NASA TM-86423

NASA Technical Memorandum 86423

NASA-TM-86423 19850019305

An Abstract Specification Language for Markov Reliability Models

Ricky W. Butler

April 1985

NASA
National Aeronautics and
Space Administration

Langley Research Center
Hampton, Virginia 23665

INTRODUCTION

The reliability analysis of an aircraft's or a spacecraft's electronics system is an essential part of both the design and the validation process. Traditional electronics systems were static in nature, not relying on system reconfiguration for fault tolerance. For such systems, combinatorial mathematics were adequate to analyze system reliability. A graphical representation of such a combinatorial analysis, called "Fault-tree analysis", is frequently utilized by reliability engineers. Unfortunately, for systems where reconfiguration is used, the fault-tree approach is inadequate. This is true whether the reconfiguration is accomplished by replacing a faulty unit with a spare or by removing the faulty unit and degrading to a lower level of redundancy. The more powerful Markov model approach must be used to analyze such systems.

Recently, a new mathematical technique was developed enabling the efficient computation of such models. (See ref. 1.) This technique was embedded in a new reliability analysis tool called SURE. (See ref 2.) The computational power of the tool enables it to process extremely large and complex models. However, the process of defining such models is still quite tedious. For well-structured systems such as the Software Implemented Fault Tolerance (SIFT) system (see ref. 3) very small and simple models can capture the essential fault tolerance behavior. Also, highly modular systems consisting of many independent "stages" can be analyzed by dealing with each stage in a separate model and computing the overall system reliability with simple combinatorics. However, for large highly integrated systems a single very large model may be necessary. This problem is not related to any particular reliability analysis tool, but arises fundamentally from the complex nature of the system. The process of defining such a model can be tedious and error-prone. In this paper, a new approach to defining such models using an abstract model definition language is described.

N85-27616 #

THE MODEL DEFINITION CONCEPT

The process of modeling a fault-tolerant system is not an exact science. It is still very much an art. Reliability analysts must study a fault-tolerant architecture and capture the essential aspects of its design which contribute to its fault tolerance. The modeling process is perhaps best described by example. Suppose we have a SIFT-like system consisting initially of six independent processors each executing the exact same program. Each processor receives exactly the same inputs so that all non-faulty processors produce exactly the same output. Furthermore we assume the system "votes" the outputs prior to external use. Thus, so long as a majority of the processors are non-faulty, any erroneous values are "masked" while the system removes the faulty processors via reconfiguration. The Markov model in figure 1 characterizes this system.

$$\begin{tikzcd}
(6,0) \arrow[r, "6\lambda"] & (6,1) \arrow[r, "5\lambda"] \arrow[d, "\delta"] & (6,2) \arrow[r, "4\lambda"] \arrow[d, "2\delta"] & (6,3) & & \\
 & (5,0) \arrow[r, "5\lambda"] & (5,1) \arrow[r, "4\lambda"] \arrow[d, "\delta"] & (5,2) \arrow[r, "3\lambda"] \arrow[d, "2\delta"] & (5,3) & \\
 & & (4,0) \arrow[r, "4\lambda"] & (4,1) \arrow[r, "3\lambda"] \arrow[d, "\delta"] & (4,2) & \\
 & & & (3,0) \arrow[r, "3\lambda"] & (3,1) \arrow[r, "2\lambda"] \arrow[d, "\delta"] & (3,2) \\
 & & & & (2,0) \arrow[r, "2\lambda"] & (2,1)
\end{tikzcd}$$

Figure 1. Markov Model of SIFT-Like Architecture

The states of the model are defined by the ordered pair (NC,NF), where NC =

the number of processors currently in the configuration, and NF = the number of faulty processors in the configuration. There are three main concepts that dictate the structure of this model:

1. Every processor in the current configuration fails at rate λ.

2. The system removes faulty processors at rate δ.

3. A majority of processors in the configuration must not have failed in order for the system to be "safe".

It is the goal of the abstract model definition language introduced in this paper to express concepts such as these so that an automatic generation of the corresponding Markov model is possible. Such a capability could be used in conjunction with a Markov model analysis program such as SURE to provide a high-level reliability analysis work station. This concept is illustrated in figure 2.

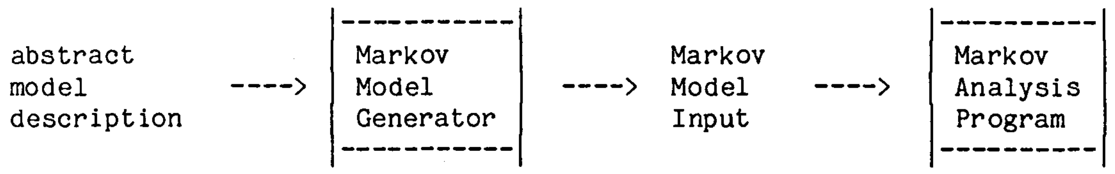

Figure 2. Reliability Analysis Work-station Concept

THE ABSTRACT LANGUAGE

A formal description of the language will not be presented. This paper is not intended as a specification for the design of a translator but rather as an exposition of the problem of Markov model description and a possible approach to the problem. Nevertheless, it is necessary to define a few conventions to facilitate the description of the language:

1. All reserved words will be underlined.

2. Lowercase words which are surrounded by quotes, such as "const", indicate items which will be replaced by something defined elsewhere.

3. Items enclosed in braces { } may be omitted or repeated as many times as desired.

Language Details

The language consists of 5 types of statements:

1. The constant definition statement
2. The <u>SPACE</u> statement
3. The <u>START</u> statement
4. The <u>DEATH-IF</u> statement
5. The <u>TRANTO</u> statement

Each of these statements will be discussed in the following sections.

<u>The constant definition statement</u> - A constant definition statement equates an identifier consisting of letters and digits to a number. For example:

```
LAMBDA = 0.0052;
RECOVER = 0.005;
```

Once defined, an identifier may be used instead of the number it represents. In the following sections, the phrase "const" will be used to represent a constant which can be either a number or a constant identifier. Constants may also be defined in terms of previously defined constants:

```
   LAMBDA = 1E-4;
   GAMMA = 10*LAMBDA;
```

In general the syntax is

```
 "ident" = "expression";
```

Where "ident" is a string of up to 8 characters and digits beginning with a character and "expression" is an arbitrary mathematical expression using constants and any of the following operations:

+	addition
-	subtraction
*	multiplication
/	division
**	exponentiation

=	equals
>	greater than
>=	greater than or equal
<	less than
<=	less than or equal

AND	logical and
OR	logical or
NOT	logical not

and functions:

EXP(X)	exponential function
LN(X)	natural logarithm
SIN(X)	sine function
COS(X)	cosine function
ARCSIN(X)	arcsine function
ARCCOS(X)	arccosine function
ARCTAN(X)	arctangent function
SQRT(X)	square root

Both ( ) and [ ] may be used for grouping in the expressions. The following commands contain legal expressions:

```
ALPHA = 1E-4;
RECV = 1.2*EXP(-3*ALPHA);
DELTA = (ALPHA + 2.3E-5)*RECV;
```

The SPACE statement - This statement is used to specify the state space on which the Markov model is defined. Essentially, the state space is defined by a n-dimensional vector where each component of the vector defines an attribute of the system being modelled. In the SIFT-like architecture above the state space was (NC,NF). This would be defined in the abstract language as

```
SPACE = (NC: 0..6, NF: 0..6);
```

The 0..6 represents the range of values over which the components can vary. The number of components (i.e. the dimension of the vector space) can be as large as desired. In general the syntax is

```
SPACE = ( "ident": "const" .. "const" {, "ident": "const" .. "const"} )
```

The identifiers, "ident", used in the SPACE statement will be referred to as the "state space variables".

The START statement - This statement indicates which state represents the start state. This state corresponds to the initial state of the system being modeled, i.e. the probability the system is in this state at time 0 is 1. In the SIFT-like architecture described above the initial state was (6,0). This is specified in the abstract language by the following:

```
START = (6,0);
```

In general the syntax is:

```
START = ( "const" {, "const" } );
```

The dimension of the vector must be the same as in the SPACE statement.

The DEATH-IF statement - The DEATH-IF statement specifies which states are death states, i.e. absorbing states in the model. The following is an example in the space (DIM1: 2..4, DIM2: 3..5)

```
DEATH-IF (DIM1 = 4) OR (DIM2 = 3);
```

This statement defines (4,3), (4,4), (4,5), (2,3), and (3,3) as death states. In general the syntax is

```
DEATH-IF "expression".
```

The expression in this statement must be a boolean expression.

The TRANTO statement - This is the most important statement in the language. It is used to describe and consequently generate the model in a recursive manner. The following statement generates all of the fault arrival transitions in the figure 1 model:

```
IF NC > 0 THEN TRANTO (NC, NF+1) BY (NC-NF)*LAMBDA;
```

The general syntax for a TRANTO statement is

IF "expression" THEN TRANTO ("expression", {,"expression"}) BY "expression"

In all of the expressions of this statement the state space variables may be used. The value of a state space variable is the corresponding value in the source state to which the TRANTO statement is being applied. For example, if the TRANTO statement is currently being applied to state (4,5) and the state space was defined by SPACE = (A: 0..10, Z: 2..15) then A = 4 and Z = 5. The first expression following the IF must be a boolean expression. Conceptually, it determines whether this rule will apply to a particular state. For example, in the state space SPACE = (A1: 1..5, A2: 0..1), the expression (A1 > 3) AND (A2 = 0) is true for states (4,0) and (5,0) only. The vector following the TRANTO reserved word defines the destination state of the transition to be added to the model. Each expression within the parentheses must evaluate to an integer. For example, if the state space is (X1, X2) and the source state is (5,3), then the vector (X1+1, X2-1) refers to (6,2). The expression following the BY indicates the rate of the transition to be added to the model. This expression must evaluate to a real number.
The TRANTO statement is applied to every state in the model as described by the following model generation algorithm:

```
1. initialize READY-SET to contain start state only

2. select a state from READY-SET. If the selected state is a death state
   as defined by a DEATH-IF statement, then remove it from the READY-SET
   and repeat step 2, otherwise continue on to step 3.

3. apply each TRANTO rule to the selected state as follows:

   3a. if the destination state is new, add it to the model and the READY-
       SET.
   3b. add the transition to the model.
   3c. remove selected state from READY-SET

4. go to step 2.
```

Examples

In this section the abstract model definition language will be illustrated by two examples.

Example 1 - Now we can specify the model of figure 1 in the language:

```
NP = 6;             (* number of processors initially *)
LAMBDA = 1E-4;      (* fault arrival rate *)
DELTA = 3.6E3;      (* recovery rate *)

SPACE = (NC: 0..NP, NF: 0..NP);
START = (NP,0);

DEATH-IF 2*NF >= NC;

IF NC > 0 THEN TRANTO (NC, NF+1)   BY (NC-NF)*LAMBDA;  (* fault arrivals *)
IF NF > 0 THEN TRANTO (NC-1, NF-1) BY NF*DELTA;        (* system recovery*)
```

The two TRANTO statements correspond to the first two concepts used to define the model:

1. Every processor in the current configuration fails at rate λ.
2. The system removes faulty processors at rate δ.

The DEATH-IF statement corresponds to the third concept:

3. A majority of processors in the configuration must not have failed in order for the system to be safe.

The flexibility and power of this language can be seen by observing that only the NP=6 statement would have to be changed in order to model a similar system which initially contains 9 processors.

Example 2 - In this example a system consisting of 2 triads and an arbitrary number of spares will be modelled. If two processors fail in either triad then the system fails. As long as spares are available, a faulty processor in a triad is replaced from the spare pool. If no spares are available, then a triad is collapsed into a simplex and the other good processor is added to the spares pool. Once a simplex processor fails, the system fails. Spares fail at a different rate than active processors. The following model describes this system:

```
NSI   = 5;              (* Number of spares initially, can be anything *)
SPACE = (N1: 0..3,      (* Number of processors in first triad *)
         N2: 0..3,      (* Number of processors in second triad *)
         F1: 0..2,      (* Number of faulty processors in first triad *)
         F2: 0..2,      (* Number of faulty processors in second triad *)
         NS: 0..NSI);   (* Number of spares *)

START = (3,3,0,0,NSI);

LAMBDA = 5E-4;          (* failure rate of active processors *)
GAMMA  = 2E-5;          (* failure rate of spares *)
DELTA  = 3.6E3;         (* rate at which faulty processors are removed *)

DEATH-IF (2*F1 > N1) OR (2*F2 > N2);

IF N1 > 0 THEN TRANTO (N1,N2,F1+1,F2,NS) BY (N1-F1)*LAMBDA;
IF N2 > 0 THEN TRANTO (N1,N2,F1,F2+1,NS) BY (N2-F2)*LAMBDA;
IF NS > 0 THEN TRANTO (N1,N2,F1,F2,NS-1) BY NS*GAMMA;

IF ((F1 > 0) OR (F2 > 0)) AND (NS > 0) THEN
   TRANTO (N1,N2,F1,F2,NS-1) BY DELTA;

IF (F1 > 0) AND (NS=0) THEN TRANTO (1,N2,0,F2,NS+1) BY DELTA;
IF (F2 > 0) AND (NS=0) THEN TRANTO (N1,1,F1,0,NS+1) BY DELTA;
```

The DEATH-IF statement specifies that system failure occurs if a majority of processors fail in either triad. The first three TRANTO statements specify fault arrivals in the two triads and spares. The next TRANTO statement specifies recovery when spares are available. The last TRANTO statement describes the recovery process when no spare are available.
Although, increasing the number of processors in the spares pool significantly increases the size of the Markov model, it can be accomplished in this language by changing only one constant.

CONCLUSIONS

This paper has introduced an abstract Markov model definition language which can be used to specify reliability models. The language essentially defines a set of rules which are used to automatically generate the Markov model. These rules correspond to the basic concepts used to create models of fault-tolerant systems. A small number of statements in the language can be used to describe a very large model. Furthermore, a variation in the system (such as in the number of initial spares) can be accomplished by changing only one line in the model definition, although such a change represents a large increase in the size of the generated Markov model.

APPENDIX

Language Extensions

The fundamental concept for an abstract specification language for Markov models has been developed in the main body of this paper. The constructs of the language have adequate expressive power to describe complex systems with a minimal number of statements. However, there are many possible extensions to this language which may further simplify the model description process. Several of these will be discussed briefly here.

Extension 1: Array State Variables. - The basic language allows the definition of state space variables with a SPACE statement. The language could easily be extended to allow an array of state space variables as follows:

```
SPACE = ( NC: ARRAY[1..3] OF 0..6,
          NF: ARRAY[1..3] OF 0..3 );
```

This statement creates a 6-dimensional space. The state space variables are NC[1], NC[2], NC[3], NF[1], NF[2], and NF[3].

Extension 2: FOR Statement. - Many times several TRANTO statements are needed which are identical except they operate on different state space variables. The FOR statement defines several TRANTO rules at once:

```
SPACE = ( NC: ARRAY[1..5] OF 0..6,
          NF: ARRAY[1..5] OF 0..3 );

FOR I IN {1..5} DO IF NC[I] > 0 THEN TRANTO NF[I] <- NF[I] + 1 BY LAMBDA;
```

This FOR statement is equivalent to five TRANTO statements, one for each value of I in the set {1..5}. The assignment statement after the TRANTO reserved word replaces the vector of the basic TRANTO statement. This statement defines the destination state of each new transition by specifying the change in a state variable from the source to destination state. There can be as many of these assignment statements after the TRANTO reserved word and before the BY reserved word as there are variables in the state space.

Extension 3: Nested IF THEN ELSE. The IF expression of the TRANTO statement could be extended in the obvious way:

```
IF "expression" THEN
    IF "expression" THEN
       TRANTO "vector" BY  "expression";
       TRANTO "vector" BY  "expression";
       TRANTO "vector" BY  "expression";
    ENDIF
ELSE
    TRANTO "vector" BY  "expression";
    TRANTO "vector" BY  "expression";
ENDIF
```

Extension 4: Semi-Markov Models - The "expression" following the BY statement could easily be extended to contain the mean and variance of a non-exponential transition (e.g. for SURE) as follows:

```
IF "expression" THEN TRANTO "vector" BY <"expression", "expression">;
```

REFERENCES

(1) White, Allan L.: Upper and Lower Bounds for Semi-Markov Reliability Models of Reconfigurable Systems. NASA CR-172340, 1984.

(2) Butler, Ricky W.: The Semi-Markov Unreliability Range Evaluator (SURE) Program, NASA TM-86261, July 1984.

(3) Goldberg, Jack, et. al.: Development and Analysis of the Software Implemented Fault Tolerance (SIFT) Computer. NASA CR-172146, 1984.

1. Report No. NASA TM-86423	2. Government Accession No.	3. Recipient's Catalog No.
4. Title and Subtitle An Abstract Specification Language for Markov Reliability Models		5. Report Date April 1985
		6. Performing Organization Code 505-34-13-32
7. Author(s) Ricky W. Butler		8. Performing Organization Report No.
		10. Work Unit No.
9. Performing Organization Name and Address NASA Langley Research Center Hampton, Virginia 23665		11. Contract or Grant No.
		13. Type of Report and Period Covered Technical Memorandum
12. Sponsoring Agency Name and Address National Aeronautics and Space Administration Washington, DC 20546		14. Sponsoring Agency Code
15. Supplementary Notes		

16. Abstract

Markov models can be used to compute the reliability of virtually any fault-tolerant system. However, the process of delineating all of the states and transitions in a model of a complex system can be devastatingly tedious and error-prone. This paper presents a new approach to this problem utilizing an abstract model definition language. This high-level language is described in a non-formal manner and illustrated by example.

17. Key Words (Suggested by Author(s)) Reliability Analysis Markov Models Reliability Modeling Fault Tolerance		18. Distribution Statement Unclassified - Unlimited Subject Category 65	
19. Security Classif. (of this report) Unclassified	20. Security Classif. (of this page) Unclassified	21. No. of Pages 16	22. Price A02

For sale by the National Technical Information Service, Springfield, Virginia 22161

End of Document