at a high level with fault trees. Then by running a SHARPE input file, we can analyze quantitatively the unreliabilities of the PMS.

Acknowledgements

This work was supported in part by the Department of Defense, the IBM Corporation, the Kenan Institute and NC ACTS as enhancement projects to CACC. We are pleased to thank Dr. Arun K. Somani and Dr. Robin A. Sahner for their valuable suggestions on the draft of this paper and the SHARPE implementation of the algorithm. We would also like to thank an anonymous reviewer for his/her careful review and constructive comments.

Table 6
Solutions of the PMS examples

PMS	Total no. of mincuts		DPP	Time (μs)	Unreliability ($\times 10^{-3}$)
	Before cancellation	After cancellation			
Fig. 1	6	5	6	2305	3.99300865
Fig. 11	7	3	3	1594	8.95962123
Fig. 12	4	4	4	1866	2.99650050
Fig. 13	5	4	4	1871	3.99300567
Fig. 14[a]	7	6	8	2894	97.3969923
Fig. 15	13	7	19	8266	0.522038735
Fig. 16	29	13	102	77541	0.127960793

[a] The first set of failure rates in Ref. [1] is used here.

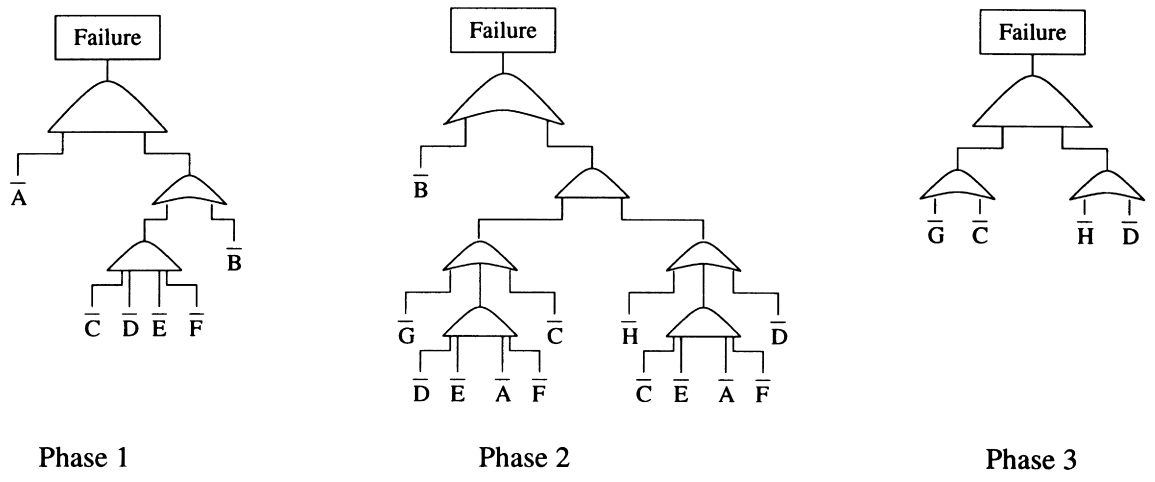

Fig. 15. Fault trees for the ECCS of a BWR.

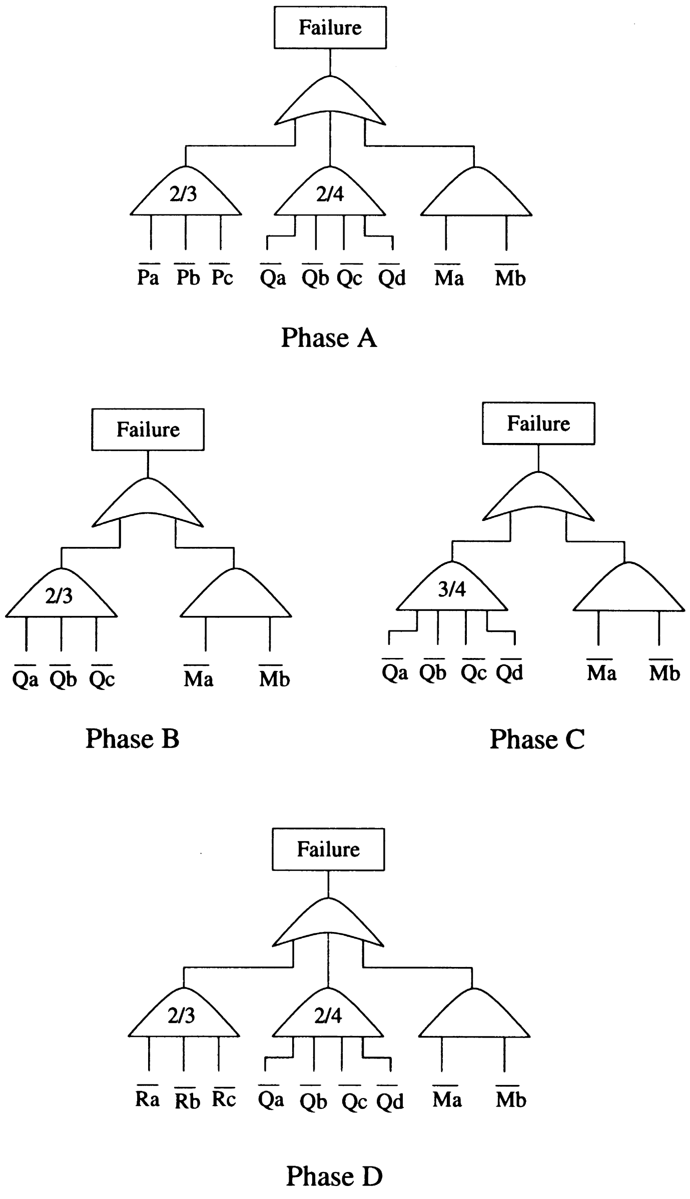

Fig. 16. Fault trees for a distributed computer system.

Appendix

The mincuts for Phase 3 in Fig. 15 are listed two times in Ref. [1] as following. None of them is correct.

1. {HX-A, HX-B, HX-A, LPCI-B}, {LPCI-A, HX-B, LPCI-A, LPCI-B}.
2. {HX-A, HX-B, LPCI-A, LPCI-B}, {LPCI-A, HX-B, LPCI-A, LPCI-B}.

It is easy to verify that the correct mincuts for Phase 3 should be:

{HX-A, HX-B}, {HX-A, LPCI-B}, {LPCI-A, HX-B}, {LPCI-A, LPCI-B}.

In the simplified notation (Table 4), the mincuts are:

$\{\overline{G},\overline{H}\},\{\overline{G},\overline{D}\},\{\overline{C},\overline{H}\},\{\overline{C},\overline{D}\}$.

References

[1] Burdick GR, Fussell JB, Rasmuson DM, Wilson JR. Phased mission analysis: a review of new developments and an application. IEEE Trans Reliability 1977;26:43–49.

[2] Alam M, Al-Saggaf U. Numerical solution of sparse singular systems of equations arising from ergodic Markov chains. IEEE Trans Reliability 1989;35:498–503.

[3] Dugan JB. Automated analysis of phased-mission reliability. IEEE Trans Reliability 1991;40:45–51.

[4] Esary JD, Ziehms H. Reliability analysis of phased missions. In: Barlow RE, Fussel JB, Singpurwalla ND, editors. Reliability and fault tree analysis: theoretical and applied aspects of system reliability and safety assessment, Philadelphia: SIAM, 1975. pp. 213–236.

[5] Mura I, Bondavalli A, Zang X, Trivedi KS. Dependability modeling and evaluation of phased mission systems: a DSPN approach. In: Proceedings of Dependable Computing for Critical Applications (DCCA), 7, 1999.

[6] Kim K, Park KS. Phased-mission system reliability under Markov environment. IEEE Trans Reliability 1994;43:301–309.

[7] Pedar A, Sarma VVS. Phased-mission analysis for evaluating the effectiveness of aerospace computing-systems. IEEE Trans Reliability 1981;30:429–437.

[8] Smotherman M, Zemoudeh K. non-homogeneous Markov model for phased-mission reliability analysis. IEEE Trans Reliability 1989;38:585–590.

[9] Somani AK, Ritcey J, Au S. Computationally efficient phased-mission reliability analysis for systems with variable configuration. IEEE Trans Reliability 1992;42:504–509.

[10] Somani AK, Palnitkar S, Sharma T. Reliability modeling of systems with latent failures using Markov chains. In: Proceedings of the RAMS. 1993, pp. 120–125.

[11] Akers B. Binary decision diagrams. IEEE Trans Computers 1978;27:509–516.

[12] Bryant RE. Graph based algorithms for Boolean function manipulation. IEEE Trans Computers 1986;35:677–691.

[13] Bryant RE. Smybolic Boolean manipulation with ordered binary decision diagrams. ACM Computing Surveys 1992;24:293–318.

[14] Zang X, Sun H, Trivedi KS. A BDD-based algorithm for reliability evaluation of phased mission system. Submitted for publication.

[15] Somani AK, Trivedi KS. Phased-mission system analysis using Boolean algebraic methods. In: Proceedings of SIGMETRICS, 1994, pp. 98–107.

[16] Sahner R, Trivedi KS. Performance and reliability analysis of computer systems: an example-based approach using SHARPE software package, Boston: Kluwer Academic, 1995.

[17] Colbourn CJ. The combinatorics of network reliability, New York: Oxford University Press, 1987.

[18] Rai S, Agrawal DP. Distributed computing network reliability, Silver Spring, MD: IEEE Computer Society Press, 1990 Tutorial text.

[19] Rai S, Veeraraghavan M, Trivedi KS. A survey on efficient computation of reliability using disjoint products approach. Networks 1995;25:147–163.

[20] Fussell JB, Vesely WE. new methodology for obtaining cut sets for fault trees. Am Nucl Soc Trans 1972;15:262–263.

[21] Bennetts RG. On the analysis of fault trees. IEEE Trans Reliability 1975;24:194–203.

[22] Rauzy A. New algorithms for fault trees analysis. Reliability Engineering and System Safety 1993;40:203–211.

[23] Abraham JA. An improved algorithm for network reliability. IEEE Trans Reliability 1979;28:58–61.

[24] Heidtmann KD. Smaller sums of disjoint products by subproduct inversion. IEEE Trans Reliability 1989;38:305–311.

[25] Soh S, Rai S. Computer aided reliability evaluator for distributed computing networks. IEEE Trans Parallel and Distributed Systems 1991;2:199–213.

[26] Veeraraghavan M, Trivedi KS. An improved algorithm for symbolic reliability analysis. IEEE Trans Reliability 1991;40: 347–358.

[27] Soh S, Rai S. Experimental results on preprocessing of path/cut terms in sum of disjoint products technique. IEEE Trans Reliability 1993;42:24–33.

[28] Luo T, Trivedi KS. An improved algorithm for coherent-system reliability. IEEE Trans Reliability 1998;47:73–78.

[29] Somani AK. Simplified phased-mission system analysis for systems with independent component repairs. Int J Reliability Quality Safety Engng 1997;4:167–191.

[30] Biggerstaff TJ, Perlis A, editors. Software reusability New York: ACM Press, 1989.

[31] Pree W. Design patterns for object-oriented software development, Reading, MA: Addison-Wesley, 1995.