Symbolic Manipulation of Structure Functions in Availability Analysis

Yoichi Watanabe

November 1985

UWFDM-658

FUSION TECHNOLOGY INSTITUTE

UNIVERSITY OF WISCONSIN

MADISON WISCONSIN

Symbolic Manipulation of Structure Functions in Availability Analysis

Yoichi Watanabe

Fusion Technology Institute
University of Wisconsin
1500 Engineering Drive
Madison, WI 53706

http://fti.neep.wisc.edu

November 1985

UWFDM-658

Symbolic Manipulation of Structure Functions in Availability Analysis

Yoichi Watanabe

Fusion Technology Institute
Nuclear Engineering Department
University of Wisconsin-Madison
1500 Johnson Drive
Madison, WI 53706

November 1985
revised in November 1986

UWFDM-658

Contents

List of Figures

List of Tables

1 Introduction

The state of a system can be represented by the states of components, of which the system is composed, and states of events outside the system that influence the system state. The function representing the relationship among the system state and states of components and external events is called the structure function and denoted by ϕ [1]. Let the state of a component or event i be denoted by X_i. Then the state of the system, Y, can be obtained by

$$Y = \phi(X_1, X_2, ..., X_n) \tag{1}$$

where n is the number of components/events. In general, Y and X_i are functions of time. Hence the time-dependent behavior of the system can be obtained once the behavior of components and the structure function are known. In the binary-state model, Y and X_i randomly take on the value 1 or 0. Here the values 1 and 0 denote a normal state and a failed state, respectively. Then, the availability of a system at time t, which is called the point availability, is identical to the probability that the random variable Y takes on the value 1 at time t.

In practical analysis, the relationship among components is displayed as a success tree, a block diagram, or a network graph. In the success tree approach, the system state, i.e. the top event, is represented by using logic gates and component states. A main task of availability analysis, therefore, is to obtain the state of the top event from the states of components at a specific time. One of methods for this analysis utilizes the concepts of cut sets and path sets [1].

An estimation of the top event can be made by directly obtaining an expression for the structure function of a success tree. This straightforward approach has been investigated by several researchers [2],[3],[4], [5],[6],[7],[8], [9],[10]. The method is truly arithmetic and requires a large amount of algebra. Because of this, the method has not been widely used by reliability engineers.

The evolution of computing technology has made symbolic manipulation on a computer much easier and faster; currently several software packages such as MACSYMA, REDUCE, and ALTRAN [11] are available for nonspecialists. In particular, the REDUCE program is readily obtained by users

at the National Magnetic Fusion Energy Computer Center (NMFECC), Livermore, California [12].

In this report, we shall demonstrate several applications of REDUCE for generation and manipulation of structure functions in reliability and availability analysis using the binary-state model. In the following section, four major usages of the structure functions will be discussed. In addition, several other possible applications will be suggested. In Section 3, a computer program REDFOR, which is an input for the REDUCE program, will be described in detail. The REDFOR program is used to perform symbolic manipulations for the applications discussed in Section 2. Outcomes of this program are Fortran programs, which are compiled and linked to create executable programs and numerical answers for specific problems. Section 4 will conclude this work.

2 Applications of Structure Functions

2.1 Reliability and Availability Computation

2.1.1 Independent Components

Suppose that a system, whose state is denoted by a random variable Y, is composed of n components with states denoted by random variables X_i,i=1,2,....,n. For convenience, introduce a vector $\mathbf{X}$, whose components are $X_1, X_2, ..., X_n$. Then the structure function, ϕ, is given by

$$Y = \phi(\mathbf{X}). \tag{2}$$

We make three assumptions for further discussion:

(a) ϕ is a polynomial function of $X_1, X_2, ..., X_n$.

(b) ϕ includes only terms with the first power of any X_i.

(c) Random variables $X_1, X_2, ..., X_n$ are independent.

Since any success tree can be built by means of AND, OR, and m-out-of-n gates and the structure functions of these gates are polynomials of input state variables, the assumption (a) is true. There may be higher order terms

of a particular state variable in ϕ if there is more than one component with the same random variable. These higher terms, however, can be reduced to first order terms by applying Boolean algebra as we shall see later. Hence the assumption (b) is true. The independence of components stated by the assumption (c) is not true in many cases. In the next section, we shall show how dependency can be incorporated in the current approach. Meanwhile, in this section we assume the independence of components.

Let the expectation of a random variable X be denoted by $E(X)$. Then under the above three assumptions, the expectation of Y is given by

$$E(Y) = \phi(E(\mathbf{X})) \tag{3}$$

where $E(\mathbf{X}) = (E(X_1), E(X_2), ..., E(X_n))$.

Since the expectation is equivalent to the point availability, Eq. (3) leads to an equation of the system availability $a(t)$ in terms of component availabilities $a_i(t)$:

$$a(t) = \phi(\mathbf{a}(t)) \tag{4}$$

where $\mathbf{a}(t) = (a_1(t), a_2(t), ..., a_n(t))$.

EXAMPLE

Consider a system represented by the success tree shown in Fig. 1, where the symbols have the standard meanings [1]. Denote states of gates A,B,C, and D by Y_A, Y_B, Y_C, and Y_D, respectively. Then we have

$$Y_A = Y_B Y_C X_7 \tag{5}$$

$$Y_B = X_1 X_2 Y_D \tag{6}$$

$$Y_C = 1 - (1 - X_3)(1 - X_4)(1 - X_1) \tag{7}$$

$$Y_D = 1 - (1 - X_5)(1 - X_6) \tag{8}$$

Substituting Y_B, Y_C,and Y_D given by Eqs. (6),(7), and (8) into Eq. (5) and manipulating it, we have

$$Y_A = X_1 X_2 (X_5 + X_6 - X_5 X_6)(X_3 + X_4 + X_1 - X_3 X_4 - X_4 X_1 - X_3 X_1 X_1 X_3 X_4) X_7 \tag{9}$$

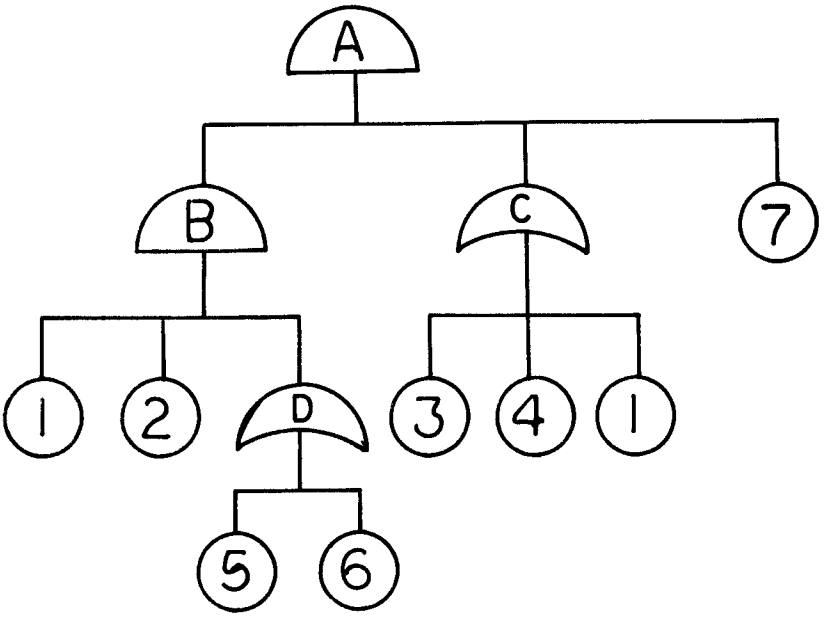

Figure 1: Success tree of the first example

This equation contains the second order terms of X_1. According to a rule of Boolean algebra [1],

$$X_1^2 = X_1. \tag{10}$$

Thus Eq. (9) becomes

$$Y_A = X_1X_2(X_5 + X_6 - X_5X_6)X_7. \tag{11}$$

If components 1,2,5,6, and 7 are independent, the availability of the system, $a(t)$, is given by

$$a(t) = a_1(t)a_2(t)(a_5(t) + a_6(t) - a_5(t)a_6(t))a_7(t). \tag{12}$$

2.1.2 Dependent Components

Suppose that m out of n components, $X_{i_1}, X_{i_2}, ..., X_{i_m}$, are dependent. By expanding the function ϕ given by Eq. (2), Y can be represented as a sum of terms such as $X_{j_1}X_{j_2}...X_{j_k}$ $(= Z)$, $k \leq m$. If $X_{j_1}, X_{j_2}, ...,$ and X_{j_k} are independent,

$$E(Z) = E(X_{j_1})E(X_{j_2})...E(X_{j_k}). \tag{13}$$

But if some of these variables, say $X_{j_1}, X_{j_2}, ..X_{j_l}$, for $l \leq k$, are dependent, $E(Z)$ should be represented as

$$E(Z) = Pr[\{X_{j_1} = 1\} \cap \{X_{j_2} = 1\} \cap ... \cap \{X_{j_l} = 1\}]E(X_{j_{l+1}})...E(X_{j_k}). \tag{14}$$

Here $Pr[\{X_{j_1} = 1\} \cap \{X_{j_2} = 1\} \cap ... \cap \{X_{j_l} = 1\}]$ is the probability that the random variables $X_{j_1}, X_{j_2}, ..,$ and X_{j_l} take on the value 1 simultaneously.

In this section we consider four types of component dependency:
(i) warm or cold standby redundancy, (ii) failure induced by another failure, (iii) common cause failure, and (iv) mutually exclusive events.

(i) Warm or cold standby redundancy

Warm or cold standby redundancy can be represented by using an m-out-of-n voting gate; that is, m components are in operation and n-m components are either in warm standby or cold standby. This type of redundancy is fully described in §8.2.3 of Ref. [1]. According to this reference, the steady-state probability that the output state of the gate is 1, i.e. the availability of the gate, A, is given by

$$A = \sum_{k=1}^{n-m} P_{(k)} \tag{15}$$

where

$$P_{(k)} = \frac{\theta_k}{\sum_{k=0}^{n} \theta_k} \tag{16}$$

$$\theta_0 = 1, \theta_k = \frac{\prod_{i=0}^{k-1} \lambda_i}{\prod_{i=1}^{k} \mu_i} \tag{17}$$

$$\lambda_k = \begin{cases} m\lambda + (n-m-k)\overline{\lambda} & for \quad 0 \leq k \leq n-m \\ (n-m)\lambda & for \quad n-m+1 \leq k \leq n-1 \end{cases} \tag{18}$$

$$\mu_k = min\{r, k\}\mu \quad for \quad 1 \leq k \leq n. \tag{19}$$

In Eqs. (18) and (19), λ and μ are the failure rate and repair rate of a component. $\overline{\lambda}$ is the failure rate of a standby component. It is assumed that at most r components can be repaired at a time.

In order to obtain the steady-state availability of a system having warm or cold standby components, the state of the gate for which input components are dependent must be replaced by an expression given by Eq. (15). For example, suppose that components 5 and 6 in the example given in Section 2.1.1 are dependent (warm or cold standby redundancy). For independent components,

$$E(Y_D) = a_5 + a_6 - a_5 a_6 = \frac{\mu(\mu + 2\lambda)}{(\lambda + \mu)^2}. \tag{20}$$

But for the dependent case,

$$E(Y_D) = \frac{\theta_0 + \theta_1}{\theta} = \frac{2\mu(\mu + \lambda + \overline{\lambda})}{2\mu^2 + (\lambda + \overline{\lambda})(2\mu + \lambda)} \quad . \tag{21}$$

Here r=1 is assumed.

(ii) Failure induced by another failure

Suppose that there are two components 1 and 2 with failure rates λ_1, λ_2 and repair rates μ_1, μ_2 and the failure of component 1 changes the failure rate of component 2 from λ_2 to $\hat{\lambda_2}$. The state transition diagram for this two component system is given in Fig. 2. It shows that there are four states. Let the probability that the system is in state i be denoted by P_i. Since the process is Markovian, the following differential equations can be derived:

$$\frac{dP_1}{dt} = -(\lambda_1 + \lambda_2)P_1 + \mu_1 P_2 + \mu_2 P_3 \tag{22}$$

$$\frac{dP_2}{dt} = \lambda_1 P_1 - (\hat{\lambda_2} + \mu_1)P_2 + \mu_2 P_4 \tag{23}$$

$$\frac{dP_3}{dt} = \lambda_2 P_1 - (\lambda_1 + \mu_2)P_3 + \mu_1 P_4 \tag{24}$$

$$\frac{dP_4}{dt} = \hat{\lambda_2} P_2 + \lambda_1 P_3 - (\mu_1 + \mu_2)P_4 \quad . \tag{25}$$

We also have

$$P_1 + P_2 + P_3 + P_4 = 1 \quad . \tag{26}$$

In the steady-state case, the solution is easily obtained. For example P_1 is given by

$$\begin{aligned} &P_1 = Pr[\{X_1 = 1\} \cap \{X_2 = 2\}] \\ &= (\mu_1\mu_2(\lambda_1 + \hat{\lambda_2} + \mu_1 + \mu_2))/((\lambda_1^2\mu_2 + \lambda_1^2\hat{\lambda_2} + \lambda_1\lambda_2\mu_1 + \lambda_1\lambda_2\mu_2 + \\ &\lambda_1\lambda_2\hat{\lambda_2} + 2\lambda_1\mu_1\mu_2 + \lambda_1\mu_1\hat{\lambda_2} + \lambda_1\mu_2^2 + \lambda_1\mu_2\hat{\lambda_2} + \lambda_2\mu_1^2 \\ &+\lambda_2\mu_1\mu_2 + \lambda_2\mu_1\hat{\lambda_2} + +\mu_1^2\mu_2 + \mu_1\mu_2^2 + \lambda_1\lambda_2\hat{\lambda_2}) \quad . \end{aligned} \tag{27}$$

When the failure of a component influences failure rates of more than one component, the similar argument can be applied. But the arithmetic involved becomes very cumbersome.

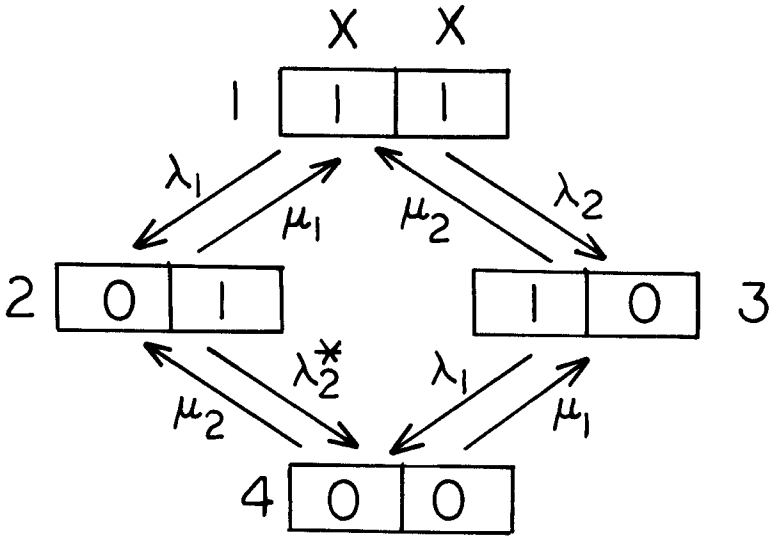

Figure 2: State transition diagram

(iii) Common cause failure

Consider a common cause failure of ℓ components. Let the failure rate due to the common cause be c. Then the probability that all components are in state 1 at time t is given by

$$Pr[\{X_1=1\}\cap\{X_2=1\}\cap....\cap\{X_\ell=1\}] \\ =e^{-ct}\prod_{i=1}^{\ell}\left[\frac{\mu_i}{\lambda_i+\mu_i}-\frac{\lambda_i}{\lambda_i+\mu_i}e^{-(\lambda_i+\mu_i)t}\right]+\int_0^t dsce^{-cs}\prod_{i=1}^{\ell}\left[\frac{\mu_i}{\lambda_i+\mu_i}(1-e^{-(\lambda_i+\mu_i)s})\right] \tag{28}$$

This equation can be derived by using the arguments given in §8.3 of Ref. [1].

For the steady state case, i.e. t $\rightarrow \infty$, Eq. (28) becomes

$$Pr[\{X_1=1\}\cap\{X_2=1\}\cap....\cap\{X_\ell=1\}] \\ =\int_0^\infty dsce^{-cs}\prod_{i=1}^{\ell}[\frac{\mu_i}{\lambda_i+\mu_i}(1-e^{-(\lambda_i+\mu_i)s})] \tag{29}$$

In particular, for ℓ =2,

$$Pr[\{X_1=1\}\cap\{X_2=1\}] \\ =\frac{(\lambda_1+\mu_1+\lambda_2+\mu_2+2c)\mu_1\mu_2}{(\lambda_1+\mu_1+c)(\lambda_2+\mu_2+c)(\lambda_1+\mu_1+\lambda_2+\mu_2+c)} \tag{30}$$

(iv) Exclusive events

Suppose that two events i and j are mutually exclusive. Then the following relation between the random variables X_i and X_j holds:

$$X_i+X_j=1 \quad . \tag{31}$$

Hence, if there are mutually exclusive events in a success tree, we can remove one of those two random variables from the structure function by using Eq. (31).

2.2 Importance Calculation

Let us assume that n components are independent. Then we have Eq. (4):

$$a(t)=\phi(a_1(t),a_2(t),.....,a_n(t)) \quad . \tag{32}$$

We define importances for success trees similar to those of fault trees described in Chapter 10 of Ref.[1].

Birnbaum's structural importance

$$\triangle g_i(t) = \frac{\partial a(t)}{\partial a_i(t)} \tag{33}$$

Critical importance

$$I_i^{CR}(t) = \frac{a_i(t)}{a(t)} \triangle g_i(t) \tag{34}$$

Upgrading function

$$I_i^{UF}(t) = \frac{\lambda_i \partial u(t)}{u(t) \partial \lambda_i} \tag{35}$$

where $u(t) = 1 - a(t)$ and $a_i(t) = e^{-\lambda_i t}$.

Barlow-Proschan importance

$$I_i^{BP} = \int_0^t \frac{\partial u(t)}{\partial u_i(t)} w_i(t) dt \tag{36}$$

where $u(t) = \phi(u_1(t), u_2(t), ..., u_n(t))$ and $w_i(t)$ is the unconditional failure intensity. ϕ is the structure function of a fault tree and u and u_i are unavailabilities.

2.3 Error Propagation Analysis

The uncertainty due to the uncertainties of λ_i and μ_i leads to the uncertainty of a_i and consequently, the uncertainty of the system availability $a(t)$. The estimation of the uncertainty of $a(t)$ can be made by using a Monte Carlo program such as the SAMPLE program [1]. For this analysis

an explicit expression of the structure function is useful. In fact, the expression must be provided by a user to the SAMPLE program. Although the Monte Carlo approach gives the distribution function of $a(t)$, the method is rather expensive.

In this section, we obtain an estimate of the variance of $a(t)$ analytically from expectations and variances of $a_i(t)$. For simplicity, make the three assumptions given in Section 2.1.1. The structure function is now the first order polynomial with respect to any a_i. Thus we have

$$E(a) = \phi(E(\mathbf{a})) \tag{37}$$

where $E(\mathbf{a}) = (E(a_1), E(a_2), ..., E(a_n))$.

If the variance of a random variable X is denoted by $V(X)$, the variance of a is given by

$$V(a) = E(\phi^2(\mathbf{a})) - E(\phi(\mathbf{a}))^2 = E(\phi^2(\mathbf{a})) - \phi^2(E(\mathbf{a})). \tag{38}$$

The first term on the right hand side of Eq. (38) includes the expectations of the second order term of a_i. This can be eliminated by using the definition of the variance:

$$E(a_i^2) = V(a_i) \ + E^2(a_i). \tag{39}$$

Therefore, $V(a)$ is represented as a function of a_i and $V(a_i)$, i=1,2,....,n.

EXAMPLE

An error analysis is carried out for the fault tree given in Fig. 3. This problem is taken from §12.5.2 in Ref. [1], which lists the results of the Monte Carlo program SAMPLE. The structure function for the top gate is given by

$$Y = X_2 + X_1X_3 - X_1X_2X_3. \tag{40}$$

Since $X_1X_2X_3$ is small compared with other terms, we can ignore this term. The means and variances of random variables X_1, X_2, and X_3 are given in Table 1.

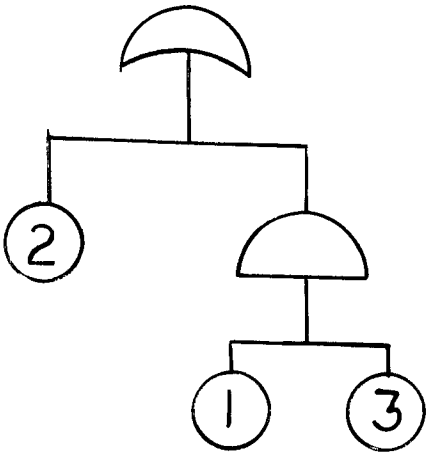

Figure 3: Fault tree for Error Propagation Analysis

Table 1: Means and variances of components

Component ID	Median Unavail.	Mean	Variance
1	9.90E-3	1.25E-2	9.36E-5
2	7.41E-2	9.37E-2	5.25E-3
3	1.53E-1	1.93E-1	2.22E-2

Table 2: Computational results

Method	Expectation	Standard deviation
SAMPLE	9.4798E-2	6.7285E-2
Analytical	9.61E-2	7.24E-2

The expectation of Y is given by

$$E(Y) = E(X_2) + E(X_1)E(X_3). \tag{41}$$

The variance of Y is given by the following equation after some manipulation using Eq. (39):

$$V(Y) = V(X_2) + V(X_1)V(X_3) + E^2(X_1)V(X_3) + V(X_1)E^2(X_3). \tag{42}$$

The expectations and standard deviations $\sqrt{V(Y)}$ by the SAMPLE program and analytical formulas Eq. (41) and (42) are shown in Table 2. These results agree well with each other.

2.4 Utilization in Monte Carlo Simulation

In Monte Carlo simulations of system availability, states of gate outputs are computed by using input states. The procedure is repeated for all gates from the gates at the lowest level up to the top gate [13],[14]. Since this algorithm is serial in nature and involves bookkeeping processes such

as obtaining the number of inputs, ID numbers of input components and gates, and a proper logic for the gate, it is not sufficiently efficient even on supercomputers utilizing vector and parallel processing.

Here we propose a very different approach: a use of structure functions. The analytical expressions of structure functions are obtained for every gate in terms of only component states. Hence the gate states can be computed simultaneously by just calling a subroutine including those expressions once the states of all components are known. It is noteworthy, however, that the vectorization of this algorithm cannot be realized by the present Cray compiler because each gate is associated with a different form of function, while, the multiprocessing can be easily applied.

In order to demonstrate this new approach, the PROPA program (Version 1.5) has been modified so that it does binary-state simulations. A program POLD uses the old algorithm; a program PNEW utilizes the structure function approach. The CPU times used by the portion where gate states are computed are compared for the following three problems. All cases are solved by using 100 histories for 8760 hours of operation time with 24 hours for time step.

- The first problem is from Ref. [15]. It has 2 AND gates and 3 components.
- The second problem is taken from Ref. [16]. It has 6 AND gates and 14 components.
- The third problem is the same as the second one except component 6 is replaced by a 2-out-of-3 gate. It has now 7 gates and 16 components.

The results are shown in Table 3. This indicates the new approach is about twice as fast as the old one. Although obtaining the structure functions takes some time, the new method is attractive, in particular, when a very large number of histories are executed by varying reliability parameters.

Table 3: Comparison of CPU times

Problem	POLD	PNEW
1	1,131,472	580,426
2	2,827,137	1,173,933
3	2,823,817	1,333,018

1) Unit is μsec.
2) All cases were run on a Cray-1 at NMFECC

2.5 Other Possible Applications

- Simplification of a tree

  Rearrangement of a structure function by using Boolean laws leads to a structure function corresponding to a simpler tree. For example, let us return to the example given in Section 2.1.1. Here Eq. (11) suggests a success tree illustrated in Fig. 4.

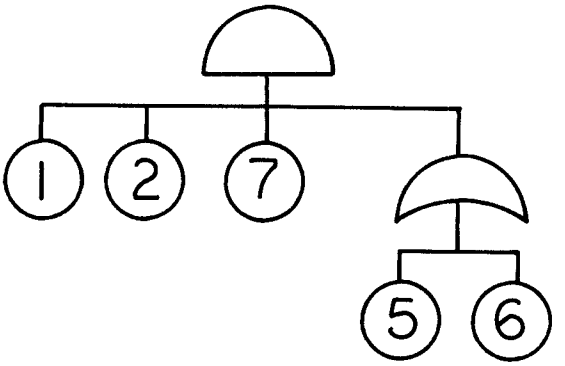

Figure 4: Simplified success tree

- Checking the correctness of minimal cut sets

  Suppose that we have obtained minimal cut sets but we are not sure if they are correct. To prove this, we can use the structure function. Let a minimal cut set i be denoted by Z_i. Then the top event Y is given by

  $$Y = 1 - \prod_i (1 - Z_i) \tag{43}$$

  where the product with respect to i is taken for all minimal cut sets. Z_i is given by a product of event states included in the cut set i. Thus if the difference between Y given by Eq. (43) and the structure function obtained by a direct method is 0, then we can say that the minimal cut sets are correct.

- Optimization

  Consider an optimization problem:
  Maximize the system availability $a = \phi(\mathbf{a})$
  subject to constraints $G_j(\mathbf{a}) \leq C$.

  Knowing the expression of ϕ in terms a_i, we easily take derivatives of ϕ with respect to any a_i. Hence, standard mathematical methods can be applied. See References [17],[18] about this subject.

3 Computer Program REDFOR

3.1 Program Description

In the previous sections, several applications of structure functions were discussed. Those applications require a large amount of symbolic manipulation. And what we need to know finally are not symbolic expressions but numerical values for a problem being considered. The computer software REDUCE can be used to manipulate complex algebraic expressions and write Fortran programs which eventually give a user numerical answers. For this purpose, a program REDFOR has been developed.

The program REDFOR is an input file for REDUCE and contains data necessary to construct the structure function of a tree. Structure functions are obtained for each gate. In the current version, it is assumed that a

gate is one of three types: AND, OR, and p-out-of-q gates and components are independent. If the output state and states of n inputs to a gate are denoted by Y and X_i,(i=1,2,....,n), the expressions of Y for the three gates can be represented as follows:

AND gate

$$Y = \prod_{i=1}^{n} X_i \tag{44}$$

OR gate

$$Y = 1 - \prod_{i=1}^{n} (1 - X_i) \tag{45}$$

p-out-of-q voting gate

$$Y = 1 - \prod^{\ell} (1 - X_{i_1} X_{i_2} ... X_{i_p}) \tag{46}$$

where $i_1, i_2, ..., i_p$ is a set of p inputs out of q inputs. The product over ℓ (=q!/p!(q-p)!) combinations must be taken. In REDFOR , the algorithms for Eqs. (44) and (45) are simple; but the derivation of Eq. (46) requires a special algorithm[19]. The product term is expanded and a Boolean law $X^n = X$ is used to simplify the expression. If all i_p inputs are identical, Eq. (46) is equivalent to

$$Y = \sum_{k=p}^{q} \binom{q}{k} X^k (1 - X)^{q-k} \tag{47}$$

Once structure functions for all gates are generated, the expressions are written in a file FUNCT so that they can be used in another run of REDUCE to manipulate them further.

Next, Fortran programs are written in a file FNCT according to an input parameter IOPT. There are six options:

IOPT=1: A Fortran subprogram FNCT containing expressions of structure functions of gates is written.

IOPT=21: A Fortran subprogram FNCT containing the expression of the structure function of the top gate and a Fortran subprogram DERV containing expressions of derivatives of the structure function of the top gate with respect to component state variables are written.

IOPT=22: A Fortran subprogram FNCT containing the expressions of all gates and a Fortran subprogram DERV containing the expressions of derivatives of the structure function of the top gate with respect to component state variables are written.

IOPT=23: Same as IOPT=22.

IOPT=3: A Fortran subprogram VAR containing the expressions of the expectation and variance of the top event is written. The expressions are derived by the method described in Section 2.3.

Finally, the Fortran subprograms obtained by running REDUCE are used for quantitative availability analysis.

- In order to utilize the structure functions in a Monte Carlo simulation as discussed in Section 2.4, the subprogram FNCT obtained by the IOPT=1 option is combined with other subprograms to create an executable program.

- In order to obtain system reliability at a specific time, the subprograms FNCT and DERV are combined with an existing main program OPT21 to create an executable program. The program computes the system unavailability and the Birnbaum and critical importances at the specific time.

- If the availabilities of components are known, availabilities of all gates and the Birnbaum and critical importances can be computed by a Fortran program, which is created by combining the subprograms FNCT and DERV obtained by the IOPT=22 option with an existing main program OPT22.

- When only failure rates and repair rates of components are known, the steady-state availabilities of gates and the Birnbaum and critical

importances can be computed by the program that is obtained from an existing main program OPT23 and subprograms FNCT and DERV created by the IOPT=23 option.

- If the expectations and variances of component availabilities are known, the expectation and variance of the top event can be computed by the program that is obtained by combining an existing program OPT3 and the subprogram VAR.

The computational flow discussed in this section is illustrated in Fig. 5. For users who are concerned about the computing cost by REDUCE, the following data is useful: the elapsed CPU time on a Cray-1(c) is 1383 msec for a 14 components and 6 gates problem with the IOPT=22 option.

3.2 Input Data for REDFOR

Parameter	Format	Number of data	Description
IOPT	I	1	Option number: 0/1/21/22/23/3
MM	I	1	Number of gates
NN	I	1	Number of components
ITP(m)	I	MM	Type of gate 1/2/3/4=AND/OR/p-out-of-q /p-out-of-q identical
INNO(m)	I	MM	Number of inputs to gate
VOTE(m)	I	-	p for p-out-of-q gate
ID(n,m)	I	ℓ * MM	ID number of inputs to gate; a positive number for component a negative number for gate

A listing of the input for the example in Section 2.1.1 is given in Fig. 6.

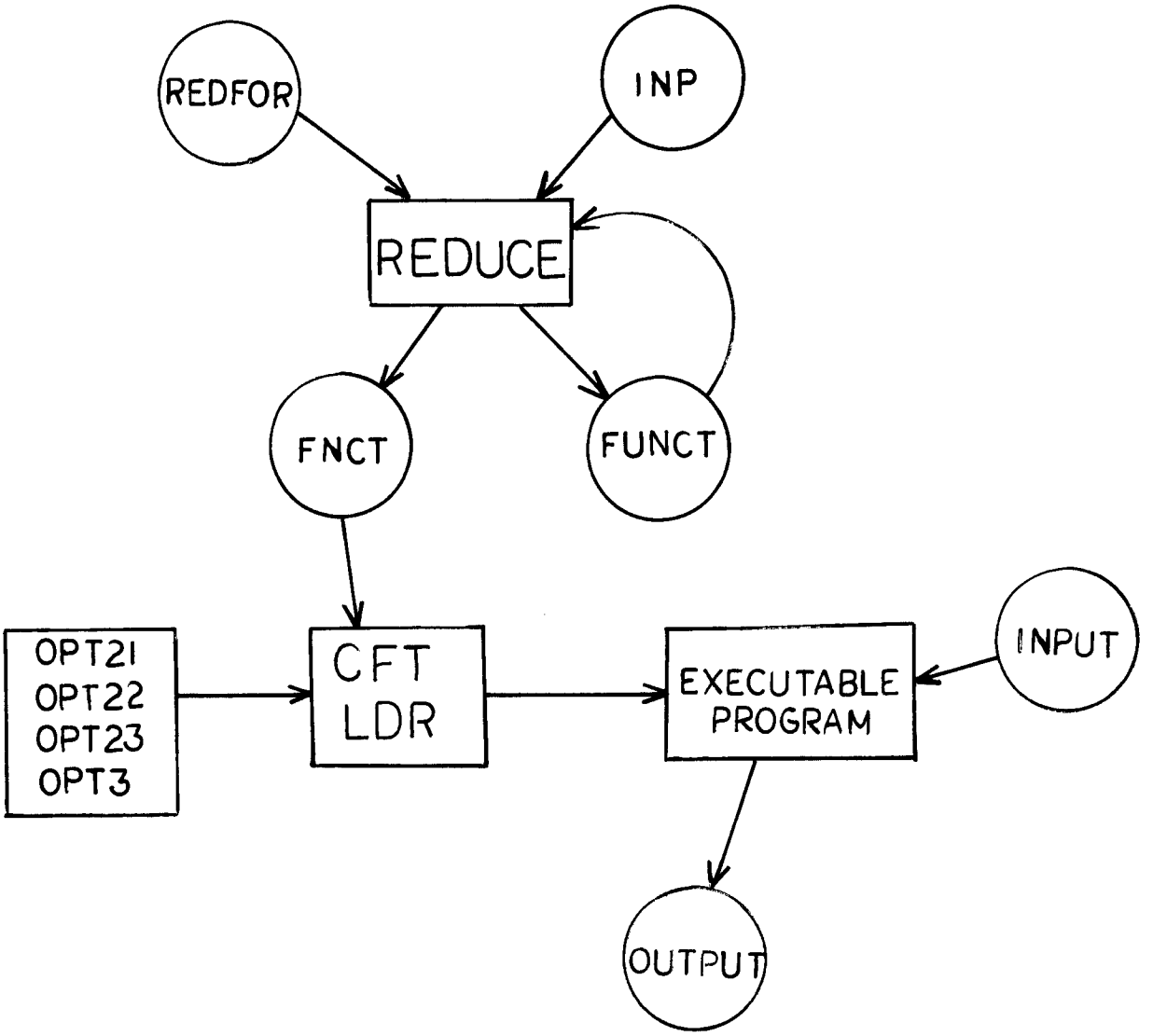

Figure 5: Computational flow by REDFOR

Figure 6: A sample input

```
%This is an input for the example problem
off echo$
iopt:=23$
mm:=4$
nn:=7$
itp(1):=1$
itp(2):=1$
itp(3):=1$
itp(4):=2$
inno(1):=3$
inno(2):=3$
inno(3):=3$
inno(4):=2$
id(1,1):=-2$ id(2,1):=-3$ id(3,1):=7$
id(1,2):=1$ id(2,2):=2$ id(3,2):=-4$
id(1,3):=3$ id(2,3):=4$ id(3,3):=1$
id(1,4):=5$ id(2,4):=6$
end$
```

3.3 Computational Procedures on NMFECC Crays

1. Obtain a LIB library BIGAPPLE by typing in
   FILEM 1751 .AVAILPHI BIGAPPLE / t v

2. Obtain the REDFOR program and a necessary Fortran program by typing in
   LIB BIGAPPLE / t v
   ok: REDFOR
   ok: the name of a file(OPT21,OPT22,OPT23,or OPT3)
   ok: END

3. Prepare an input data file INP for REDUCE, then run REDUCE by typing in
   REDUCE / t v
   1: IN "REDFOR";

4. Create a program FORT by combining the FNCT file and one of the OPT files.

5. To obtain an executable file XFORT, type in
   RCFT I=FORT,X=XFORT / t v

6. Prepare an input file, I , for XFORT.

7. Run XFORT by typing in
   XFORT I="input file name",O="output file name" / t v
   The file O contains the computational results.

To make the run more compact, a batch run file for the COSMOS routine was written. The file is named COSRED and obtained by typing FILEM 1751 .AVAILPHI COSRED. To use this, type in
COSMOS I=COSRED WITH IOPT= an option number / t v
It is noted that old files FUNCT and FNCT must be destroyed or renamed before new job is started.

4 Summary and Suggestions

Several applications of structure functions of success trees have been proposed. To quickly perform the analysis, a program REDFOR for the symbolic manipulation software REDUCE has been developed. The execution of REDUCE with REDFOR as an input creates several types of Fortran subprograms, and these subprograms can be used to quickly obtain numerical values of system availability parameters such as unavailability, availability, importances, and variance. One of subprograms also can be used in a Monte Carlo simulation program to reduce the computing time.

The current version of REDFOR derives only two types of importances. Other importances such as the upgrading function and Barlow-Proschan importances can be included. The upgrading function requires the derivative with respect to λ_i. The Barlow-Proschan importance requires an integration with respect to a time variable. These can be performed by REDUCE without any difficulty. In the current version of REDFOR, components/events must be independent. Dependency can be included in a system model by using the expressions given in Section 2.1.2.

To the author's knowledge, there is only one published work on an application of symbolic manipulation computer software to reliability and availability analysis[20]. One example of applications besides those discussed in the present report is to solve the linear differential equations and a set of algebraic equations for Markov models. In fact, Eqs. (22) to (26) in this report are solved by using REDUCE for the steady-state case, in which the time-derivatives are zero. The resulting solution, Eq. (27), is very complicated; it would be very cumbersome to obtain the solution by hand.

Acknowledgement

This work was supported by the U.S. Department of Energy.

References

[1] E.J. Henley and H. Kumamoto, "Reliability Engineering and Risk Assessment," Prentice-Hall, Englewood Cliffs, N.J. (1981).

[2] L. Fratta and U.G. Montanari, "A Boolean algebra method for computing the terminal reliability in a communication network," IEEE Trans. Circuit Theory, CT-20, 203-211 (1973).

[3] E. Hansler, G.K. McAulife, and R.S. Wilkov, "Exact calculation of computer network reliability," Networks, 4, 95-112 (1974).

[4] T.C. Huang, B.J. Leon, and P.M. Lin, "Evaluation of mutually exclusiveness for network reliability analysis," Proc. 1974 Int'l. Symp. on Circuit and Systems, pp. 176-180, San Francisco, CA (April 1974).

[5] R.G. Bennetts, "On the analysis of fault trees," IEEE Trans. Reliability, R-24, 175-185 (1975).

[6] P.M. Lin, B.J. Leon, and T.C. Huang, "A new algorithm for symbolic reliability," IEEE Trans. Reliability, R-25, 2-15(1976).

[7] W.G. Schneeweiss, "Calculating the probability of Boolean expression being 1," IEEE Trans. Reliability, R-26, 16-22 (1977).

[8] H. Gupta and J. Sharma, "A method of symbolic steady-state availability evaluation of k-out-of-n:G system," IEEE Trans. Reliability, R-28, 56-57 (1979).

[9] J.A. Abraham, "An improved algorithm for network reliability," IEEE Trans. Reliability, R-28, 58-61 (1979).

[10] T.B. Boffey and R.J.M. Walters, "Calculation of system reliability by algebraic manipulation of probability expression," IEEE Trans. Reliability, R-28, 358-363 (1979).

[11] B. Buchberger, G.E. Collins, and R. Loos, eds., "Computer Algebra: Symbolic and Algebraic Computation," 2nd ed., Springer-Verlag, Wien, New York (1983).

[12] A. Hearn, "REDUCE – A Program for Algebraic Computations," Computer Documentation, National Magnetic Fusion Energy Computer Center, Livermore, CA (1985).

[13] Z. Musicki and C.W. Maynard, "AVSYS: A computer program for fusion systems availability calculations," University of Wisconsin, Fusion Technology Institute report UWFDM-531 (1984).

[14] Y. Watanabe, "PROPA:Probabilistic performance analysis program," University of Wisconsin, Fusion Technology Institute report UWFDM-657 (1985).

[15] Y. Watanabe, "A Monte Carlo simulation method for systems with a degraded state," University of Wisconsin, Fusion Technology Institute report UWFDM-644 (1986).

[16] Y. Watanabe, "Availability analysis of fusion reactor plants with degraded states," University of Wisconsin, Fusion Technology Institute report UWFDM-646 (1985).

[17] F.A. Tillman, C-L. Hwang, and W. Kuo, "Optimization of System Reliability," Marcel Dekker, INC, New York and Basel (1980).

[18] E.J. Henley and H. Kumamoto, "Designing For Reliability and Safety Control," Chapter 9, Prentice-Hall, Englewood Cliffs, (1985).

[19] D.J. Chase, "Combinations of M out N objects," Comm. ACM, 13, 368 (1970).

[20] R. Chattergy, "Reliability analysis of on-line computer systems using computer algebraic manipulations," University of Hawaii, Report A76-1 (1976).

Appendix A: REDFOR Program Listing

```
 1  comment  REDFOR Version 1.1   created on 06/21/86;
 2  comment  REDFOR is an input file for a symbolic manipulation
 3           computer software REDUCE and creates FORTRAN programs
 4           for system availability and reliability analyses.
 5           Input file = INP, Output files = FUNCT and FNCT;
 6  off echo;
 7  operator x,y,r;
 8  array inno(50),itp(50),vote(50),id(20,50);
 9  array a(100),ind(100);
10  % read data
11  %  The top gate must be m=1
12  in "inp";
13  % compute the structure function
14  off exp;
15   for m:=1 step 1 until mm do begin
16   if itp(m)=1 then go to aaa
17   else if itp(m)=2 then go to bbb
18   else if itp(m)=3 then go to ccc
19   else go to ddd;
20  % AND gate
21  aaa:
22  g:=1;j:=1;
23  while num(inno(m)-j) >= 0 do begin
24    jj:=id(j,m);
25    if jj > 0 then xx:=x(jj)
26    else xx:=y(-jj);
27    g:=g*xx;
28    j:=j+1;
29    end;
30  y(m):=g;
31  return;
32  % OR gate
33  bbb:
34  g:=1;j:=1;
35  while num(inno(m)-j) >= 0 do begin
36    jj:=id(j,m);
37    if jj > 0 then xx:=x(jj)
38    else xx:=y(-jj);
39    g:=g*(1-xx);
40    j:=j+1;
41    end;
```

```
42  y(m):=1-g;
43  return;
44  % p-out-of-q voting gate
45  ccc:
46  p:=vote(m);
47  q:=inno(m);
48  q1:=q+1;qp:=q-p;qp1:=q-p+1;
49  r(0):=q1;r(q1):=-2;
50  for l:=1:qp do <<r(l):=0>>;
51  for l:=qp1:q do <<r(l):=l-qp>>;
52  for l:=1:q do <<a(l):=l>>;
53  done:=0;
54   z:=1;
55   for l:=qp1:q do z:=z*x(l);
56   zz:=1-z;
57  begin integer ll;
58  LNEXT:
59  %-----------------------------------------------
60  comment TWIDDLE can be used in generating all
61          combinations of p out of q objects.
62          Ref: P.J.Chase,"Combinations of M out N objects",
63               Com. ACM, Vol.13, p368 (1970);
64  begin integer j,k,l;k:=0;
65  L1:
66   k:=k+1; if r(k) <= 0 then go to L1;
67   k1:=k-1;
68   if r(k1) neq 0 then go to L11;
69   for j:=k1 step -1 until 2 do r(j):=-1;r(k):=0;
70    r(1):=1;go to L4;
71  L11:
72   if k > 1 then r(k1) :=0;
73  L2:
74   k:=k+1; if r(k) > 0 then go to L2;
75   j:=l:=k-1;
76  L3:
77   j:=j+1;
78   if r(j)=0 then << r(j):=-1; go to L3>>;
79   if r(j)=-1 then <<r(j):=r(l);r(l):=-1;go to L4>>;
80  if j=r(0) then << done:=1;go to L4>>;
81  r(k):=r(j);r(j):=0;
82  L4:
83  end;
84  %-----------------------------------------------
85   if done=1 then return;
```

```
 86    ll:=0;
 87    for l:=1:q do begin
 88     if r(l)>=1 then << ll:=ll+1;ind(ll):=a(l)>>;
 89    end;
 90    z:=1;
 91    for l:=1:p do
 92     <<lid:=ind(l);z:=z*x(lid)>>;
 93    zz:=zz*(1-z);
 94    go to LNEXT;
 95  end;
 96  for j:=1:nn do
 97   << for all l let x(j)**l = x(j) >>;
 98  y(m):=1-zz;
 99  for j:=1:nn do
100   << for all l clear x(j)**l >>;
101  return;
102  % p-out-of-q voting gate (indentical units)
103  ddd:
104  p:=vote(m);
105  q:=inno(m);
106  j:=id(1,m);
107   zz:=0;
108   for l:=p:q do begin
109    cb:=(for k:=1:q product k)/
110        ((for k:=1:l product k)*(for k:=1:(q-l) product k));
111    zz:=zz+cb*x(j)**l*(1-x(j))**(q-l);
112   end;
113  y(m):=zz;
114  return;
115  end;
116  %process results
117  %save structure function for later use
118  off nat;out funct;
119  write "off exp$";
120  write "operator x,y$";
121  for j:=1:mm do
122   <<yy:=y(j);write " y(",j,"):=",yy>>;
123  write "end";
124  shut funct;on nat;
125  comment ***************************************;
126  comment  The following will create FORTRAN programs
127           according to an option chosed by IOPT.;
128  begin
129   if iopt=0 then go to OPT0;
```

```
130   if iopt=1 then go to OPT1;
131   if iopt=21 then go to OPT21;
132   if iopt=22 then go to OPT22;
133   if iopt=23 then go to OPT22;
134   if iopt=3 then go to OPT3;
135  % Option 1 : FORTRAN program FNCT containing
136  %     structure functions y(i) for gates is created.
137  OPT1:
138  cardno!*:=20;
139  fortwidth!*:=72;
140  on fort;
141  off echo;
142  out "fnct";
143  write "c--------------------------------";
144  write "      subroutine fnct(x,y)";
145  write "      integer x(100),y(100)";
146  for n:=1 step 1 until mm do begin
147       ddd:=y(n);
148  off period;
149  write ddd;
150  write "      y(",n,")=ans";
151  on period;
152    end;
153  write "      return";
154  write "      end";
155  shut "fnct";
156  off fort;on echo;
157  out t;
158  return;
159  % Option 21: writes a FORTRAN program computing
160  %      unavailabilities and importances at time t.
161  OPT21:
162  cardno!*:=20;
163  fortwidth!*:=72;
164  on fort;
165  off echo;
166  out "fnct";
167  write "c-----------------------------------";
168  write "      subroutine fnct(x,y)";
169  write "      dimension x(100)";
170  av:=y(1);
171  write "      y=",av;
172  write "      return";
173  write "      end";
```

```
174  write "c--------------------------------";
175  write "      subroutine derv(x,dd)";
176  write "      dimension x(100),dd(100)";
177  for n:=1:nn do begin
178   ddd:=df(y(1),x(n));
179   off period;
180   write ddd;
181   write "       dd(",n,")=ans";
182   on period;
183  end;
184  write "      return";
185  write "      end";
186  shut "fnct";
187  off fort;on echo;out t;
188   return;
189  % Option 22: writes FROTRAN subprogram for steady-state
190  % availability and importances from component avails.
191  OPT22:
192  cardno!*:=20;
193  fortwidth!*:=72;
194  on fort;
195  off echo;
196  out "fnct";
197  write "c--------------------------------";
198  write "      subroutine fnct(x,y)";
199  write "      real x(100),y(100)";
200  for n:=1 step 1 until mm do begin
201       ddd:=y(n);
202  off period;
203  write ddd;
204  write "      y(",n,")=ans";
205  on period;
206    end;
207  write "      return";
208  write "      end";
209  write "c--------------------------------";
210  write "      subroutine derv(x,dd)";
211  write "      dimension x(100),dd(100)";
212  for n:=1:nn do begin
213   ddd:=df(y(1),x(n));
214   off period;
215   write ddd;
216   write "       dd(",n,")=ans";
217   on period;
```

```
218  end;
219  write "      return";
220  write "      end";
221  shut "fnct";
222  off fort;on echo;out t;
223  return;
224  % Option 23: writes a FORTRAN program for steady-state
225  % availability and importances from failure and
226  %            repair rates of components.
227  OPT23:
228   % The same subprograms as option 22 are used
229   return;
230  % Option 3: writes a FORTRAN prog. for the variance of
231  % system avail from given compts expects and variances.
232  OPT3:
233  off echo;
234  off exp;
235  operator u,v;
236  for all n let x(n)**2=v(n)+u(n)**2;
237  vv:=y(1)**2;
238  for all n clear x(n)**2;
239  for all n let x(n)=u(n);
240  ee:=y(1)**2;
241  aa:=y(1);
242  cardno!*:=20;
243  fortwidth!*:=72;
244  on fort;
245  out "fnct";
246  write "c--------------------------------------";
247  write "      subroutine var(u,v,aa,vvv)";
248  write "      dimension u(100),v(100)";
249  write "      vv=",vv;
250  write "      ee=",ee;
251  write "      vvv=vv-ee";
252  write "      aa=",aa;
253  write "      return";
254  write "      end";
255  shut "fnct";
256  off fort;on echo;out t;
257   return;
258  % Option 0:
259  OPT0:
260    % This option does nothing.
261   return;
```

```
262  end;
263  % time used in milisecond
264  showtime;
265  bye;
```

Appendix B: COSMOS run file COSRED

```
 1  */ cosmos input file "cosred"
 2  */ for reduce run by using redfor
 3  */ needs a library "bigapple"
 4  */ this file is in FILEM 1751 .AVAILPHI
 5  */to run, type COSMOS INPUT=COSRED WITH IOPT= num
 6  *select printlog=cosout
 7  *filem rds .availphi bigapple
 8  end
 9  *lib bigapple
10  x redfor
11  end
12  *destroy funct fnct
13  *reduce
14  in "redfor"$
15  *if iopt .eq. 21 then goto l21
16  *if iopt .eq. 22 then goto l22
17  *if iopt .eq. 23 then goto l23
18  *if iopt .eq. 3 then goto l30
19  *l21:
20  *lib bigapple
21  x opt21
22  end
23  *tedi opt21
24  cfa1,*,29;fnct
25  wr;fort21
26  end
27  *rcft i=fort21,x=xf21
28  *go to lend
29  *l22:
30  *lib bigapple
31  x opt22
32  end
33  *tedi opt22
```

```
34  cfa1,*,33;fnct
35  wr;fort22
36  end
37  *rcft i=fort22,x=xf22
38  *go to lend
39  *l23:
40  *lib bigapple
41  x opt23
42  end
43  *tedi opt23
44  cfa1,*,42;fnct
45  wr;fort23
46  end
47  *rcft i=fort23,x=xf23
48  *go to lend
49  *l30:
50  *lib bigapple
51  x opt3
52  end
53  *tedi opt3
54  cfa1,*,23;fnct
55  wr;fort3
56  end
57  *rcft i=fort3,x=xf3
58  *lend:
59  */ end of cosmos run
60  */ now run the xfort program
```

Appendix C-1: Program OPT21

```
 1  c23456
 2  c     RELIAN
 3  c     compute time dependent unavailablity of unrepairable system
 4        real lambda(100),q(100),brn(100),cir(100),upg(100)
 5        namelist/inp/ncomp,time
 6  c
 7        call link("unit5=(i,open),unit6=(o,create,text)//")
 8        read(5,inp)
 9        do 10 i=1,ncomp
10        read(5,*) lambda(i)
11        q(i)=1-exp(-lambda(i)*time)
12   10   continue
13  c
14        call fnct(q,unav)
15        call derv(q,brn)
16  c
17        write(6,911) time,unav
18        write(6,920)
19        do 20 i=1,ncomp
20        cir(i)=brn(i)*q(i)/unav
21        write(6,921) i,lambda(i),q(i),brn(i),cir(i)
22   20   continue
23        call exit
24   911  format(//"** Program RELIAN **",//,
25       15x,"system unavailablity at time ",f10.4," = ",e12.4//)
26   920  format(" id",2x,"lambda",6x,"unavail",5x,"BIRNBAUM",4x,
27       1"Critical")
28   921  format(i3,2x,e12.4,f10.6,2e12.4)
29        end
```

Appendix C-2: Program OPT22

```
1  c**************************************
2  c  STRCT: computes the system availability
3  c         by using the structure function
4  c--------------------------------------
5  c23456
6        dimension x(100),y(100),cri(100),dd(100)
7        namelist/par/ncomp
8  c
9        call link("unit5=(i,open),unit6=(o,create,text)//")
10 c
11       read(5,par)
12 c
13       do 10 i=1,ncomp
14        read(5,*) x(i)
15  10   continue
16 c
17       call fnct(x,y)
18       call derv(x,dd )
19       do 40 i=1,ncomp
20  40   cri(i)=dd(i)*x(i)/y(1)
21 c
22       write(6,901)
23       do 50 i=1,ncomp
24       write(6,902) i,x(i),dd(i),cri(i)
25    50 continue
26       write(6,905) y(1)
27 c
28       call exit
29 901   format(10h comp. no.,2x,"availability",4x,"BIRNBAUM",
30      14x,"Critical")
31 902   format(2x,i5,4x,3f12.6)
32 905   format(//" system availability = ",f10.4//)
33       end
```

Appendix C-3: Program OPT23

```
c  STAVAL:
c  computes steady state availabilities of components and gates
c  and BIRNBAUM and Critical importances
c  Failure and repair rates of components are input data.
c23456
      real lambda(100),mu(100),a(100),y(100),dd(100),cri(100)
      namelist/inp/ncomp,ngate
      call link("unit5=(i,open),unit6=(o,create,text)//")
c
      read(5,inp)
      write(6,900)
      do 10 i=1,ncomp
 10   read(5,*) lambda(i),mu(i)
c
c     compute steady state availability
      write(6,910)
      do 20 i=1,ncomp
      a(i)=mu(i)/(lambda(i)+mu(i))
      write(6,911) i,lambda(i),mu(i),a(i)
 20   continue
c
      call fnct(a,y)
      call derv(a,dd)
c
      do 30 i=1,ncomp
 30   cri(i)=dd(i)*a(i)/y(1)
c
      write(6,920)
      write(6,921)(m,y(m),m=1,ngate)
      write(6,950)
      do 40 i=1,ncomp
 40   write(6,951)i,dd(i),cri(i)
      call exit
 900  format(//"*** program STAVAL ****")
 910  format(//"* component data and availability",
     1//," i",3x,"  lambda",4x,"mu",10x,"availability")
 911  format(i3,2x,2e12.4,f10.6)
 920  format(//"* gate availability",//," m",3x,"availability")
 921  format(i3,2x,f10.6)
 950  format(//"id",3x," BIRNBAUM",4x,"Critical")
 951  format(i3,2x,2e12.4)
      end
```

Appendix C-4: Program OPT3

```
 1  c   program ERRAN
 2  c   computes expectaion and variance from those of components
 3        dimension a(100),v(100)
 4        namelist/inp/ncomp
 5        call link("unit5=(i,open),unit6=(o,create,text)//")
 6        read(5,inp)
 7        write(6,900)ncomp
 8        do 10 i=1,ncomp
 9        read(5,*) a(i),v(i)
10        write(6,910) i,a(i),v(i)
11   10   continue
12  c
13        call var(a,v,aa,vv)
14  c
15        write(6,950) aa,vv
16        call exit
17   900  format(//"**  program ERRAN **",//,
18       1" ncomp =",i5,//,
19       2" id",2x,"expectaion",2x,"variance")
20   910  format(i3,2x,f10.6,2x,e12.4)
21   950  format(//" system availability expectaion =",f10.6,/,
22       1         " system availability variance   =",e12.4)
23        end
```