012146-2-T

Computation-Based Reliability Analysis

Final Technical Report covering the period from January, 1974 to December, 1974

J. F. MEYER

January 1975

Prepared under
NASA Grant NGR 23-005-622

DEPARTMENT OF ELECTRICAL AND COMPUTER ENGINEERING
SYSTEMS ENGINEERING LABORATORY
THE UNIVERSITY OF MICHIGAN, ANN ARBOR

THE UNIVERSITY OF MICHIGAN

SYSTEMS ENGINEERING LABORATORY

Department of Electrical & Computer Engineering
College of Engineering

SEL Technical Report No. 83

COMPUTATION-BASED RELIABILITY ANALYSIS

Final Technical Report Covering the Period from
January 1, 1974 through December 31, 1974

Project Director

John F. Meyer

Prepared under

NASA Grant
NGR-23-005-622

TABLE OF CONTENTS

COMPUTATION-BASED RELIABILITY ANALYSIS

John F. Meyer

1. Introduction

Quantitative methods of analyzing system reliability have been recognized as an important need since the end of World War II. Early objects of reliability analysis were pieces of electronic communication and control equipment whose functional requirements were relatively easy to specify. Accordingly, what constituted the "success" or "failure" of such systems could be described in a straightforward manner and reliability measures such as "probability of success," "mean-time-to-failure," "availability," etc. were relatively easy to formulate.

During the last thirty years, however, the structural and functional complexity of man-made systems has increased trememdously, particularly in the computer field, and the reliability analysis task has likewise become much more complex. This is especially true in the case of fault-tolerant computing systems, where various types of structural redundancy must be accounted for in the analysis.

Beginning with the reliability analysis of fault-tolerant logic networks [1] and relay networks [2], a considerable amount of effort has been devoted to developing analytic methods for assessing the reliability of computing systems. Recent contributions in this regard include the analysis of fault-tolerant systems based on architecture-level descriptions of their structure [3]-[6]. In general, these methods are based on formal models which, at some desired level of abstraction, represent the structure of the systems to be analyzed. Given a particular class of models, the ability to "rely on" a system is then quantified via one or more "reliability measures" (defined on the model class). In defining such measures, what it means to "rely on" a system is usually expressed in terms of some underlying concept of system "success" or "failure." Indeed, the measure that is commonly referred to as "reliability" can be generally defined as "the probability of system success in its use environment" (see [7], for example). Thus, it is the meaning of success (or failure) that gives meaning to a reliability measure and, in turn, to any analysis that uses the measure.

In the discussion that follows, we wish to focus on the last of these issues, namely the concept of "success" as it pertains to the reliability analysis of computing systems. In particular, we contend that success criteria should be "computation-based" so that they can adequately reflect the computational needs of the user. This is in

contrast to "structure-based" success criteria which can specify what is required of a system's structure, but can specify what constitutes successful behavior only as it depends on the success of the structure. In relatively simple computing systems where computational integrity is closely related to structural integrity (e.g., as the success of addition relates to the structural integrity of an adder), such structure-based success criteria may indeed suffice. On the other hand, if the success of a computation depends not only on structure but also on such things as the state of the system prior to initiation of the computation, the time of initiation, and the input data, then structure-based criteria cannot express variations in success that result from such dependencies. Consequently, reliability measures that utilize structure-based criteria may not be indicative of a system's ability to successfully perform computations.

The purpose of the discussion that follows is to formally establish the point we have just made. We begin with an example that is intended to illustrate the need for computation-based reliability analysis, that is, analysis that utilizes computation-based success criteria. We then develop a formal model of a computer that is just complex enough to admit to the formulation of computation-based criteria and, in turn, reliability measures that utilize these criteria. Finally, we apply the formal model to

illustrate how the results of a computation-based analysis can differ from those of a structure-based analysis.

2. The Need

To illustrate the need for computation-based reliability analysis consider, for example, the reliability equations that have been developed for systems employing modular redundancy and sparing [3], [4]. These equations are based solely on a structural representation (at the architecture level) of the system in question and, consequently, whatever the underlying success criteria may be, they too are structure-based.

In particular, therefore, these criteria apply as well to a fault-tolerant aerospace computer as they do to a fault-tolerant pocket calculator. Let us examine, on the other hand, how the computational requirements for two such systems might differ.

A computer housed in an aircraft or spacecraft is called on to perform a variety of functions at different times and for different lengths of time during the course of a flight (see [8], for example). A pocket calculator may be required only to add or multiply (any time it is called on to do so). What constitutes successful computation is likewise very different. In the case of an aircraft or spacecraft computer, success criteria will vary according to what function is computed and when

the computation takes place. For example, more strict criteria might apply to flight control computations during an automatic landing than to graphic display computations performed while en route. Consequently, a structural failure of a given type might cause a computational failure if it occurs during an automatic landing, but might be tolerated if it occurs before that time. In the case of a pocket calculator, on the other hand, success criteria are simple and essentially independent of what is computed or when computations take place.

Given these differences in computational requirements, as they are perceived by the users of each system, let us examine the consequences of applying structure-based reliability equations of the type referred to at the outset. In the case of a pocket calculator, the reliability values determined by the equations may be quite meaningful to the user since, here, the success or failure of a computation corresponds closely to the success or failure of the structure (as defined by the structure-based success criteria of the model). In the case of an aircraft or spacecraft computer, on the other hand, the reliability values determined by the equations may be misleading since, in general, the success or failure of computations will not correspond to the success or failure of the computer's structure. In

particular, as noted earlier, a structural failure might correspond to computational failure at one time (while landing) and to computational success at another time (while en route). In other words, "success in the use environment" may differ considerably from the kind of structure-based success criteria that models of this type employ.

The example just cited is indicative of the need to more fully account for the behavior of a computer (i.e., the computations it performs) when analyzing its reliability. To accomplish this, reliability measures must refer to concepts of system success which involve more than just the status of various components or subsystems.

For systems described at the architecture-level, this need has already been acknowledged by Bouricious, et al., [5], [6] through the introduction of parameter called "coverage." According to their definition, coverage is "the conditional probability that, given the existence of a failure in the operational system, the system is able to recover and continue information processing with no permanent loss of essential information." Thus, coverage involves the kind of "computation-based" success criteria, the use of which we are advocating.

To analytically evaluate coverage and, more generally, any computation-based reliability measure, the system models used must be capable of representing behavior (compu-

tations) as well as structure (the computer). In this regard, one should not be misled by the use of the coverage parameter in connection with purely structural models. Although such models can employ coverage as a parameter (as was done when the concept was first introduced [5]), they cannot be used to evaluate the parameter. The latter problem is the type of problem that we are concerned with here and one that we feel deserves further investigation.

In the remainder of this paper, our objective is to establish, in quite simple and general terms, the kinds of things that need to be considered in developing models and measures for computation-based reliability analysis.

3. Computers with Faults

We begin by viewing a digital computer as a rather general type of system which, at discrete points in time, receives input data which, in turn, effects changes in the system's internal state. It will be assumed that time is represented by the natural numbers, i.e., the time base is the set $T = \{0,1,2,...\}$. It will be further assumed that the state set is "coordinatized" where a subset of the coordinates represent the values of those state variables that are observable as output variables.

The transition structure of such a system may vary with time because faults (structural failures) occur or because the system is reconfigured in an attempt to recover from a fault. At a given instant of time the

structure is fixed, however, and is described by a transition function which determines the state of the computing system at time i + 1, given the state at time i and the input received at time i. Formalizing this notion, we have:

Definition: A (formal) computer is a system

$$C = (X,Q,\Delta)$$

where

X is a nonempty set, the input set of C,

Q is a nonempty set, the state set of C,

Δ is a sequence of functions

$$\Delta = (\delta_0, \delta_1, \delta_2, \ldots)$$

where δ_i: $Q \times X \to Q$, the transition function of C at time i ($i \in T$).

Thus a computer, as defined above, is a discrete-time, time-varying system whose structure at time i is described by transition function δ_i. In particular, if $q \in Q$ is the state of C at time i and $a \in X$ is the input received at time i then $\delta_i(q,a)$ is the state of C at time i + 1. In case structure does not vary with time, that is,

$$\delta_{i+1} = \delta_i, \quad \text{for all } I \in T \qquad (3.1)$$

then C is time-invariant. Thus if $C = (X,Q,\Delta)$ is time-invariant, Δ is uniquely determined by δ_0 and C can alternatively be regarded as a (state) sequential machine with (fixed) transition function $\delta = \delta_0$.

A computer is finite-input if $|X| < \infty$ and finite-state if $|Q| < \infty$. Note that even in case a computer is both finite-input and finite-state, it is not finitely specifiable unless its structure Δ is finitely specifiable. However, in the subsequent application of this model to reliability analysis, all computers (both fault-free and faulty) of concern in the analysis will indeed be finitely specifiable.

The most general view of computer behavior is that of "string manipulation." Beginning in some initial state q_0, determined by the program to be executed and by stored data, at some initial time i, C receives an input sequence of symbols $a_0a_1\ldots a_{n-1}$ where $a_j \in X$ is interpreted as the input received at time $i + j$. In response to this input sequence, there results a sequence (trajectory) of states $q_0q_1\ldots q_n$ where $q_j \in Q$ is interpreted as the state of C at time $i + j$. Thus the "state behavior" of C may be viewed as a function from $T \times X^*$ into Q^+ where T is the time base, X^* is the set of all finite-length sequences of input symbols (including the null sequence Λ), and Q^+ is the set of all finite-length sequences of states. More precisely, if $C = (X,Q,\Delta)$

and $q \in Q$, the state-behavior of C in q is a function α_q: $T \times X^* \to Q^+$ defined inductively as follows for all $i \in T$:

i) $\alpha_q(i,\Lambda) = q$

If $x \in X^*$, $a \in X$:

ii) $\alpha_q(i,xa) = \alpha_q(i,x)\delta_j(q',a)$

where $j = \ell g(x)$ (the length of x)

and q' is the final state of $\alpha_q(i,x)$

It is easy to verify that this formal notion of state-behavior captures the intuitive notion discussed above. Note that α_q maps input sequences of length n into state trajectories of length n + 1 .

Having established the concepts of "computer" and "state-behavior," we adopt a concept of "computation" that is somewhat more general than usually considered. Since computational errors may be due to erroneous initial states, initial times, and input sequences, as well as to faulty computers, we regard a computation as consisting of four things: an initial state q, an initial time i, an input sequence x and a state sequence y . More precisely, a computation (over X and Q) is a quadruple (q,i,x,y) where $q \in Q$, $i \in T$, $x \in X^*$ and $y \in Q^+$ such that $\ell g(y) = \ell g(x) + 1$. Accordingly, q, i, x, and y are referred to as the initial state, initial time, input sequence and state trajectory (respectively) of the com-

putation. Relative to a particular computer C, a computation of C is a computation of the form $(q,i,x,\alpha_q(i,x))$ The fundamental question of deciding whether a computer is a "success in its use environment" will be based on the nature of such computations. However, even more basic than the notion of a computational error is the concept of a "fault," that is, a transient or permanent change in computer structure that may, in turn cause errors.

In terms of the concepts of a "representation scheme" and a "system with faults" [9], the "specification class $\mathscr{S}$ and "realization class" $\mathscr{R}$ that we wish to consider is the class of all computers (as defined above), that is, both $\mathscr{S}$ and $\mathscr{R}$ are equal to the class

$$\mathscr{C} = \{C | C \text{ is a computer}\} .$$

Moreover, we will restrict our attention to faults that occur during the use of a computer (as opposed to faults that occur during the design process) and so, in the representation scheme $(\mathscr{C},\mathscr{C},\rho)$, ρ is taken to be the identity function. In this representation scheme, a "computer with faults" will be defined as follows.

The "fault-free" specification, that is, the description of the underlying system as it exists before any physical failures occur, will be assumed time-invariant (see condition (3.1)). This is not unreasonable since many physical systems and, in particular, most computing systems can be represented as time-invariant

systems as long as there are no structural changes due to physical failures. Suppose now that a physical failure does occur where the failure may be transient, permanent, or a combination of the two, that is, a permanent physical failure that has a transient component while the permanent change is taking place. Such physical failures can then be represented by (formal) faults as follows. If $C = (X,Q,\Delta)$ is a computer, a <u>fault of C</u> (at time i) is a triple (τ,π,i) where

τ: $Q \times X \to Q$, the <u>transient component</u>,

π: $Q \times X \to Q$, the <u>permanent component</u>,

i is a nonnegative integer, the <u>time of occurrence</u> ($i \in T$).

The interpretation of (τ,π,i) is a physical failure that occurs between time i and time $i + 1$. τ is the transition function that the failing system exhibits while the failure is taking place and π is the transition function that the system exhibits after the failure has taken place. Thus, if $f = (\tau,\pi,i)$ is a fault of computer $C = (X,Q,\Delta)$, the <u>result of f</u> is the computer $C^f = (X,Q,\Delta^f)$ where, if $\Delta^f = (\delta_0^f,\delta_1^f,\delta_2^f,\ldots)$ then

$$\delta_j^f = \begin{cases} \delta_j \text{ if } 0 \leq j < i \\ \tau \text{ if } j = i \\ \pi \text{ if } j > i \ . \end{cases} \qquad (3.3)$$

If, in the result of $f = (\tau,\pi,i)$, there is no permanent change in structure, that is, $\pi = \delta_{i-1}$ then f is a transient fault (at time i). A fault (π,τ,i) which represents no change whatsoever, that is, $\pi = \delta_{i-1}$ and $\tau = \delta_{i-1}$, is referred to as a null or improper fault (at time i). Given this notion of a fault at time i, by a "fault" we will generally mean a sequence of faults that represents a succession of physical failures. More precisely, a (multiple) fault of C is a sequence

$$f = (f_{i_1}, f_{i_2}, \ldots, f_{i_k})$$

where $i_1 < i_2 < \ldots < i_k$ and f_{i_j} is a fault of C at time i_j. The corresponding result of f is an immediate generalization of definition (3.3).

Given these concepts of "fault," "result of a fault," we obtain the following specialization of the general notion of a "system with faults."

Definition: A computer with faults is a triple (C,F,φ) such that

i) $C \in \mathscr{C}$ where C is time invariant,

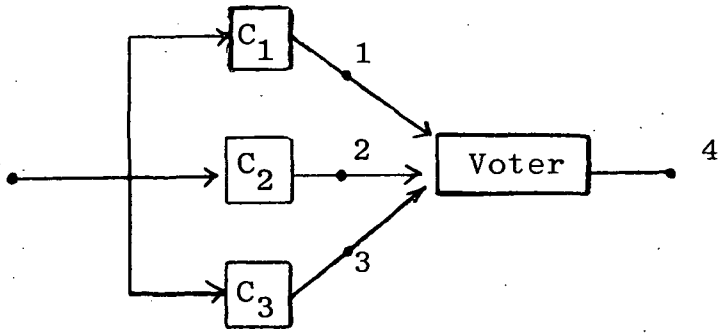
ii) F is a set of faults of C, where F contains at least one null fault,

iii) $\varphi: F \to \mathscr{C}$, where $\varphi(f) = C^f$ (the result of f).

In keeping with our earlier interpretations of these objects, if (C,F,φ) as a computer with faults, C will be

referred to as the fault-free computer and if f is not a null fault, C^f will be referred to as faulty.

To illustrate each of the ingredients of a computer with faults, consider the triply modular redundant (TMR) configuration:



where each module C_j is a time-invariant computer $\overline{C} = (\overline{X}, \overline{Q}, \overline{\Delta})$ with

$$\overline{X} = \overline{Q} = \{0,1\}$$

and $$\overline{\Delta} = \{\overline{\delta}, \overline{\delta}, \overline{\delta}, \ldots\} .$$

Then this (fault-free) TMR configuration is represented by the computer:

$$C = (\{0,1\}, Q, \Delta)$$

where

$$Q = \{(q_1, q_2, q_3, q_4) | q_i \in \{0,1\}\}$$

with q_1, q_2 and q_3 representing the states of modules 1, 2 and 3 (respectively) and q_4 representing the value of

the voter output. The transition structure

$$\Delta = (\delta_0, \delta_1, \delta_2, \ldots)$$

is given by a fixed function $\delta = \delta_i$ for all i, where

$$\delta((q_1, q_2, q_3, q_4), a) = (q_1', q_2', q_3', \mu(q_1', q_2', q_3'))$$

with $q_i' = \overline{\delta}(q_i, a)$ and μ equal to the majority function (realized by the voter).

To illustrate the concepts of a "fault" and the "result of a fault," suppose that at time 2 there is a transient struck-at-one failure at the output of module 1 and at time 4 there is a permanent stuck-at-zero failure at the output of module 3. Then this succession of failures is represented by the (multiple) fault

$$f = (f_2, f_4)$$

where f_2 is the fault at time 2 and f_4 is the fault at time 4. More specifically, f_2 is the fault

$$(\tau_2, \pi_2, 2)$$

where (letting $q_i' = \overline{\delta}(q_i, a)$):

$$\tau_2((q_1, q_2, q_3, q_4), a) = (1, q_2', q_3', \mu(1, q_2', q_3'))$$

and

$$\pi_2 = \delta \quad .$$

f_4 is the fault

$$(\tau_4, \pi_4, 4)$$

where

$$\tau_4((q_1,q_2,q_3,q_4),a) = (q_1',q_2',0,\mu(q_1',q_2',0))$$

and

$$\pi_4 = \tau_4$$

The result of the fault f is the computer $C^f = (\{0,1\},Q,\Delta^f)$ where

$$\delta_j^f = \begin{cases} \delta \text{ if } 0 \leq j < 2 \\ \tau_2 \text{ if } j = 2 \\ \pi_2 \text{ if } j = 3 \\ \tau_4 \text{ if } j = 4 \\ \pi_4 \text{ if } j > 4 \ . \end{cases}$$

4. Tolerance Relations for Computations

Given the class of computers with faults (over some specified input set X and state set Q), we now consider the basic issue raised at the outset of this discussion, namely, the formulation of computation-based success criteria. Although such criteria could be formally specified in a variety of specific ways, the following general formulation appears to be quite reasonable.

We view a particular computation realized by some possibly faulty computer as being a "success" if it is "within tolerance" of the desired (error-free) computation. In these terms, what is regarded as a successful computation (in the use environment) is specified by a "tolerance relation" on the set of all possible computations. In general, such a relation can be formally defined as follows.

Definition: If U is the set of all computations (over X and Q), a tolerance relation (for computations) is a relation σ on U such that σ is reflexive.

The reflexive condition of the definition says simply that every computation is within tolerance of itself. Accordingly the strongest tolerance relation is the relation of equality; the weakest is the relation $\sigma = U \times U$ where every computation is within tolerance of every other computation. The latter says that anything the computer does is acceptable and therefore represents a theoretical extreme as opposed to a practical one.

It should also be noted that the concept of tolerance, as defined above, is general enough to permit tolerable deviations in initial state, initial time and input as well as tolerable deviations in the state trajectory. Thus, for example, if (q,i,x,y) were the desired computation and a delay of up to 5 time steps could be tolerated then

$$(q,i+j,x,y)\sigma(q,i,x,y). \qquad (1\leq j\leq 5)$$

However, the purpose of the present investigation can be adequately served by examining how internal causes (faults) affect the state trajectory of a computation, and neglecting external causes that might affect initialization, timing, and input.

Given a computer with faults (C,F,φ) and some specified tolerance relation σ, it is now possible to define precisely what is meant by computational success. To

this end, suppose $f \in F$, u is a computation of the (possibly faulty) computer C^f and u' is the computation of the fault-free computer C where u' has the same initial state, initial time, and input sequence as u. Then, assuming no external causes of error, we can regard u as a "success" if u is within tolerance of u'. More precisely, if σ is a tolerance relation and $f \in F$:

Definition: A computation u of C^f is a σ-success if $u\sigma(q,i,x,\alpha_q(i,x))$ where q, i, and x are the initial state, initial time and input sequence of u. Otherwise u is a σ-failure.

If a computation of C^f is a σ-failure, we will say it is caused by f. In case f can cause no σ-failures, then f is σ-tolerated. When the tolerance relation is understood, we will drop the reference to σ and refer to a computation u as simply a "success" or, in the opposite case, a "failure."

Since state trajectories can be distinguished by a tolerance relation, the concept of failure, as defined above, can capture internal computational failures as well as input-output failures. To illustrate, let C be the TMR configuration considered earlier and suppose σ is the relation of equality on U (i.e., $u\sigma u'$ iff $u = u'$). Then the fault $f = (f_2, f_4)$, considered in the earlier example, can cause σ-failures even though f cannot cause

input-output failures (assuming the modules are properly initialized). To be more specific, let us suppose the module transition function $\bar{\sigma}$ is given by:

(q,a)	$\bar{\delta}$(q,a)
(0,0)	0
(0,1)	1
(1,0)	1
(1,1)	0

Then, for example, if q = (0,0,0,0), i = 0, and x = 101 then

$$\alpha_q^f(i,x) = (0,0,0,0)(1,1,1,1)(1,1,1,1)(1,0,0,0)$$

since the transition function at time 2 is τ_2 . On the other hand

$$\alpha_q(i,x) = (0,0,0,0)(1,1,1,1)(1,1,1,1)(0,0,0,0) .$$

Thus the computations $u = (q,i,x,\alpha_q^f(x))$ and $u' = (q,i,x,\alpha_q(x))$ are not equal, that is $u \not\phi u'$ and hence u is a σ-failure.

To continue the example, suppose σ' is a second tolerance relation which requires only that values on the output line (coordinate 4) be what they should be. More precisely, $(q,i,x,y)\sigma'(q,i,x,y')$ if y and y' have the same length, say n, and the j^{th} state of y has the

the same 4^{th} coordinate as the j^{th} state of y', i = 0,1,...,n-1 . Given that σ' is the tolerance relation of interest, it can be shown that the fault $f = (f_2, f_4)$ does not cause any σ'-failures (provided all module states are the same when the computation begins). In other words, although f can cause internal failures (according to tolerance relation σ), it can cause no input-output failures (according to tolerance relation σ').

5. Reliability Measures

In general, a "realiability measure" is a function from some class of systems into some set of numbers (or product set of numbers) whose value, for a given system, reflects the ability to rely on that system in some specified use environment. When viewed in this way, the concept of a reliability measure includes such measures as "mean-time-to-failure," "availability," "recoverability," etc., as well as the measure "probability of success (in the use environment)." What we wish to examine now is how such reliability measures might be formulated in terms of the computation-based success criteria developed in the previous section. The investigation will focus on the measure "probability of success," although other measures of the type mentioned above could be dealt with in a similar fashion.

In general, to formulate the measure "probability of success" where, as earlier, success means "success in the use environment," the probabilistic nature of both the system and the environment must be taken into account. In classical structure-based formulations, the environment is usually described by a single parameter t (the duration of time that the system is utilized) and assumed to be deterministic (i.e., it is assumed that t has a known fixed value when evaluating probability of success). However, when other aspects of the environment are considered, such as the computational requirements of the user, it is more realistic to regard the environment as probabilistic.

To formalize this view, if (C,F,φ) is a computer with faults where $C = (X,Q,\Delta)$, the environment of C can be represented by a probability space

$$(E, \mathscr{E}, P_E) \tag{5.1}$$

where

$$E = Q \times T \times X^* ,$$

$$\mathscr{E} = \{E' \mid E' \subseteq E\} \quad \text{(the "events" on E)} ,$$

$P_E : \mathscr{E} \to [0,1]$ is a probability measure.

Here, an element (q,i,x) in the sample space E describes an environment wherein the computer is to realize a computation with initial state q, initial time i, and

input sequence x. The interpretation of the probability measure P_E is the usual one, that is, if $E' \in \mathscr{E}$ then:

$P_E(E')$ = the probability that the (experienced) environment is in the event E'.

As for the probabilistic nature of the faults of C, it can be represented by a second probability space

$$(F, \mathscr{F}, P_F) \qquad (5.2)$$

where F is the set of faults of C

$\mathscr{F} = \{F' \mid F' \subseteq F\}$

and $P_F : \mathscr{F} \to [0,1]$ is a probability measure.

Again the interpretation of P_F is the usual one, that is, if $F' \in \mathscr{F}$ then:

$P_F(F')$ = the probability that the (experienced) fault is in the event F'.

Given the spaces $(E, \mathscr{E}, P_E)$ and $(F, \mathscr{F}, P_F)$, the probabilistic nature of both the environment and the faults of C can then be represented by a single space

$$(G, \mathscr{G}, P)$$

where $G = E \times F$,

$\mathscr{G} = \{G' \mid G' \subseteq G\}$,

and $P : \mathscr{G} \to [0,1]$ is the (unique) probability measure that satisfies the condition:

$$P(\{(e,f)\}) = P_E(\{e\}) \cdot P_F(\{f\}), \text{ for all } (e,f) \in G \qquad (5.3)$$

Note that Eq. (5.3) expresses an underlying assumption that environmental events are independent of faults, which we feel is quite reasonable. If $G' \in \mathcal{G}$, the interpretation of $P(G')$ is the probability that the (experienced) environment and fault has a description e and f, respectively, such that $(e,f) \in G'$.

A probabilistic framework has now been established for a formal definition of "probability of success in the use environment" or what we will refer to simply as "reliability."

Definition: If $C = (C,F,\varphi)$ is a computer with faults, σ is a tolerance relation on the computations of C, and P is the probability measure defined by Eq. (5.3) then the reliability of C (denoted $R_\sigma(C)$) is the probability

$$R_\sigma(C) = P(H)$$

where $H = \left\{ (q,i,x,f) \,\middle|\, \begin{array}{l} \text{the computation } (q,i,x,\alpha_q^f(i,x)) \\ \text{is a } \sigma\text{-success} \end{array} \right\}$.

Note that R_σ may be viewed as a reliability measure (from computers with faults into the real interval $[0,1]$) where the value of R_σ for computer C is $R_\sigma(C)$. Thus the above definition yields a whole class of reliability measures that differ according to the choice of a tolerance relation σ.

6. Analysis of a Read-only Memory

To illustrate the application of computation-based reliability measures, let us suppose the system to be analyzed is a 1024 word, 32 bit/word read-only memory (ROM). Then the ROM (before the occurrence of any physical failures), can be represented by the fault-free computer $C = (X,Q,\Delta)$ where

$$X = \{0,1\}^{10}$$

$$Q = X \times Y \quad \text{where } Y = \{0,1\}^{32}$$

and $$\Delta = (\delta,\delta,\delta,\ldots\}.$$

To describe the (fault-free) transition function δ, with each "address" $a \in X$ we associate a word $c(a) \in Y$, the "content of a." Then for all $q \in Q$,

$$\delta(q,a) = (a,c(a)) .$$

Suppose further that the physical failures of concern are memory cell failures that permanently alter the content of an address. Then, for some specific address b, such failures can be formally represented by (single) faults of the form

$$f(b,i) = (\tau,\pi,i)$$

where $\tau = \pi$ (i.e., $f(b,i)$ is a permanent fault) and

$$\pi(q,a) = \begin{cases} \delta(q,a) \text{ if } a \neq b \\ (a,c), \text{ where } c \neq c(a), \text{ otherwise.} \end{cases}$$

If null-faults of the form $f = (\delta,\delta,i)$ are also included, then the fault set F is the set of all sequences of single faults, i.e., $f \in F$ if and only if, for some $m \geq 1$,

$$f = (f(b_1,i_1),f(b_2,i_2),\ldots,f(b_m,i_m))$$

where $i_1 < i_2 < \ldots < i_m$. As for the underlying tolerance relation σ , we assume that no readout errors can be tolerated (i.e., there can be no errors in the Y coordinate values of a state). As no failures are postulated for the addressing structure (which determines the X coordinate values), we can therefore take σ to be the relation of equality on the computations of C.

Regarding the environment of the system, let us suppose the ROM is part of an aircraft computer where it receives slowly changing address updates at the rate of 1 per minute. (Time i will be interpreted as the i^{th} minute). Let us suppose further that as inputs change, the likelihood of repeating a given address is negligible. Then, for a mission duration of t minutes (where $t \leq 1024$), the environment of the ROM is described by a probability space $(E, \mathcal{E}, P_E)$, where E and $\mathcal{E}$ are as defined in (5.1), and P_E is subject to the condition that, whenever

$$P_E(\{(q,i,a_0a_1\ldots a_{\ell-1})\}) > 0 \qquad)$$

then $i = 0$, $\ell = t$, and $a_j \neq a_k$ if $j \neq k$.

The probabilistic nature of the faults $f \in F$ is determined by that of the physical memory cells. Assuming that a single cell failure changes the contents of that cell, a fault $f(b,i)$ corresponds to no failures in each of the 32 cells (addressed by b) until time i and then at least one cell failure (among the 32) between time i and time $i + 1$. Thus, if cell failures occur at constant hazard rate λ , the value of P_F (see (5.2)) for a fault $f(b,i)$ is given by:

$$P_F(\{f(b,i)\}) = e^{-32\lambda i}(1 - e^{-32\lambda})$$

$$= e^{-32\lambda i} - e^{-32\lambda(i+1)} .$$

The probability of a sequence of single faults can be formulated in a similar manner.

We now have enough information to determine the reliability of C according to the measure R_σ . By definition, $R_\sigma(C)$ is the probability P(H) of the event H consisting of all tuples (q,i,x,f) such that the computation $(q,i,x,\alpha_q^f(i,x))$ is a σ-success, that is $\alpha_q^f(i,x) = \alpha_q(i,x)$. We note first that $P(\{(q,i,x,\alpha_q^f(i,x))\})$ will be 0 if $i \neq 0$, $\ell g(x) \neq t$ or the sequence x has repeated addresses (since $P_E(\{(q,i,x)\}) = 0$ under these conditions). Thus we need only consider tuples (q,i,x,f)

such that $i = 0$, $x = a_0 a_1 \cdots a_{\tau-1}$, and $a_j \neq a_k$ if $j \neq k$. In this case, it follows that the computation $u = (q,0,x,\alpha_q^f(0,x))$ is a σ-success if and only if, for each time i $(0 \leq i \leq \tau - 1)$ and for all j such that $0 \leq j \leq i$, the fault $f(a_i,j)$ does not occur in the sequence f. This will ensure that, for each time i, the $(i+1)^{th}$ state of $\alpha_q^f(0,x)$ is equal to the $(i+1)^{th}$ state of $\alpha_q(0,x)$. Moreover, since a_i occurs exactly once in the sequence x, no fault $f(a_i,j)$ with $j > i$, will cause a σ-failure at a later time. Thus, if we let

$$F_x = \{f \mid (q,0,x,\alpha_q^f(0,x)) \text{ is a } \sigma\text{-success}\} ,$$

it follows that

$$F_x = \prod_{i=1}^{t} e^{-32\lambda i}$$

$$= e^{\frac{-32\lambda t(t+1)}{2}} .$$

Summing over the environment

$$\overline{E} = \{(q,0,x) \mid P_E(\{(q,0,x)\}) > 0\}$$

we have:

$$P(H) = \sum_{(q,0,x)\in\overline{E}} P_E(\{q,0,x\}) \cdot P_F(F_x)$$

$$= \sum_{(q,0,x)\in\tilde{E}} P_E(\{q,0,x\}) \cdot e^{\frac{-32\lambda t(t+1)}{2}}$$

$$= 1 \cdot e^{\frac{-32\lambda t(t+1)}{2}}$$

Hence $R_\sigma(C) = e^{\frac{-32\lambda t(t+1)}{2}}$. (6.1)

The above example is intended to illustrate the many concepts introduced in previous sections and, for this reason, the development has been somewhat more lengthy than what would normally be required to achieve the end result. The example also serves to illustrate how a computation-based reliability measure can differ from a structure-based measure. In particular, suppose we consider the usual structure-based success criteria for the example in question, that is, "no memory cell failures during the utilization interval t." Since the ROM has 1024 words with 32 bits/word, the reliability (probability of success) in this case is given by

$$R(C) = e^{-(1024)32\lambda t} \qquad (6.2)$$

Comparing Eqs. (6.1) and (6.2), we note first that the structure-based formula says that the system failure rate is constant, while the computation-based formula does not. Second, we note that, even in the case of maximum utilization for the environmental assumptions

of (6.1) (i.e., t = 1024) wherein all addresses are interrogated, the effective failure rate given by the computation-based measure is only half that of the structure-based measure. To obtain a more concrete comparison, suppose that the memory cell hazard rate is 10^{-7} failures per hour and the utilization interval is 10 hours (which might be required of a long-range aircraft). Then $\lambda = 10^{-7}/60$, $t = 600$ and substituting in Eq. (6.1):

$$R_{\sigma}(C) = e^{-9.616\times10^{-3}}$$

$$= .9904 .$$

On the other hand, substituting these same values in (6.2):

$$R(C) = e^{-3.2768\times10^{-2}}$$

$$= .9680 .$$

Thus the computation-based measure yields a considerably higher estimate of the ROM's reliability (in this use environment) than does the structure-based measure. Judging by other examples we have looked at, this kind of difference is typical. In other words, structure-based measures will often yield a more pessimistic view of a computer's reliability than is warranted by the computational needs of the user.

7. Conclusion

The purpose of this investigation has been to give a precise meaning to the notion of a computation-based reliability analysis in terms of a simple but general model of a computer with faults. This is not to suggest that the proposed model is the only one that could be or should be used in the analysis of a specific class of computing systems. The investigation does indicate, however, the kinds of things that should be considered if reliability measures are to more accurately reflect computational needs of the user. It is hoped that this will provide a framework for more detailed investigations regarding the feasibility of computation-based analysis methods.

REFERENCES

[1] J. von Neumann, "Probabilistic logics and the synthesis of reliable organisms from unreliable components," Automata Studies, (Ed. by C. E. Shannon and J. McCarthy), Princeton University Press, Princeton, N.J., 1956, pp. 43-98.

[2] E. F. Moore and C. E. Shannon, "Reliable circuits using less reliable relays," Journal of the Franklin Institute, Vol. 262, part I, pp. 191-208, part II, 1956, pp. 281-297.

[3] F. P. Mathur and A. Avizienis, "Reliability analysis and architecture of a hybrid-redundant digital system: Generalized triple modular redundancy with self-repair," Proc. 1970 Spring Joint Computer Conference, AFIPS Conf. Proc., Vol. 36, 1970, pp. 375-383.

[4] F. P. Mathur, "On reliability modeling and analysis of ultra-reliable fault-tolerant digital systems," IEEE Transactions on Computers, Vol. C-20, No. 11, November, 1971, pp. 1376-1382.

[5] W. G. Boricius, W. C. Carter, and P. R. Schneider, "Reliability modeling techniques for self-repairing computer systems," Proceedings ACM 1969 Annual Conference, 1969, pp. 295-309.

[6] W. G. Boricious, W. C. Carter, D. C. Jessep, P. R. Schneider, and A. B. Wadia, "Reliability modeling for fault tolerant computers," IEEE Transactions on Computers, Vol. C-20, No. 11, November, 1971, pp. 1306-1311.

[7] A. M. Breipohl, Probabilistic Systems Analysis, John Wiley and Sons, Inc., New York, 1970.

[8] R. S. Ratner, et al., "Design of a fault tolerant airborne digital computer (Vol. II - Computational requirements and technology)," Final Report, SRI Project 1406, Stanford Research Institute, Menlo Park, California, October, 1973.

[9] J. F. Meyer, "A general model for the study of fault tolerance and diagnosis," Proc. of the 6th Hawaii International Conference on System Sciences, January, 1973, pp. 163-165.