IN-62
80
74P

NASA Technical Memorandum 109101

TOTAL User Manual

Sally C. Johnson
Langley Research Center, Hampton, Virginia

David P. Boerschlein
Lockheed Engineering & Sciences Company, Hampton, Virginia

(NASA-TM-109101) TOTAL USER MANUAL (NASA. Langley Research Center) 74 p

N94-30160

Unclas

G3/62 0004780

April 1994

National Aeronautics and
Space Administration
Langley Research Center
Hampton, Virginia 23681-0001

TOTAL User Manual

Sally C. Johnson
and
David P. Boerschlein

NASA Langley Research Center
Hampton, VA 23681-0001

Abstract

Semi-Markov models can be used to analyze the reliability of virtually any fault-tolerant system. However, the process of delineating all of the states and transitions in the model of a complex system can be devastatingly tedious and error-prone. Even with tools such as the Abstract Semi-Markov Specification Interface to the SURE Tool (ASSIST), the user must describe a system by specifying the rules governing the behavior of the system in order to generate the model. With the Table Oriented Translator to the ASSIST Language (TOTAL), the user can specify the components of a typical system and their attributes in the form of a table. The conditions that lead to system failure are also listed in a tabular form. The user can also abstractly specify dependencies with causes and effects. The level of information required is appropriate for system designers with little or no background in the details of reliability calculations. A menu-driven interface guides the user through the system description process, and the program updates the tables as new information is entered. The TOTAL program automatically generates an ASSIST input description to match the system description.

Contents

List of Tables

List of Figures

1 Introduction

The Table-Oriented Translator to the ASSIST Language (TOTAL) computer program is a prototype spreadsheet interface for a reliability analysis tool set developed at NASA Langley. The tool set began with the development of several Markov and semi-Markov model solvers, which provide the flexibility to represent the failure behavior of virtually any fault-tolerant system. These include the Semi-Markov Unreliability Range Evaluator (SURE), the Scaled Taylor Exponential Matrix (STEM) solver, and the Padé Approximation With Scaling (PAWS) program[1, 2, 3]. Since the enumeration by hand of the states and transitions of the Markov model is only tractable for the simplest of systems, the ASSIST language was developed to allow the user to describe the failure behavior of the system in a high-level abstract language[4]. The ASSIST computer program was then developed to use the ASSIST description as a set of rules for automatically generating a reliability model[5]. The ASSIST program simply follows the rules as specified by the user and makes no assumptions about the system, so ASSIST is completely general and could theoretically be used to generate a reliability model for virtually any fault-tolerant system. However, the user must learn the concepts and syntax of the ASSIST input language before he can use it, and validation that the ASSIST description is accurate can be difficult for fault-tolerant systems with complex failure properties and dependencies.

The TOTAL interface program allows the user to describe a system at an even higher level of abstraction. Instead of providing the complete flexibility of the ASSIST language, the TOTAL interface provides the user with a structured set of choices for describing the system components and their interactions. The flow of information between the computer programs in this tool set is shown in Figure 1.

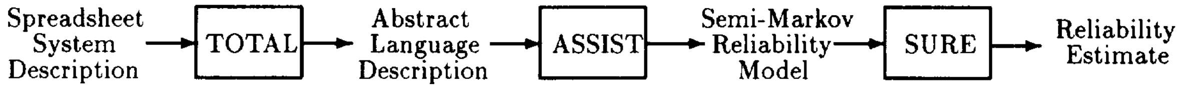

Spreadsheet System Description
TOTAL
Abstract Language Description
ASSIST
Semi-Markov Reliability Model
SURE
Reliability Estimate

Figure 1: Reliability Analysis Tool Set

The simplest systems are non-reconfigurable ones whose system failure is based on what combinations of its components have failed. The failure behaviors of these systems can be fully described by listing the components and their failure rates and enumerating which combinations of component failures lead to system failure, such as is done in a fault tree. This description is typically facilitated by grouping like components into subsystems.

Reconfigurable systems can be more complex to describe. Each subsystem may have spare components that are brought into the active configuration to replace failed components. Once the available spares are exhausted, a subsystem may degrade by removing failed components from the voting process. The system may provide full or partial detection of failed spares. Some spare components may be shared between subsystems and thus available to replace failed components in one or more subsystems.

More complex systems may exhibit failure dependencies between components; for example, failure of a power supply may cause the components dependent upon it to fail as well. A processor may depend on a network element for communication with the other processors in the system. Such dependencies can sometimes be included in the system failure combination descriptions. However, it is often more convenient and concise to list these as separate failure dependencies.

The level of system description given above seems appropriate for designers of fault-tolerant systems, even those with no background knowledge in reliability analysis calculations. This is the level of information needed to build a TOTAL system description. The user does not have to know how to build reliability models; he only needs to know the failure behaviors of his system.

The TOTAL program provides a menu-driven interface to guide the user through each phase of the system description. The system description consists of four parts: 1) a spreadsheet describing the components in the system, 2) specification of the failure dependencies between the components, 3) a description of the conditions representing system failure, and 4) a list of sets of pooled spares that can be shared between the components in the system. As the menu-driven interface guides the user through the system description process, the TOTAL program interactively builds and displays the set of tables describing the system. Once the system description process is complete, the user can generate the ASSIST-language description, execute the ASSIST program to enumerate the states and transitions of the model, and solve for the probability of system failure using the SURE, STEM, or PAWS programs, all from the TOTAL menu-driven interface.

The TOTAL program was developed as a prototype system to explore and demonstrate the feasiblity of a spreadsheet interface for reliability analysis. The prototype was developed and beta-tested with a limited menu-driven interface and no direct input to the displayed tables. If development were to continue, later versions of TOTAL would allow the user to directly input and manipulate data in the tables, and the TOTAL spreadsheet program would maintain consistency between tables. With these capabilities, the novice user would be guided in the system description process by the easy-to-follow menu interface, while the more experienced user would be able to efficiently enter data directly into the spreadsheet tables.

The TOTAL prototype cannot be considered a commercial-quality tool. The program was subjected to a moderate amount of testing inhouse plus a minimal amount of beta testing. Thus, the user is cautioned to examine the generated ASSIST-language description to validate that it is a reasonable representation of the intended system.

The menu-driven interface for the Version 1.0 prototype of TOTAL is introduced in Chapter 2. The details of how to use the interface to enter a system description are described and demonstrated for an example system in Chapter 3. Chapter 4 discusses generation and solution of the Markov model in general. Chapter 5 details the specific rules that are applied when generating the state transitions and the model. Chapter 6 details the system requirements for the TOTAL program, followed by some concluding remarks.

2 The TOTAL Interface

The menu-driven interface available in TOTAL Version 1.0 is introduced in this section. The user enters the interactive interface by entering the command "total". The user can modify the TOTAL program to accommodate large or unusual system descriptions using the command-line options shown in Appendix A. Upon entering TOTAL, the System Description panel shown in Figure 2 is displayed on the user's workstation. The figures in this manual show the panels as they are displayed on a SUN OS system running Motif. The panels displayed on a VMS computer contain the same information and "feel" the same but may look slightly different.

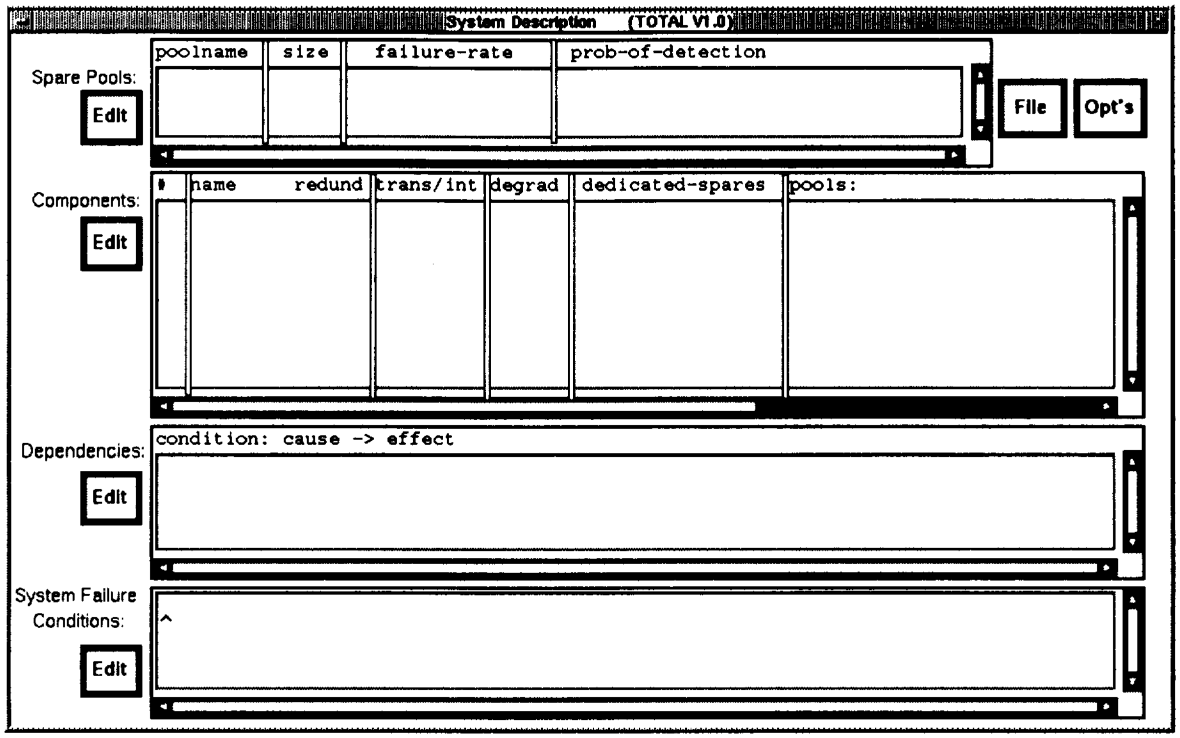


Figure 2: System Description Panel

This panel contains four tables for displaying the system description:

- Spare Pools
- Components
- Dependencies
- System Failure Conditions.

The Spare Pools table holds the descriptions of any pools of spare components that can be used to replace failed components. The Components table describes the components in the system.

Each line of the table represents one subsystem of the system, where a subsystem is made up of like components. Any failure dependencies are depicted in the Dependencies table. Finally, the System Failure Conditions table lists the conditions that are to be modeled as denoting system failure.

Beside each table is an "Edit" pull-down menu, represented as a rectangle[1], for entering or modifying entries in that table. The user displays the pull-down menu by moving the mouse cursor over the Edit pull-down menu and pressing and holding down the left mouse button. Figure 3 shows the user selecting the Append option from the Edit pull-down menu for the Components table.

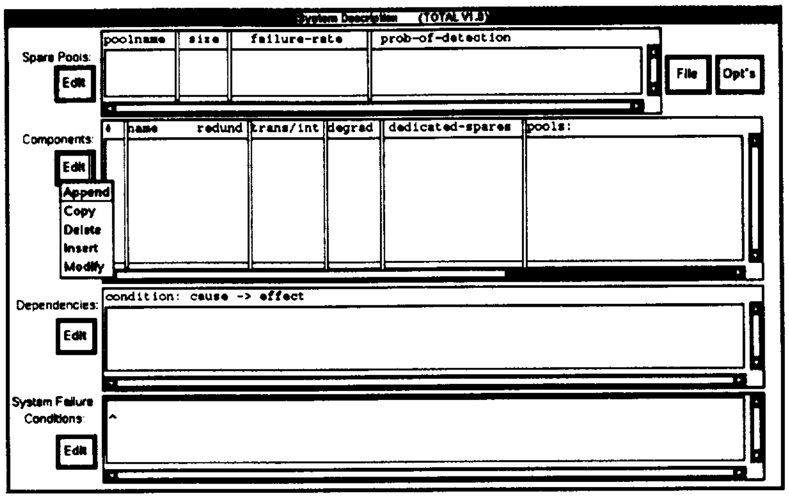


Figure 3: Using Edit Pull-Down Menu to Append a New Component

Each pull-down menu contains five options:

Append	-	Add a new entry to the end of the table
Copy	-	Create a new entry after a specified entry by modifying a copy of that entry description
Delete	-	Delete an entry
Insert	-	Insert a new entry before a specified entry
Modify	-	Modify an existing entry

While still holding down the left mouse button, the user selects an option by moving the mouse over the selection. When the user releases the left mouse button, a new menu panel corresponding to the option selected will appear on the screen. The user must use the Append option (and not the Insert option) to input the first entry in each table.

The System Description Panel also contains two additional pull-down menus: the File menu and the "Opt's" or Options menu. The File menu contains the following functions:

[1]On color terminals the pull-down menu rectangles are yellow

Erase	-	Delete all existing entries in the current system description
New	-	Delete all existing entries in the current system description and assign a new name to the current system description
Old	-	Delete all existing entries in the current system description and read in a system description from a file
Save	-	Save the current system description in a file under the current name
Save As	-	Save the current system description under the specified name
Generate ASSIST	-	Generate an ASSIST input description from the current system description
Edit ASSIST	-	Edit the generated ASSIST file using EVE under VMS or vi under Unix[2]
Run	-	Generate the model using ASSIST and solve the model using SURE
Edit Output	-	Edit the generated SURE output file as produced by the "Run" option.
Delete	-	Delete the specified system description file
Batch	-	Read in a set of TOTAL instructions from a batch file
Quit	-	Exit the TOTAL program

Figure 4 shows the user selecting to read an old file from the File menu.

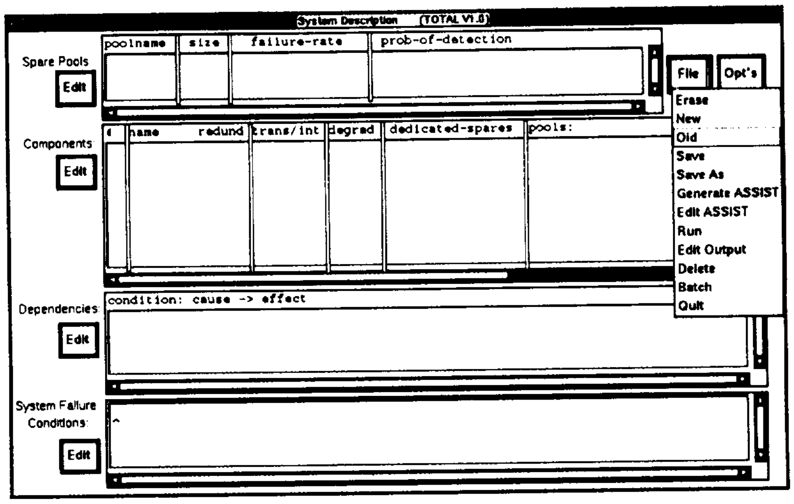


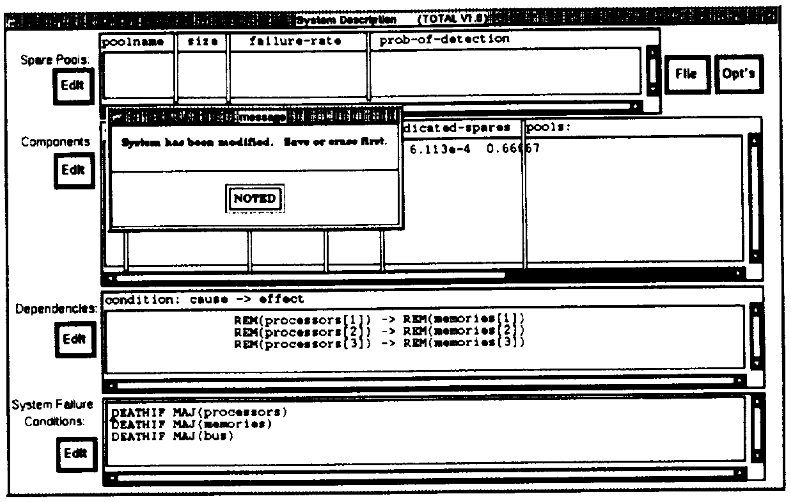
Figure 4: Using "File" pull-down menu

Once the system description is complete, the user should save the model with the "Save" or "Save As" function from the "File" menu before quitting. These functions copy the model description from the computer memory onto the disk file. Any model in memory when the user

[2]the EMACS editor is used when the **−ema** command line option is given

exits TOTAL or the system goes down will be lost forever. If the model is saved on a disk file, it will remain even if the computer is turned off.

If the user decides to quit without saving the model, TOTAL will protest as shown in Figure 5. The TOTAL program knows when a change has been made because the user had to click on an "Okay" button from some description panel in order to make a change. The program disallows the user from quitting or calling up another model with the "Old" function from the "File" menu until the old model has either been saved or erased.



poolname	size	failure-rate	prob-of-detection

```
System has been modified.  Save or erase first.
```

```
dicated-spares   pools:
6.113e-4  0.66667
```

```
condition: cause -> effect
                REM(processors[1]) -> REM(memories[1])
                REM(processors[2]) -> REM(memories[2])
                REM(processors[3]) -> REM(memories[3])
```

```
DEATHIF MAJ(processors)
DEATHIF MAJ(memories)
DEATHIF MAJ(bus)
```

Figure 5: Cannot quit until changes saved

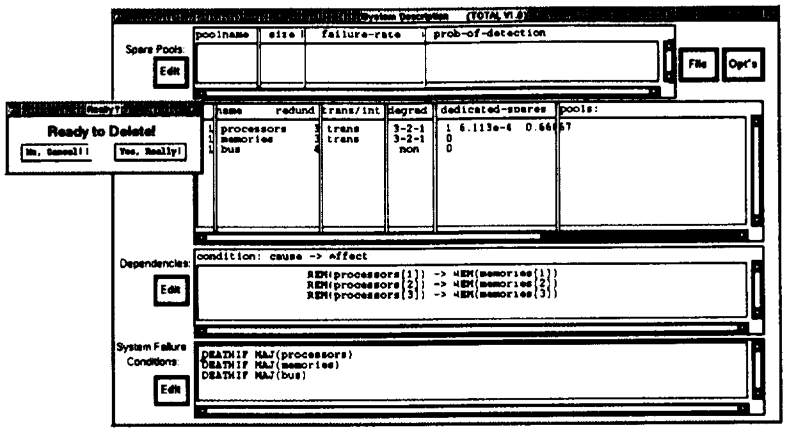
If the user decides to erase the changes entirely, the "File" menu is pulled down to the "Erase" function. The system verifies that the user really wants to erase the system from memory as illustrated in Figure 6.

As shown in Figure 7, the Options Menu contains the following three options:

- Set Code Generation Options/Values
- Show System Definitions
- System Description Comments and/or Remarks

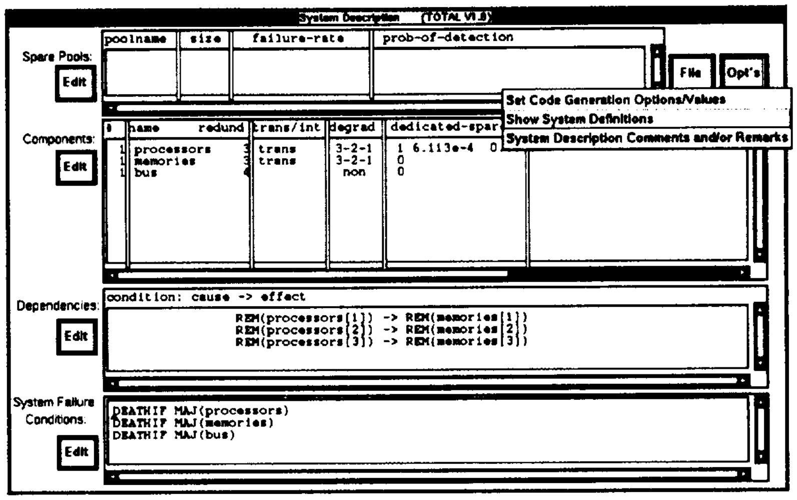
Figure 7 shows the user selecting the option to "Show System Definitions" from the "Opt's" options menu.

As the user describes his system using the menus, the system description is displayed in the tables. The text in the tables can extend past the right edge or the bottom of the tables, and this data can be viewed by scrolling each table using the scroll bars provided. Although tables can be scrolled, the text in the tables is static and cannot be entered or modified directly in Version 1.



```
poolname | size | failure-rate | prob-of-detection

# name        redund trans/int degrad dedicated-spares      pools:
1 processors  3      trans     3-2-1  1 6.113e-4  0.66667
1 memories    3      trans     3-2-1  0
1 bus         4                non    0

condition: cause -> effect
        REM(processors[1]) -> REM(memories[1])
        REM(processors[2]) -> REM(memories[2])
        REM(processors[3]) -> REM(memories[3])

DEATHIF MAJ(processors)
DEATHIF MAJ(memories)
DEATHIF MAJ(bus)
```

Figure 6: System verifies Erase option before taking action


System Description (TOTAL V1.0)
Spare Pools:
Edit
File
Opt's
Set Code Generation Options/Values
Show System Definitions
System Description Comments and/or Remarks
Components:
Edit
Dependencies:
Edit
System Failure Conditions:
Edit

```
poolname | size | failure-rate | prob-of-detection

# name        redund trans/int degrad dedicated-spar
1 processors  3      trans     3-2-1  1 6.113e-4  0
1 memories    3      trans     3-2-1  0
1 bus         4                non    0

condition: cause -> effect
        REM(processors[1]) -> REM(memories[1])
        REM(processors[2]) -> REM(memories[2])
        REM(processors[3]) -> REM(memories[3])

DEATHIF MAJ(processors)
DEATHIF MAJ(memories)
DEATHIF MAJ(bus)
```

Figure 7: Using pull-down menu to show system definitions

3 Example System Description

In this section, the TOTAL menu-driven interface will be described and illustrated by way of example. The example system consists of a triplex set of processors with one spare processor, a triplex set of memory units, and a quadruplex bus, as shown in Figure 8. All messages and calculations performed in the system are subject to majority voting to detect and mask failures. Thus, a majority of the processors in the current configuration must be working, or system failure occurs. Similarly, a majority of the memories and a majority of the buses must be working. Upon detection of the first processor failure, the spare processor is brought into the configuration to replace the faulty processor. Upon the second processor failure, the faulty processor is removed and the remaining two working processors continue in duplex mode. Failure of one of the remaining two processors is assumed to immediately defeat the majority voter. Each memory unit is attached to one of the processors, and removal of a failed processor also results in removal of its attached memory. The memory units degrade from triplex to duplex to simplex as failed memory units are detected in the configuration or are removed because of processor removals. The bus is non-reconfigurable.

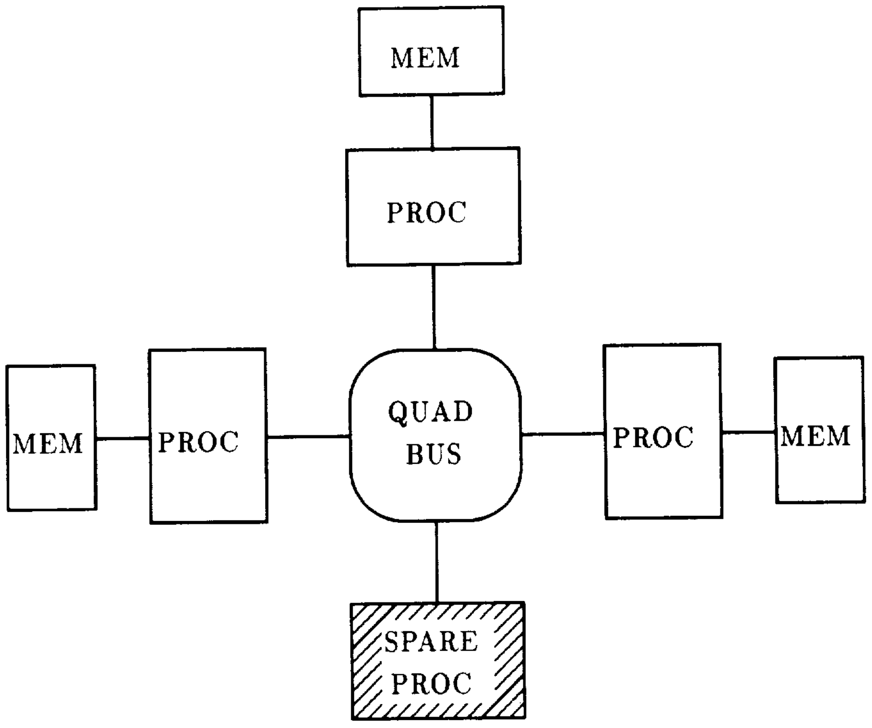


Figure 8: Example System

During the system description process, the user assigns names to identify the components in his system and defines any extra constants or state-space variables he might need. Except for special identifiers that are used by ASSIST and SURE, these identifiers are only meaningful to the user—the TOTAL program makes no implicit distinction between the behavior of a "processor" component, a "bus," or a "foobar." The user can input all rates and times to the program in whatever units (hours, minutes,...) he wishes, as long as he is consistent. However, the default mission time in SURE is 10 and must be changed if 10 units is not the desired mission time. The change can be made by editing the ASSIST output file to define "TIME = *value*;" after generating it and before running it.

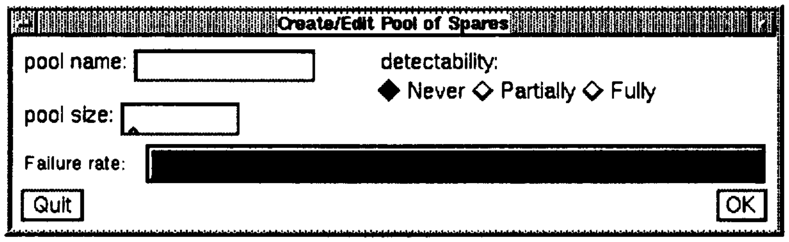
The following subsections detail how this example system can be described using the TOTAL interface.

3.1 Spare Pools

Spare pools are pools of spare components that can be used to replace multiple types of failed components. Consider, for example, a system that contains two triads of processors that share a pool of spare processors. The spare processors would be described in the Spare Pools table and given a name, such as "pspares". The two triads would then be described as two separate component types in the Components Table, and each would have "pspares" listed as a source of pooled spares.

The menu used for describing a pool of spares is shown in Figure 9. The inputs are as follows:

- Poolname $\longrightarrow$ Name of the pool, maximum 8 characters
- Size $\longrightarrow$ Number of components in pool initially
- Failure rate $\longrightarrow$ Failure rate for components while spares
- Probability of detection $\longrightarrow$ Probability that a failure will be detected before the component is brought into an active configuration (note that this shows up only when the detectability is set to "Partially". See Figure 11.


Create/Edit Pool of Spares

pool name:

pool size:

detectability: [✓ Never] Partially Fully

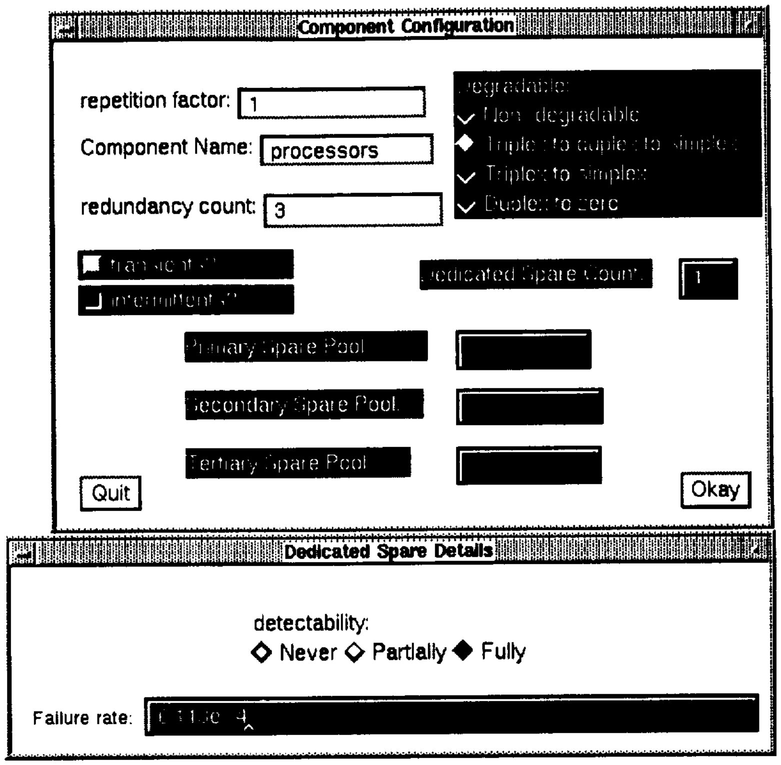
Failure rate:

Figure 9: Menu Panel for Describing Pool of Spares

Since our example system does not contain any spare components that are shared between more than one subsystem, the system description will not contain any spare pools.

3.2 Components

The example system description will contain three entries in the Components table: processors, memories, and the bus channels. To input the first entry, select the Append option from the Edit menu beside the Components table. Figure 10 shows the component configuration panel filled in with the correct information to describe the processors subsystem.


Component Configuration

repetition factor: 1

Component Name: processors

redundancy count: 3

Degradable:
Non degradable
[✓ Triple to duplex to simplex]
Triplex to simplex
Duplex to zero

Transient ?
Intermittent ?

Dedicated Spare Count: 1

Primary Spare Pool

Secondary Spare Pool

Tertiary Spare Pool

Quit

Okay

Dedicated Spare Details

detectability:
Never Partially [✓ Fully]

Failure rate: [illegible]

Figure 10: Component configuration for "processors"

The menu panel selection fields are as follows:

- Repetition Factor $\longrightarrow$ Number of subsystems of this type. The repetition feature is not supported in Version 1.0, so the default value of 1 must be used. In later versions, this will provide a convenient way to specify systems containing multiple similar subsystems, such as systems with multiple triads.

- Component Name $\longrightarrow$ Name used to identify this subsystem. May be as long as 16 characters, however, the first 8 characters[3] should be unique.

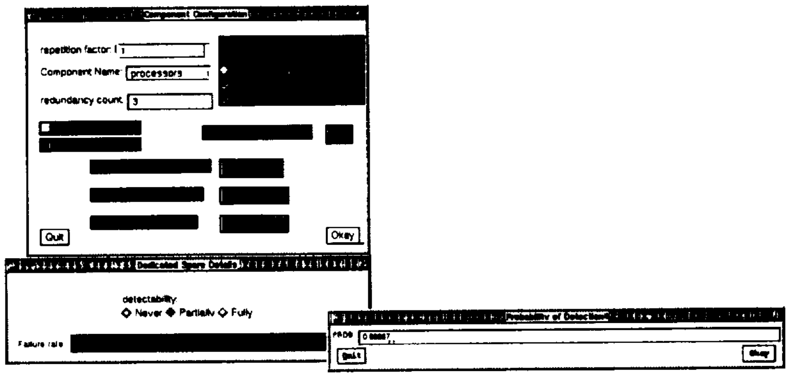
[3]TOTAL generates constant identifiers by using various prefixes of up to four characters each for constants. The SURE program requires the first 12 characters of an identifier to be unique. Although some IMPLICIT prefixes are five characters long, these do not affect the SURE model file, so eight unique characters for a name guarantees uniqueness of the model.

- Redundancy Count $\longrightarrow$ Number of components in each subsystem of this type.
- Degradable $\longrightarrow$ Description of how this component degrades when component failures occur after any available spares have been exhausted. One of the following predefined degradation strategies must be chosen. The default is Non-degradable.
  - Non-degradable $\longrightarrow$ No components are ever removed from the subsystem.
  - Triplex to duplex to simplex $\longrightarrow$ Components are removed from the subsystem one at a time as they fail. Thus for a hexad, this selection would be hexad to quintad to quadruplex to triplex to duplex to simplex.
  - Triplex to simplex $\longrightarrow$ Components are removed from the subsystem one at a time as they fail until a triplex is reached. Failure of one component of a triplex results in removal of the failed component plus one other component, leaving a simplex. Thus for a hexad, this selection would be hexad to quintad to quadruplex to triplex to simplex.
  - Duplex to zero $\longrightarrow$ Components are removed from the subsystem one at a time until a duplex is reached. On the next component failure, all of the components are removed. Thus for a hexad, this selection would be hexad to quintad to quadruplex to triplex to duplex to zero. This selection is useful for self-checking pairs.

  The logic and mathematics for degradable components is discussed in detail in Section 5.2 on page 42.
- Transients? $\longrightarrow$ Select this box if you wish to model transient faults as well as permanent faults.
- Intermittents? $\longrightarrow$ Intermittents are not supported in Version 1.0.
- Dedicated Spare Count $\longrightarrow$ The number of spares dedicated to this subsystem only. Note that dedicated spares cannot be shared across replicated subsystems; they are dedicated to a specific subsystem. Dedicated spares will be substituted for failed components before pooled spares. The logic and mathematics for spares is discussed in detail in Section 5.1 on page 41.
- Primary Spare Pool $\longrightarrow$ If failed components of this subsystem can be replaced by spare components from a spare pool, the name of the first pool they will be taken from is listed here. Any dedicated spares for this subsystem will be used before the primary spare pool. The logic and mathematics for spares is discussed in detail in Section 5.1 on page 41.
- Secondary Spare Pool $\longrightarrow$ If the name of a secondary spare pool is given here, then after the subsystem has used all of its dedicated spares and primary pool spares, then subsequent failed components will be replaced with components from the secondary spare pool. The logic and mathematics for spares is discussed in detail in Section 5.1 on page 41.
- Tertiary Spare Pool $\longrightarrow$ This is the name of the third spare pool to be used after all dedicated, primary pool, and secondary pool components are exhausted. The logic and mathematics for spares is discussed in detail in Section 5.1 on page 41.

Since there is only one set of triplex processors in our example system, the repetition factor is 1. The component name of "processors" is entered. The redundancy count is 3 because there are 3 components in the "processor" subsystem. The subsystem always degrades by one, so degradation from "triplex to duplex to simplex" is selected. Permanent faults are always modeled, and for this subsystem modeling of transient faults is also selected. Since there is one spare processor, the dedicated spare count is 1. As soon as a non-zero dedicated spare count is chosen, another panel appears for describing the dedicated spares for that subsystem. The Dedicated Spare Details panel contains the following items:

- Detectability $\longrightarrow$ Whether the system can detect that spares have failed and remove them from the available spares is specified by choosing one of the following:
  - Never $\longrightarrow$ The probability of detection that a spare has failed is zero.
  - Partially $\longrightarrow$ If this selection is chosen, an additional panel immediately appears for the user to input the probability of detection, as shown in Figure 11.
  - Fully $\longrightarrow$ The probability of detection that a spare has failed is one; i.e. the system can determine with 100% success that a failed spare is bad and will not use it.
- Failure rate $\longrightarrow$ The failure rate for dedicated spares.


Component Configuration

repetition factor: 1

Component Name: processors

redundancy count: 3

Dedicated Spare Details

detectability: Never [✓ Partially] Fully

Failure rate

Probability of Detection

PROB 0.88887

Figure 11: Partially detectable dedicated spare failures

For our example system, the spare is assigned an exponential failure rate[4] of 6.113×10^{-4}, and "Fully" detectability is chosen because it is assumed for this system that failures of the spare are always immediately detected. The example system contains no pooled spares, so no spare pool names are specified.

After clicking on the "Okay" button, a panel appears for specifying the rates pertinent to this subsystem, as shown in Figure 12. The items that can appear in the panel are as follows:

- Permanent fault arrival $\longrightarrow$ The exponentially distributed arrival rate for permanent faults for this component type.

[4]All rates must be input in "e" exponential format. For example, 6.113×10^{-4} must be input as $6.113e-4$.

- Rate to remove and/or break up $\longrightarrow$ The rate to degrade the subsystem configuration as specified in the "Degradable" section of the Component Configuration panel once all spare components have been exhausted.
- Rate to reconfigure in a spare $\longrightarrow$ The rate to replace a failed component with either a dedicated spare or a pooled spare.
- Transient fault arrival $\longrightarrow$ The exponentially distributed arrival rate for transient faults for this component type.
- Transient fault disappearance rate $\longrightarrow$ The rate at which transient faults and their effects disappear from the system for this component type.

The user will only be prompted for the items that pertain to the subsystem specified; for example, the user will only be prompted for transient fault arrival and disappearance rates if he specified that transients faults were to be modeled. Permanent and transient fault arrival rates are slow exponential rates, and are defined by simply giving a rate in real or exponential notation, such as "1e-4". The rates for degradation, reconfiguration to bring in a spare, or disappearance of transient faults are fast rates. These fast rates may be given in one of two formats: 1) by giving the mean and standard deviation of the distribution within angle brackets, such as "$< 3.6e-3, 3e-3 >$", or 2) by simply giving the rate for an exponentially distributed transition, such as "6.5e5". If the rate is given, the mean and standard deviation used for this transition will both be equal to the inverse of this rate. Figure 12 shows the specification of rates for component "processors" of the example system.

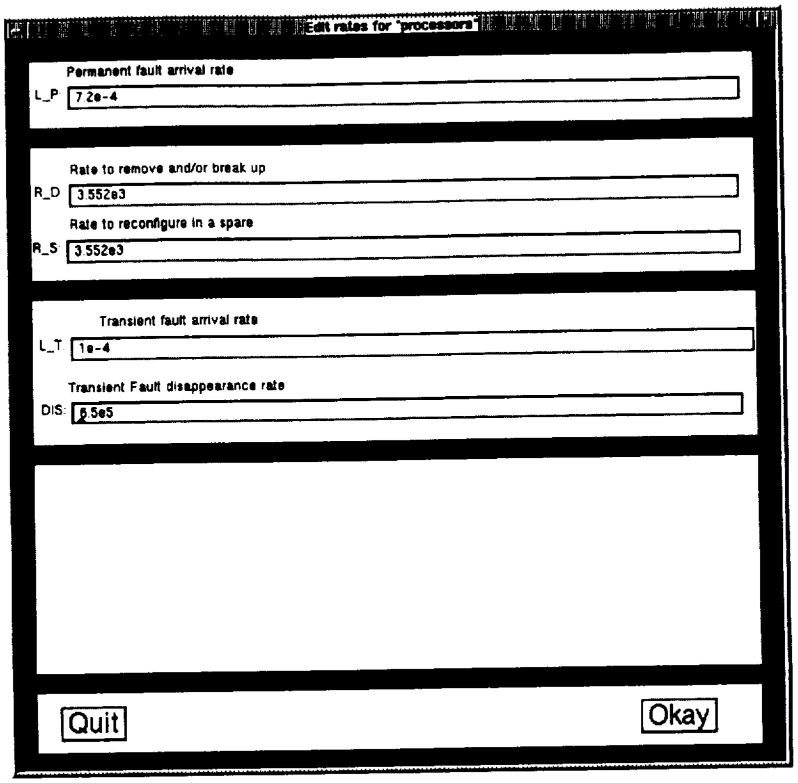
For any component type that can fail with transients, all reconfiguration rates must be input by giving rates for exponential distributions. This is necessary because TOTAL cannot model competing nonexponential transitions, and this requirement is enforced by the rates panel.

After clicking on the "Okay" button on the rates panel, the component specification menus will disappear and all changes will be reflected in the tables.

The memories and bus subsystems are specified in a similar manner. The memories subsystem description is identical to the components specification, except that the dedicated spare count is not set to 1. To specify that there are no dedicated spares, the user can just leave the dedicated spare count box empty (and TOTAL will use the default value of 0) or he can enter a 0 in the box. Since the subsystem has no dedicated spares, the menu for specifying these details does not appear.

The description for the bus subsystem is shown in Figure 13. Since the bus is a non-reconfigurable quadruplex, the redundancy count is 4 and "Non-degradable" is chosen. Transients are not selected since transient faults of the bus will not be modeled. The bus has no dedicated spares.

After clicking on the "Okay" button, the rate specification panel for the bus subsystem appears, as shown in Figure 14. Since transient bus faults will not be modeled and the bus has no spares and is non-degradable, the rate specification panel only prompts for the input of the permanent failure rate for this component.


Edit rates for "processors"

Permanent fault arrival rate

L_P: 7.2e-4

Rate to remove and/or break up

R_D: 3.552e3

Rate to reconfigure in a spare

R_S: 3.552e3

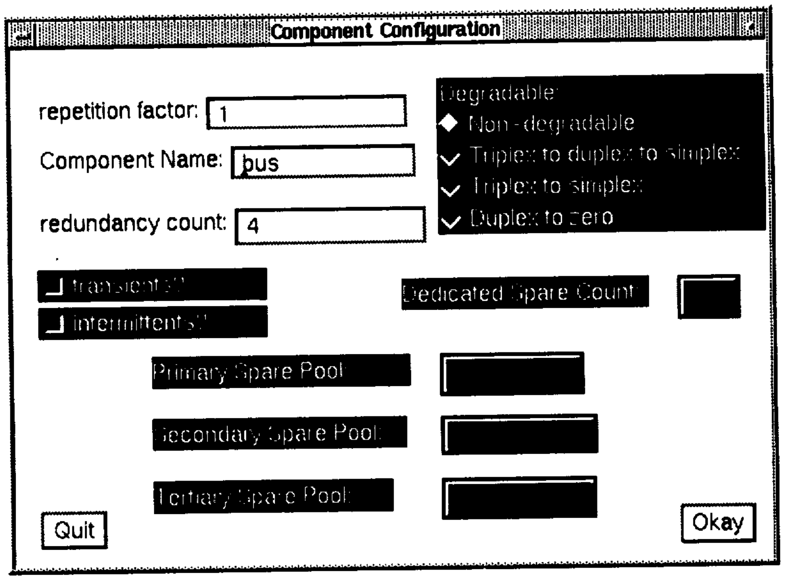
Transient fault arrival rate

L_T: 1e-4

Transient Fault disappearance rate

DIS: 6.5e5

Figure 12: Rates for component "processors"


Component Configuration

repetition factor: 1

Component Name: bus

redundancy count: 4

Degradable:
◆ Non-degradable
Triplex to duplex to simplex
Triplex to simplex
Duplex to zero

transient?

intermittent?

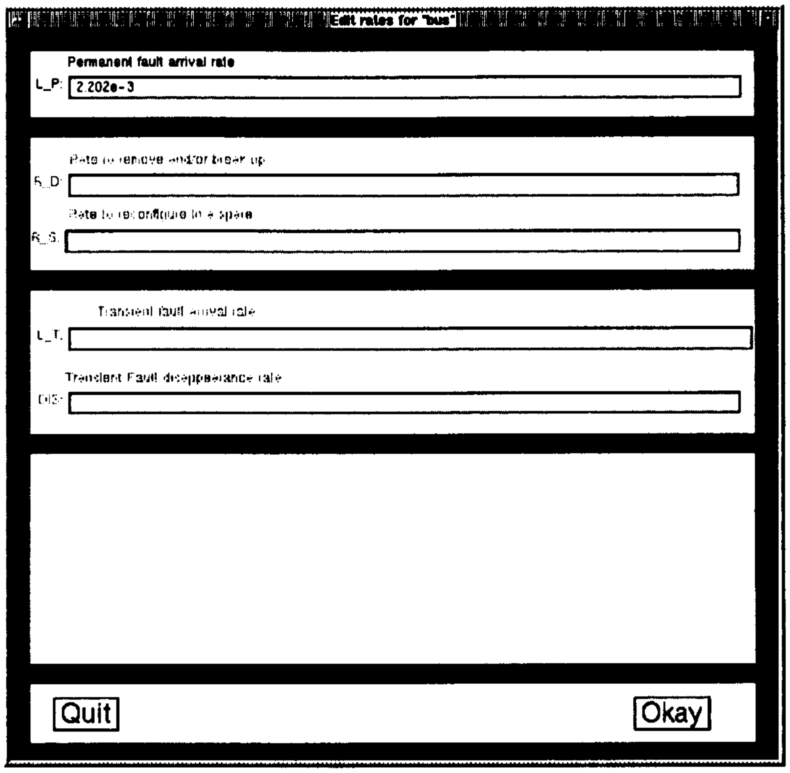
Dedicated Spare Count

Primary Spare Pool

Secondary Spare Pool

Tertiary Spare Pool

Figure 13: Component configuration for "bus"



Edit rates for "bus"

Permanent fault arrival rate

L_P: 2.202e-3

Rate to remove and/or break up

R_D:

Rate to reconfigure to a spare

R_S:

Transient fault arrival rate

L_T:

Transient Fault disappearance rate

DIS:

Figure 14: Component rates for "bus"

3.3 Dependencies

The Dependencies section allows the user to specify any dependencies in the failure behaviors between subsystems in the system. Each dependency has one or more triggering events and one or more effects. A trigger event might be permanent or transient failure of a component, replacement of a component with a spare, or removal of a component. An effect of a dependency might be the failure or removal of a component, or the incrementing or decrementing of a value in the model. As will be shown in a later section, the user can define extra state-space variables to be included in the model. The user can then define dependencies that cause the values of these variables to be changed. This can be useful for describing a system with various "modes" of operation.

Dependencies can be either conditional or unconditional. Conditional dependencies are only in effect under certain conditions of the system. Each dependency has exactly one cause and can have either a single effect or multiple effects. For example, if we were to model a system in which a processor was the master of a bus, then failure of that processor would cause (at least temporary) failure of the bus.

There is one type of dependency that must be described about the example system—because the memories are connected to the processors, removal of a faulty processor causes removal of the memory unit attached to it. The dependency between each memory unit and its processor must be specified; removal of processor 1 causes removal of memory 1, removal of processor 2 causes removal of memory 2, and removal of processor 3 causes removal of memory 3. These dependencies are unconditional, because they are always in effect regardless of the system configuration.

To describe the first dependency, select the "Append" option from the Edit menu next to the Dependencies table, and the Dependency Specification panel will appear. This panel contains the following items:

- Conditional/Unconditional Buttons $\longrightarrow$ Specifies whether this dependency is always in effect (unconditional) or only in effect under certain conditions or configurations (conditional). If Conditional is chosen, a menu appears for specifying the condition or set of conditions under which this dependency is in effect.
- Cause $\longrightarrow$ Specification of what conditions cause this dependent behavior to trigger. The entries for specifying the cause are as follows:
  - Component Name: $\longrightarrow$ Name of component type.
  - [,] $\longrightarrow$ Which component of this type triggers this dependency. Default is blank (unspecified), which means that any component of this type can trigger this dependency.
  - One of the following must be chosen to specify what behavior(s) of the specified component causes the dependency to trigger:
    * failure $\longrightarrow$ Triggers on arrival of any applicable failure of the component (permanent or transient).

        * arrival of permanent ⟶ Triggers on arrival of permanent failure of the component. Since arrival of permanent includes both working-to-permanent and transient-to-permanent transitions, this option can only be chosen if transient faults are modeled for this component.
        * arrival of transient ⟶ Triggers on arrival of transient failure of the component.
        * removal ⟶ Triggers on removal of the component from the configuration because of degradation. Replacement with a spare component is not considered removal of the component.
        * recovery ⟶ Triggers on replacement of the component with a dedicated or pooled spare regardless of whether that spare swapped in is working or has failed. Also triggers on degradation of the component by removal of a faulty component.
        * disappearance of transient ⟶ Triggers on disappearance of transient failure (and its effects) from the component.

- Effect X of Y ⟶ Listing of effect(s) of this dependency. The X is the index of the effect currently shown, and the Y is the total number of effects listed for this dependency. The entries for specifying each effect are as follows:
    - Component Name: ⟶ Name of component type or variable affected by this dependency.
    - [,] ⟶ Index to component of this type (or variable array) that is affected by this dependency. Default is blank (unspecified), which means that all components of this type are affected with equal probability by this dependency. An equal sign ("=") may also be used to indicate that the same index value is to be used as the index of the component that triggered the dependency.
    - If the effect is on a component (as opposed to a variable), one of the following must be chosen to specify the effects of this dependency on this component:
        * failure ⟶ Causes the component to fail permanently.
        * removal ⟶ Causes the component to be removed from the current configuration. The number active is always decreased with a removal effect, regardless of whether of the presence of any spares. If the user desires a spare to be swapped in, then a failure effect should be used instead. Transitions to replace failed processors with spares are automatically generated by TOTAL when spares are present in the system description.
        * transient disappearance ⟶ Causes the disappearance of any transient faults currently experienced by this component.
    - If the effect is on a variable (as opposed to a component), one of the following must be chosen to specify the effects of this dependency on this variable:
        * ++ ⟶ Causes the value of the specified variable to be incremented by one.
        * −− ⟶ Causes the value of the specified variable to be decremented by one.
        * =0(false) ⟶ Causes the specified variable to be set to 0 if integer or to false if Boolean.

    * =min ⟶ Causes the specified variable to be set to the minimum value allowed for the variable.
    * =max ⟶ Causes the specified variable to be set to the maximum value allowed for the variable.
    * =1(true) ⟶ Causes the specified variable to be set to 1 if integer or to true if Boolean.

The Dependencies panel also includes the following buttons:

- Quit ⟶ Quit entering or editing this dependency without making the specified changes
- Last Effect ⟶ Display the previous effect specified for this dependency (i.e. move from displaying Effect M of N to displaying Effect M-1 of N).
- Next Effect ⟶ Display the next effect of this dependency (i.e. move from displaying Effect M of N to displaying Effect M+1 of N).
- Edit ⟶ Menu for appending, copying or deleting effects of this dependency.

The panel for specifying the conditions under which a dependency takes effect has the following entries:

- Component Name ⟶ Name of component type.
- [,] ⟶ Which component of this type triggers this dependency. Default index is "=" or blank (not applicable), which means that any component of this type can trigger this dependency.
- If an index is given, then one or more of the following must be chosen to specify the condition of the specified component that must be effect for this dependency to occur:
    - failed ⟶ This dependency can only occur when the specified component has failed (either permanent or transient) but is still in the active configuration (has not been replaced by a working spare component or removed through degradation).[5]
    - permanent fault ⟶ This dependency can only occur when the specified component has failed permanently but is still in the active configuration (has not been replaced by a working spare component or removed through degradation).[6]
    - transient fault ⟶ This dependency can only occur when the specified component is experiencing a transient fault and is still in the active configuration (has not been replaced by a working spare component or removed through degradation).[7]
    - removed ⟶ This dependency can only occur when the specified component has been removed from the active configuration through degradation.[8]

[5] FA[ix]
[6] FP[ix]
[7] FT[ix]
[8] NOT A[ix]

- If an index is not given, then one of the following must be chosen to specify the condition of the specified component that must be effect for this dependency to occur:
  - majority vote failure $\longrightarrow$ This dependency can occur only when a majority of the active parts in the component have failed.[9]
  - exhaustion of parts $\longrightarrow$ This dependency can occur only when all of the parts in the component have failed.[10]
  - # working < n $\longrightarrow$ This dependency can occur only when the number of working parts in the component is less than "n".[11]
  - # removed > n $\longrightarrow$ This dependency can occur only when the number of parts removed from the component is greater than "n". The number of parts removed from a component is defined to be the difference between the number initially and the total active which remain in the configuration.[12]
  - # failed > n $\longrightarrow$ This dependency can occur only when the number of failed parts still active for the component is greater than "n".[13]
  - # permanent failed > n $\longrightarrow$ This dependency can occur only when the number of permanent failed parts still active for the component is greater than "n".[14]
  - # transient failed > n $\longrightarrow$ This dependency can occur only when the number of transient failed parts still active for the component is greater than "n".[15]
  - textual $\longrightarrow$ This dependency can occur only when the textual ASSIST syntax Boolean expression holds true. The syntax of the expression is not checked for validity, however, TOTAL does enforce balancing of parentheses within the expression.
- n $\longrightarrow$ If one of the above bullets which reference a value for "n" is selected, then the value to be used for "n" must be entered here.
- negate this one $\longrightarrow$ Causes the negation of this condition to be used, e.g. changes "# working < n" to "not(# working < n)."
- negate whole IF condition $\longrightarrow$ Causes the negation of the entire IF. (e.g., "not ((# working < n) and (transient fault))").

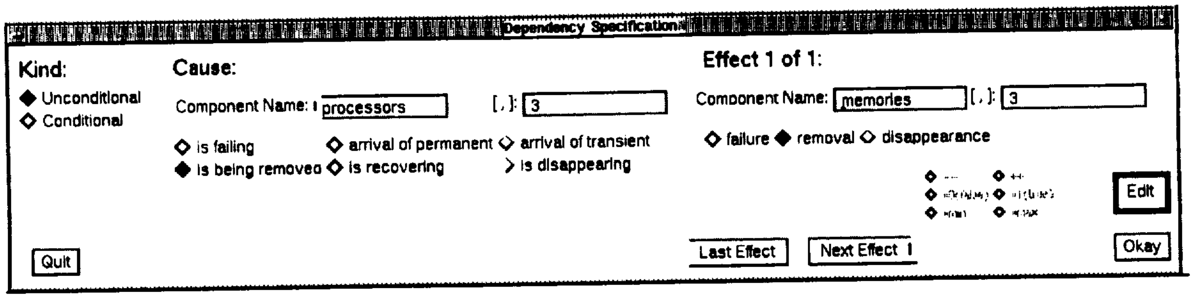
The Conditionals panel also includes the following buttons:

- Last Condition $\longrightarrow$ Display the previous condition specified for this dependency (i.e. move from displaying Conditional M of N to displaying Conditional M-1 of N).

[9] TFA >= TWA
[10] TWA <= 0
[11] NW < n
[12] (NI-TA) > n
[13] NF > n
[14] NFP > n
[15] NFT > n

- Next Condition ⟶ Display the next condition of this dependency (i.e. move from displaying Conditional M of N to displaying Conditional M+1 of N).
- Edit ⟶ Menu for appending, copying or deleting conditions of this dependency.

For the example system, the "Kind" of dependency is specified as unconditional, because it is always in effect regardless of the system configuration. If we were describing a dependency that was only in effect when certain system conditions were true, we would select conditional, and a menu would appear for describing the system conditions that must be true for this dependency to be in effect. The dependency specification panel to show that removal of processor 3 causes removal of memory 3 is shown in Figure 15.


Dependency Specification

Kind: [✓ Unconditional] Conditional

Cause: Component Name: processors [,]: 3

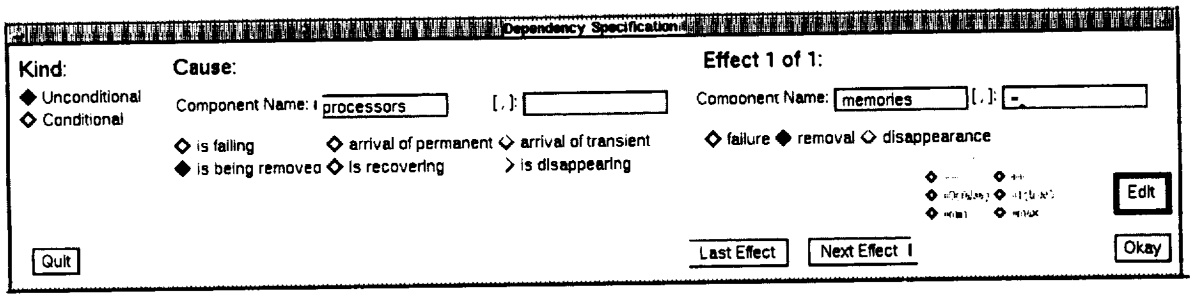
is failing, arrival of permanent, arrival of transient, [✓ is being removed], is recovering, is disappearing

Effect 1 of 1: Component Name: memories [,]: 3

failure [✓ removal] disappearance

Figure 15: Specification of third dependency

Since the second dependency is similar to the first, we will use the Copy option of the Dependency Edit menu to specify the second dependency. When the Dependency Specification panel appears, it already has the information specified for the first dependency, and we have only to change the two 1's to 2's and select the "Okay" button to input this dependency. Similarly, the third dependency can be input as a modification of the second.


Dependency Specification

Kind: [✓ Unconditional] Conditional

Cause: Component Name: processors [,]:

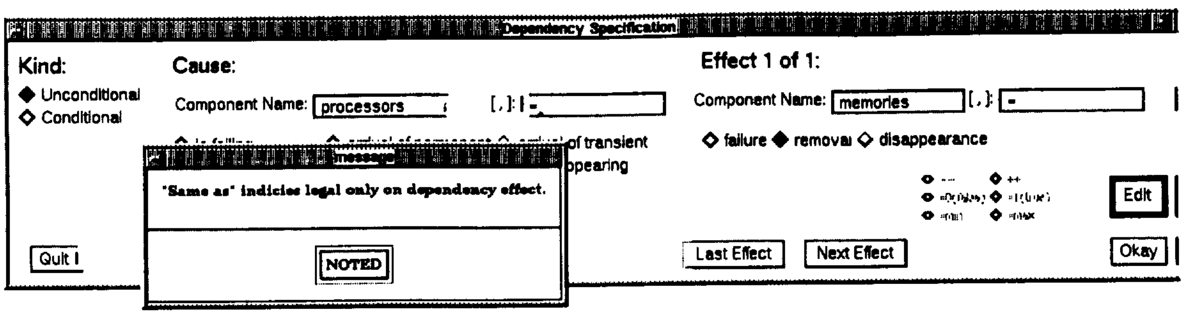
is failing, arrival of permanent, arrival of transient, [✓ is being removed], is recovering, is disappearing

Effect 1 of 1: Component Name: memories [,]: =

failure [✓ removal] disappearance

Figure 16: Concise Specification using "=" to denote "same index"

Alternately, we could have input our example system with a single dependency using the "=" in the effect to denote "use the same index" as used in the cause. This is illustrated in Figure 16. Note that the index on the cause is left blank unless a specific index value is given. The "=" is illegal in the cause as illustrated in Figure 17.


Dependency Specification

Kind: ◆ Unconditional ◇ Conditional

Cause: Component Name: processors [,]: =

Effect 1 of 1: Component Name: memories [,]: =

◇ failure ◆ removal ◇ disappearance

message

"Same as" indicies legal only on dependency effect.

Figure 17: Index of "=" Belongs only on Effect

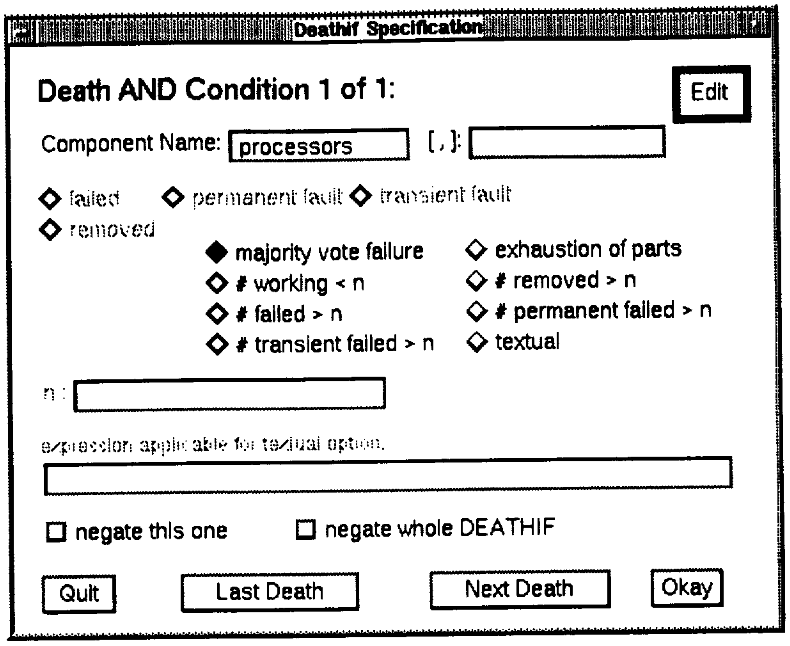
3.4 System Failure Conditions

The conditions leading to system failure are defined in terms of functions of the states of the components in the system, for example, majority vote failure of subsystem A or exhaustion of parts in subsystem B. The menu interface presents a list of functions to choose from, or the user may elect to input a textual description. The interface also allows the user to define system failure conditions that are functions of several conditions; for example, a given system may fail if both the primary memory fails a majority vote and the backup memory has failed. Using the menu-driven interface, the user creates a list of independent conditions, any one of which will cause system failure. In the table, each system failure condition in TOTAL is referred to as a "DEATHIF" condition, named for the ASSIST statement of the same name.

For the example system, we will define three system failure conditions: majority vote failure of processors, majority vote failure of memories, and majority vote failure of the bus components. The three conditions are independent in the sense that system failure occurs if any one of the three conditions holds. To describe the first condition, select the Append option from the Edit menu beside the System Failure Conditions table, and the System Failure Description panel will appear. Figure 18 shows the System Failure Description panel filled in with the correct information to describe majority vote failure of the processor subsystem.

The System Failure Descriptions panel contains the following fields:

- Component Name ⟶ Name of component type.
- [,] ⟶ Which component of this type triggers this death specification. Default index is "=" or blank (not applicable), which means that any component of this type can trigger this death specification.
- If an index is given, then one or more of the following must be chosen to specify the condition of the specified component that must be effect for this death specification to occur:
  - failed ⟶ This death specification can only occur when the specified component has failed (either permanent or transient) but is still in the active configuration (has not



Deathif Specification

Death AND Condition 1 of 1:

Component Name: processors [,]:

◇ failed ◇ permanent fault ◇ transient fault ◇ removed

[✓ majority vote failure] ◇ exhaustion of parts ◇ # working < n ◇ # removed > n ◇ # failed > n ◇ # permanent failed > n ◇ # transient failed > n ◇ textual

n :

expression applicable for textual option.

☐ negate this one ☐ negate whole DEATHIF

Figure 18: System Failure Description for Majority Vote Failure of Processors

been replaced by a working spare component or removed through degradation).[16]

- permanent fault ⟶ This death specification can only occur when the specified component has failed permanently but is still in the active configuration (has not been replaced by a working spare component or removed through degradation).[17]
- transient fault ⟶ This death specification can only occur when the specified component is experiencing a transient fault and is still in the active configuration (has not been replaced by a working spare component or removed through degradation).[18]
- removed ⟶ This death specification can only occur when the specified component has been removed from the active configuration through degradation.[19]

- If an index is not given, then one of the following must be chosen to specify the condition of the specified component that must be effect for this death specification to occur:
  - majority vote failure ⟶ This death specification can occur only when a majority of the active parts in the component have failed.[20]
  - exhaustion of parts ⟶ This death specification can occur only when all of the parts in the component have failed.[21]

[16]FA[ix]
[17]FP[ix]
[18]FT[ix]
[19]NOT A[ix]
[20]TFA >= TWA
[21]TWA <= 0

  - # working $<$ n $\longrightarrow$ This death specification can occur only when the number of working parts in the component is less than "n".[22]
  - # removed $>$ n $\longrightarrow$ This death specification can occur only when the number of parts removed from the component is greater than "n". The number of parts removed from a component is defined to be the difference between the number initially and the total active which remain in the configuration.[23]
  - # failed $>$ n $\longrightarrow$ This death specification can occur only when the number of failed parts still active for the component is greater than "n".[24]
  - # permanent failed $>$ n $\longrightarrow$ This death specification can occur only when the number of permanent failed parts still active for the component is greater than "n".[25]
  - # transient failed $>$ n $\longrightarrow$ This death specification can occur only when the number of transient failed parts still active for the component is greater than "n".[26]
  - textual $\longrightarrow$ This death specification can occur only when the textual ASSIST syntax Boolean expression holds true. The syntax of the expression is not checked for validity, however, TOTAL does enforce balancing of parentheses within the expression.

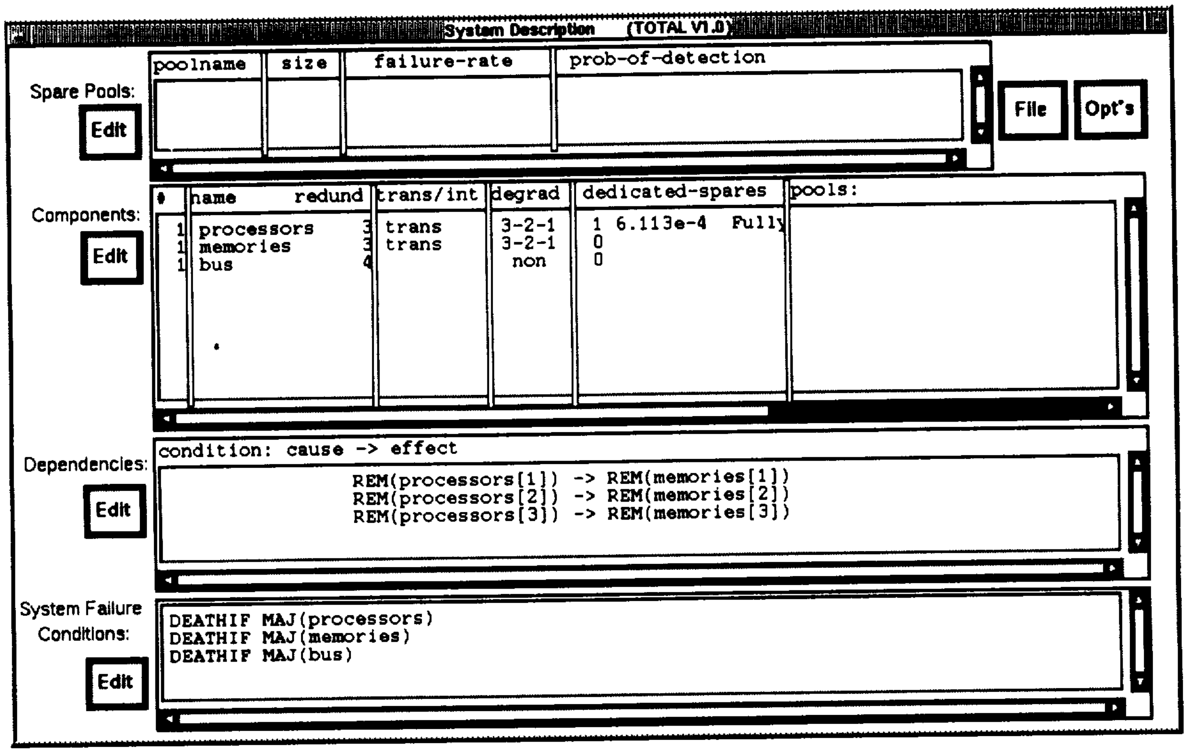
- n $\longrightarrow$ If one of the above bullets which reference a value for "n" is selected, then the value to be used for "n" must be entered here.
- negate this one $\longrightarrow$ Causes the negation of this condition to be used, e.g. changes "# working < n" to "not(# working < n)."
- negate whole DEATHIF $\longrightarrow$ Causes the negation of the entire IF. (e.g., "not ((# working < n) and (transient fault))").

Each panel holds one condition description. The user may conjunctively link together as many conditions as he wishes to form a complex system failure specification. This is accomplished using the Edit menu located on this panel. The user may view the multiple conditions that make up a single complex specification by using the "Last Death" and "Next Death" buttons to move through the specification. Note—Individual DEATHIFs are "OR'd" together and are specified using the Edit menu on the System Description panel. The multiple conditions within a single DEATHIF are "And'd" together and are specified using the Edit menu located on the Deathif Specification panel.

To specify that majority vote failure of the processors is assumed to cause system failure, specify processors for the component name, select majority vote failure, and select the "Okay" button. System failures due to memories or buses failing majority votes are specified in a similar manner using either the Append or the Modify option of the Edit menu on the System Description panel.

The complete system description is shown in Figure 19.

[22] NW $<$ n
[23] (NI-TA) $>$ n
[24] NF $>$ n
[25] NFP $>$ n
[26] NFT $>$ n



```
poolname | size | failure-rate | prob-of-detection
```

```
#  name        redund  trans/int  degrad  dedicated-spares          pools:
1  processors  3       trans      3-2-1   1 6.113e-4  Fully
1  memories    3       trans      3-2-1   0
1  bus         4                  non     0
```

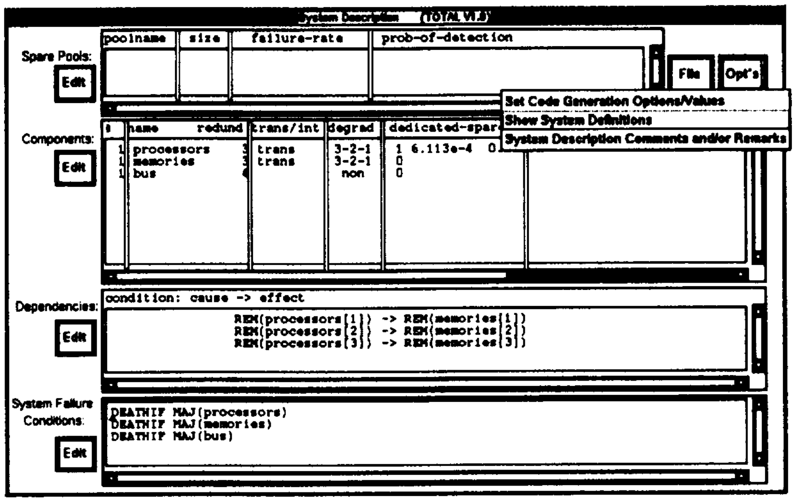
```
condition: cause -> effect
                    REM(processors[1]) -> REM(memories[1])
                    REM(processors[2]) -> REM(memories[2])
                    REM(processors[3]) -> REM(memories[3])
```

```
DEATHIF MAJ(processors)
DEATHIF MAJ(memories)
DEATHIF MAJ(bus)
```

Figure 19: A sample system modeled with TOTAL

3.5 Display of System Definitions

The user can elect to see the System Definitions that will be written to the ASSIST output file in an additional panel, which appears at the bottom of the screen so as not to obscure the System Definitions panel. To bring up the additional panel, select the option to "Show System Definitions" from the "Opt's" options menu, as shown in Figure 20.


System Description (TOTAL V1.0)
Spare Pools:
Edit
Components:
Edit
Dependencies:
Edit
System Failure Conditions:
Edit
File
Opt's
Set Code Generation Options/Values
Show System Definitions
System Description Comments and/or Remarks

poolname	size	failure-rate	prob-of-detection

#	name	redund	trans/int	degrad	dedicated-spar
1	processors	3	trans	3-2-1	1 6.113e-4 0.
1	memories	3	trans	3-2-1	0
1	bus			non	0

```
condition: cause -> effect
                    REM(processors[1]) -> REM(memories[1])
                    REM(processors[2]) -> REM(memories[2])
                    REM(processors[3]) -> REM(memories[3])
```

```
DEATHIF MAJ(processors)
DEATHIF MAJ(memories)
DEATHIF MAJ(bus)
```

Figure 20: Using pull-down menu to show system definitions

After the panel appears on the screen, the screen will appear as shown in Figure 21.

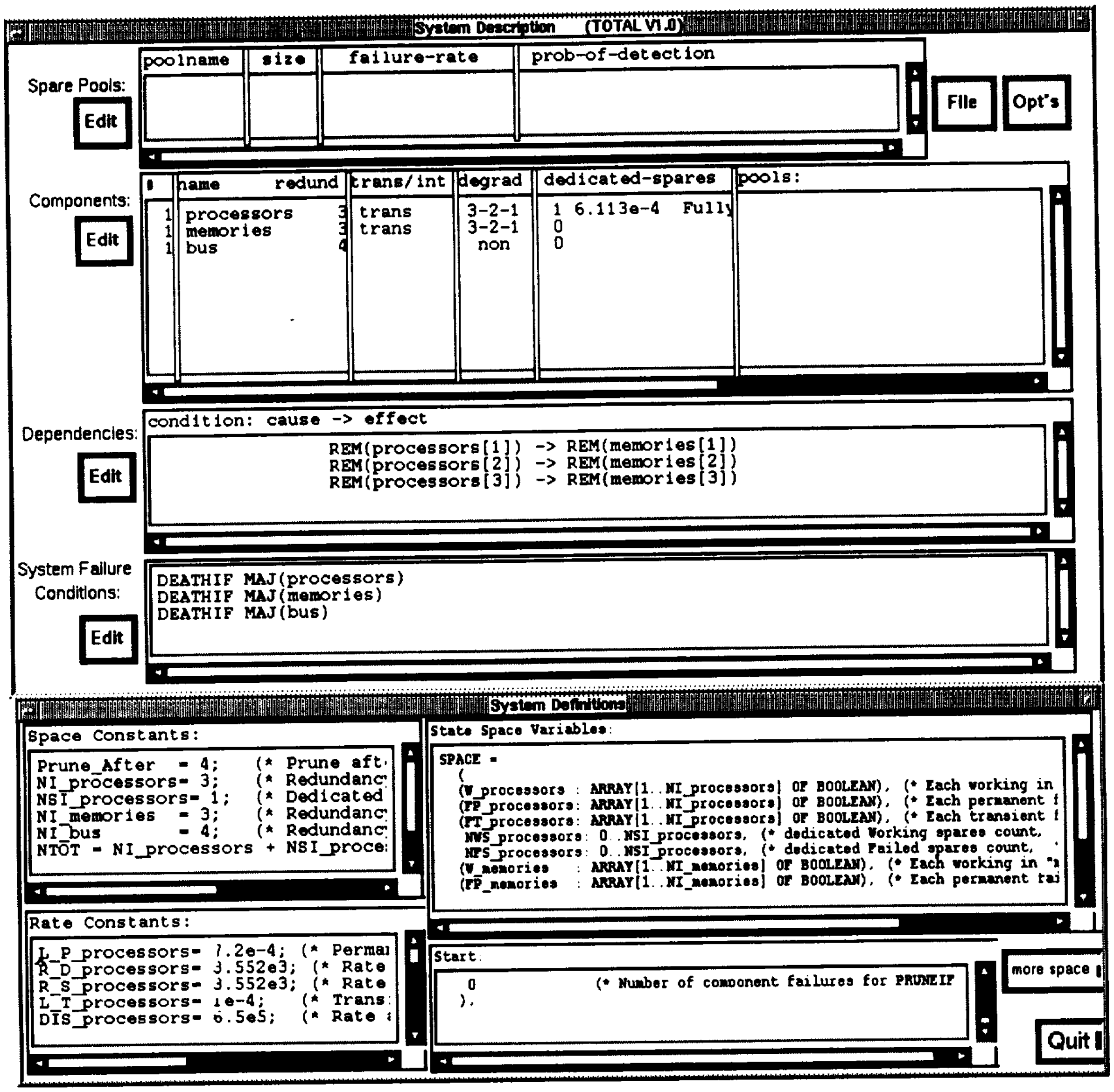
3.5.1 Adding auxiliary state-space variables

The user can define additional state-space variables and can change their values as effects of dependencies. User-defined state-space variables can be used in system failure conditions or in triggers or conditions of dependencies. To add user-defined state-space variables, the Edit menu on the System Definitions panel is used.

Figure 22 shows how the user would enter an additional state-space variable named "test", which is an array of `Boolean's`. The optional identifier applies when there is either an array range or a range of values or both. It does not apply for `Boolean` scalars (non-arrays) selected. If the array were, for example, to contain an array of 1..24, then the user could specify a name for the upper bound of 24. The initial value is used in the corresponding `START` statement in the generated `ASSIST` output file.

3.6 Writing Comments and/or Remarks to describe a system

The user may describe a system by selecting the option "System Description Comments and/or Remarks" from the "Opt's" options menu, as shown in Figure 23. Some sample remark input



poolname	size	failure-rate	prob-of-detection

#	name	redund	trans/int	degrad	dedicated-spares	pools:
1	processors	3	trans	3-2-1	1 6.113e-4 Fully	
1	memories	3	trans	3-2-1	0	
1	bus	4		non	0	

```
condition: cause -> effect
                    REM(processors[1]) -> REM(memories[1])
                    REM(processors[2]) -> REM(memories[2])
                    REM(processors[3]) -> REM(memories[3])
```

```
DEATHIF MAJ(processors)
DEATHIF MAJ(memories)
DEATHIF MAJ(bus)
```

Space Constants:

```
Prune_After   = 4;    (* Prune aft
NI_processors= 3;     (* Redundanc
NSI_processors= 1;    (* Dedicated
NI_memories   = 3;    (* Redundanc
NI_bus        = 4;    (* Redundanc
NTOT = NI_processors + NSI_proce
```

Rate Constants:

```
L_P_processors= 7.2e-4; (* Perma
R_D_processors= 3.552e3; (* Rate
R_S_processors= 3.552e3; (* Rate
L_T_processors= 1e-4;    (* Trans
DIS_processors= 6.5e5;   (* Rate
```

State Space Variables:

```
SPACE =
  (
  (W_processors : ARRAY[1..NI_processors] OF BOOLEAN), (* Each working in
  (FP_processors: ARRAY[1..NI_processors] OF BOOLEAN), (* Each permanent f
  (FT_processors: ARRAY[1..NI_processors] OF BOOLEAN), (* Each transient f
   NWS_processors: 0..NSI_processors, (* dedicated Working spares count,
   NFS_processors: 0..NSI_processors, (* dedicated Failed spares count,
  (W_memories   : ARRAY[1..NI_memories] OF BOOLEAN), (* Each working in "
  (FP_memories  : ARRAY[1..NI_memories] OF BOOLEAN), (* Each permanent fa
```

Start:

```
  0              (* Number of component failures for PRUNEIF
  ),
```

Figure 21: Workstation with both System Description and Definitions panels present

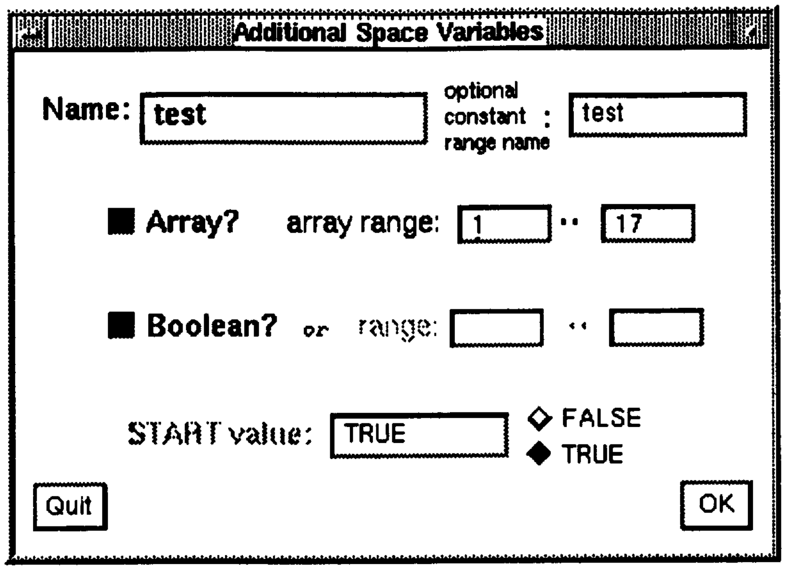
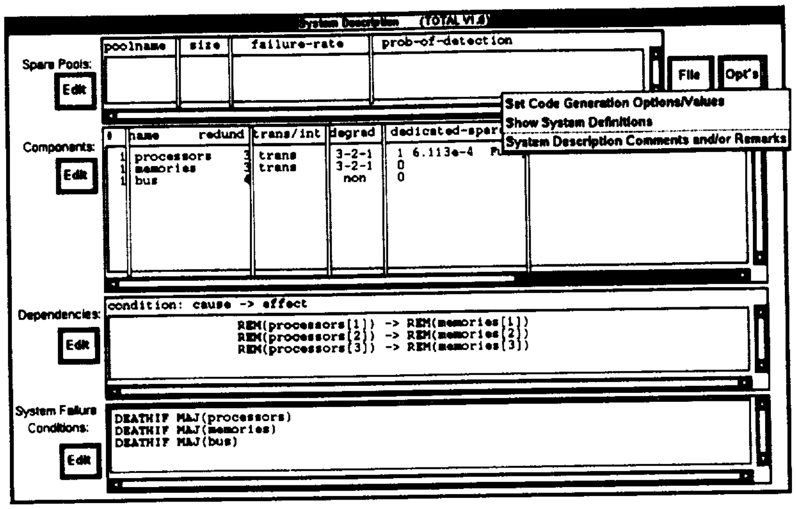

Additional Space Variables
Name: test
optional constant range name : test
■ Array? array range: 1 .. 17
■ Boolean? or range: ..
START value: TRUE
◇ FALSE
◆ TRUE
Quit
OK

Figure 22: Defining an extra state-space variable

is shown in Figure 24.


System Description (TOTAL V1.0)
Spare Pools:
Edit
Components:
Edit
Dependencies:
Edit
System Failure Conditions:
Edit
File
Opt's
Set Code Generation Options/Values
Show System Definitions
System Description Comments and/or Remarks

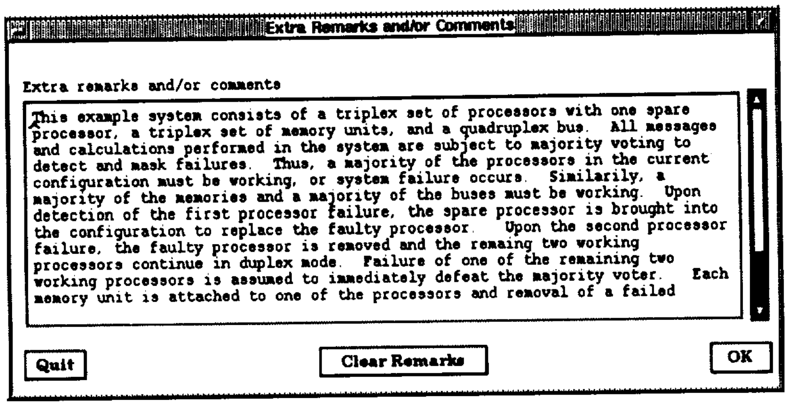
poolname	size	failure-rate	prob-of-detection

#	name	redund	trans/int	degrad	dedicated-spare
1	processors	3	trans	3-2-1	1 6.113e-4 F
1	memories	3	trans	3-2-1	0
1	bus	4		non	0

```
condition: cause -> effect
                REM(processors[1]) -> REM(memories[1])
                REM(processors[2]) -> REM(memories[2])
                REM(processors[3]) -> REM(memories[3])
```

```
DEATHIF MAJ(processors)
DEATHIF MAJ(memories)
DEATHIF MAJ(bus)
```

Figure 23: Using pull-down menu to specify comments and/or remarks



Extra remarks and/or comments

```
This example system consists of a triplex set of processors with one spare
processor, a triplex set of memory units, and a quadruplex bus.  All messages
and calculations performed in the system are subject to majority voting to
detect and mask failures.  Thus, a majority of the processors in the current
configuration must be working, or system failure occurs.  Similarily, a
majority of the memories and a majority of the buses must be working.  Upon
detection of the first processor failure, the spare processor is brought into
the configuration to replace the faulty processor.   Upon the second processor
failure, the faulty processor is removed and the remaing two working
processors continue in duplex mode.  Failure of one of the remaining two
working processors is assumed to immediately defeat the majority voter.   Each
memory unit is attached to one of the processors and removal of a failed
```

Figure 24: Input of extra remarks/comments

4 Generating and Solving the Reliability Model

Once the system description process is complete, the user can generate the ASSIST input description for his system and even execute the ASSIST and SURE computer programs directly from the TOTAL menu-driven interface. Several model-reduction techniques, such as pruning and trimming, are available through the TOTAL interface[6, 7].

4.1 Algorithm for Generating ASSIST-Language Description

As shown in the preceding section, the TOTAL menu-driven interface presents the user with a set of choices for describing the characteristics of his systems. The TOTAL program generates an ASSIST-level description of the system by executing a set of algorithms based on the system description information. The ASSIST language description is made up of the following basic elements: 1) a set of state-space variables, which are used to describe the states of the model; 2) a description of the initial state of the model; 3) a set of transition-description statements that define the legal transitions between states in the model; 4) a set of system failure conditions for the model; and 5) a set of constants and variable definitions that are used in the transition-description and system failure statements. The basic algorithm used by TOTAL for generating an ASSIST-level description of the system is as follows.

First, the TOTAL program generates the state-space variables, constants, and variable definitions needed to represent the behavior of the components in the system. As an example, the "processors" subsystem of the example system described above is degradable and has a spare that can fail, and both permanent and transient faults can be modeled. Thus, the ASSIST description would need state-space variables to represent the number of working processors in the active configuration, the number that have failed permanently, the number that have failed due to a transient fault, and the number of working and failed dedicated spare processors. Several constants would be defined for the "processors" components; for example, permanent and transient failure and recovery rates.

The next step is to generate transition-description statements to represent the failure behavior for each component in the system. Several transition-description statements are needed to describe each of the component types in the system. For example, the "processors" components can fail permanently or with a transient fault, the transient faults can disappear, failed processors can be replaced with the dedicated spare, and once the dedicated spares are exhausted, the subsystem recovers from failures by degradation. Each of these behaviors is captured in a separate statement. The dependencies must also be included in these statements. For example, the statement that represents removal of processor 1 must also remove memory 1 from the active configuration.

Finally, the system failure conditions for the model are described in the ASSIST language. This simple process consists of replacing each function in the TOTAL description with its underlying formula in ASSIST. For example, the majority vote function name used in TOTAL is replaced with a formula checking whether the number of working processors in the active configuration is greater than the number of failed processors in the active configuration.

As shown in the above description, the process of generating the ASSIST description depends on a set of algorithms for converting the information from the system description tables into an appropriate ASSIST representation. The correctness of this process lies in the generality of the individual algorithms. In other words, the algorithm for generating ASSIST transition-description statements to reflect replacing a failed component with a spare must work correctly for all possible combinations of characteristics that are allowed in the TOTAL system description. It must take into account all possible combinations of dedicated and pooled spares, spare failure rates and detectability, and all possible dependencies that might be involved. This can be especially difficult in the presence of conditional dependencies.

The approach adopted for the development of TOTAL has been to start with a limited set of choices and to carefully analyze the generality of each added feature and its accompanying algorithms. By doing so, we hope to preserve the correctness of the model generation process. We also plan to document the algorithms used so that the user will know exactly what assumptions are in the model and so that the algorithms can be independently checked for accuracy and generalism. Of course, any computer program will contain bugs, so users are cautioned to check the ASSIST model description for reasonableness and accuracy. Also, analysis of the generality of algorithms will grow increasingly difficult as the complexity of the system description choices grows. The example used in this paper was relatively simple, yet the ASSIST file generated by TOTAL was 692 lines long, including comments. As the number of components and the complexity of failure behaviors grows, reliability model descriptions tend to increase combinatorially.

The TOTAL program is significantly limited in terms of the characteristics of systems it can model accurately; however, the user will know when he has reached those limitations because the menu interface will not include those characteristics.

4.2 Naming Conventions Used During Generation

When TOTAL generates an ASSIST input model file, it generates names with a prefix and a suffix. The prefix always ends with an underscore character("_"). The suffix is always the name of the component or spare pool in question.

The prefixes used are shown in Tables 1 and 2. Only those prefixes that are applicable to the component type are defined; however, they are defined whether or not they are referenced. For example, if transients are not being modeled, then none of the prefixes that apply only to transients will appear. If, however, transients are modeled, then all of the prefixes that apply to transients will be defined regardless of whether or not they are referenced. This is because the user may wish to reference them directly by name in textual DEATHIF and textual dependency condition tests.

Some of the other variables and/or constants that are defined are listed in Table 3;

prefix	meaning
NI_	The number of parts initially active in a given component.
NSI_	The number of dedicated spares initially active in a given component or the number of spares initially active in a given spare pool.
TSI_	The total number of spares (both dedicated and pooled) initially available for use by a given component.
TFA_	Count of Total Failed Active for a given component.
TWA_	Count of Total Working Active for a given component.
NF_	Count of Number Failed Active for a given component.
FA_	Failed Active for each part of a given component.
A_	Active for each part part of a given component.
TA_	Count of total Active for a given part of a component.
TS_	Count of total number of spares available for use by a given component. This includes both dedicated and pooled spares. Do not confuse NS_ and TS_ for a component.
NA_	Count of the total number active in a component.
NW_	Count of the total number active and working in a component.
NFP_	Count of the total number active yet failed permanently in a component.
NFT_	Count of the total number active yet failed transiently in a component.
NWOT_	Count of the number of active working or failed transiently which have not failed permanently.
WOT_	Indicates whether active and either working or failed transiently for each part of a given component.
NFS_	Count of the number of failed hot or warm spares either in a spare pool or in the dedicated spares for a component.
NWS_	Count of the number of working hot or warm spares either in a spare pool or in the dedicated spares for a component.
NFD_	Count of the number of failed hot or warm spares which have been detected as having failed either in a spare pool or in the dedicated spares for a component.
NFU_	Count of the number of failed hot or warm spares which have not been detected as having failed either in a spare pool or in the dedicated spares for a component.

Table 1: Table of prefixes used in generated ASSIST file (Part I)

prefix	meaning
NUC_	Count of the number of hot or warm spares which have not been detected as having failed either in a spare pool or in the dedicated spares for a component. This is effectively the number of spares under consideration for the spare pool or the dedicated component spares in question.
TUC_	Count of the total number of spares including both dedicated and spared which have not been detected as having failed for a component. This is effectively the number of spares under consideration for the component in question.
NS_	Count of the number of spares either in a spare pool or in the dedicated spares for a component. Do not confuse NS_ and TS_ for a component.
PRF_	Probability that a given spare has failed in either a spare pool or in the dedicated spares for a component.
PRW_	Probability that a given spare is still working in either a spare pool or in the dedicated spares for a component.

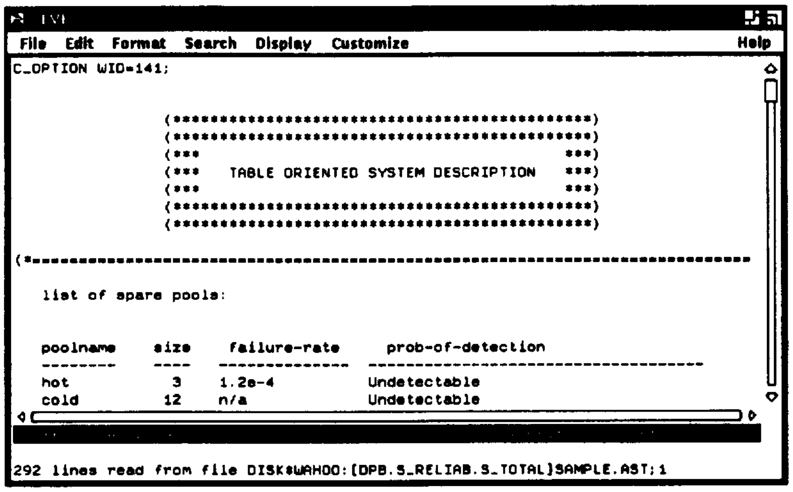
Table 2: Table of prefixes used in generated ASSIST file (Part II)

identifier	meaning
NCF	The number of component failures
NCFMX	The maximum possible component failure count. This is used as the upper bound of the range of values for NCF.
NTOT	The total number of parts in the system including all components, all dedicated spares, and all spare pools. This is used to make assertions that all parts be accounted for. This does not include any auxiliary state-space variables.
Prune_After	The number of component failures after which pruning should take place
TRASH_CAN_SIZE	The number of parts which have been removed from the system, hence the number of parts which have been discarded or "trashed". This is used to make assertions that all parts be accounted for. This does not include any auxiliary state-space variables.

Table 3: Table of miscellaneous identifiers used in generated ASSIST files

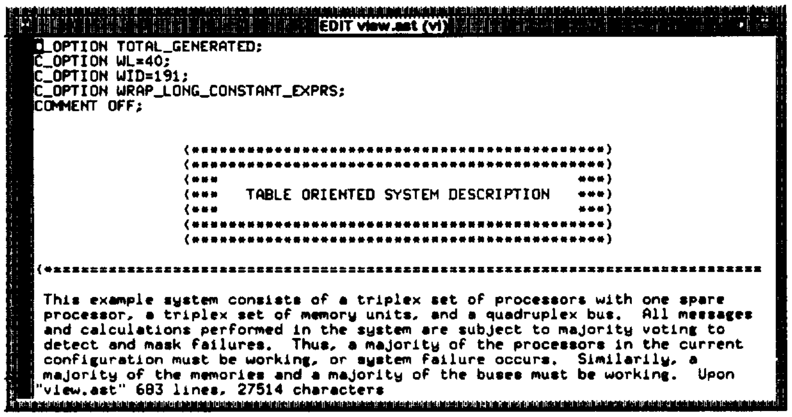
4.3 Using the Menu-Driven Interface to Generate and Edit the ASSIST-Language Description

To generate the ASSIST output file, select the option "Generate ASSIST" from the "File" options menu. The ASSIST output file generated for our example system is shown in Appendix D. After generating the ASSIST output file, the system editor can be used to view and perhaps even edit the file before running it. To do this, select the option "Edit ASSIST" from the "File" options menu. A new window will appear with the editor session as illustrated in Figure 25 taken from a VMS session.



```
C_OPTION WID=141;


                (********************************************)
                (********************************************)
                (***                                      ***)
                (***   TABLE ORIENTED SYSTEM DESCRIPTION  ***)
                (***                                      ***)
                (********************************************)
                (********************************************)

(*---------------------------------------------------------------------------

   list of spare pools:


   poolname     size    failure-rate     prob-of-detection
   --------     ----    -------------   -------------------------------------
   hot            3     1.2e-4          Undetectable
   cold          12     n/a             Undetectable

292 lines read from file DISK$WAHOO:[DPB.S_RELIAB.S_TOTAL]SAMPLE.AST;1
```

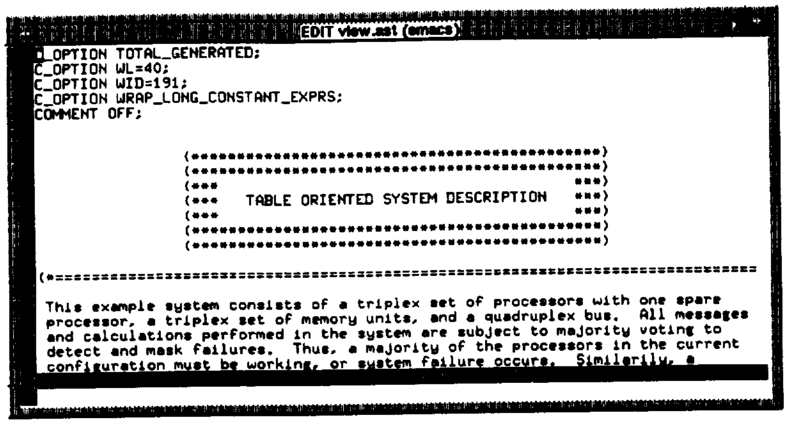
Figure 25: Editing the Generated ASSIST Code Under VMS



```
C_OPTION TOTAL_GENERATED;
C_OPTION WL=40;
C_OPTION WID=191;
C_OPTION WRAP_LONG_CONSTANT_EXPRS;
COMMENT OFF;


                (********************************************)
                (********************************************)
                (***                                      ***)
                (***   TABLE ORIENTED SYSTEM DESCRIPTION  ***)
                (***                                      ***)
                (********************************************)
                (********************************************)

(*===========================================================================

 This example system consists of a triplex set of processors with one spare
 processor, a triplex set of memory units, and a quadruplex bus.  All messages
 and calculations performed in the system are subject to majority voting to
 detect and mask failures.  Thus, a majority of the processors in the current
 configuration must be working, or system failure occurs.  Similarily, a
 majority of the memories and a majority of the buses must be working.  Upon
"view.ast" 683 lines, 27514 characters
```

Figure 26: Editing the Generated ASSIST Code Under UNIX

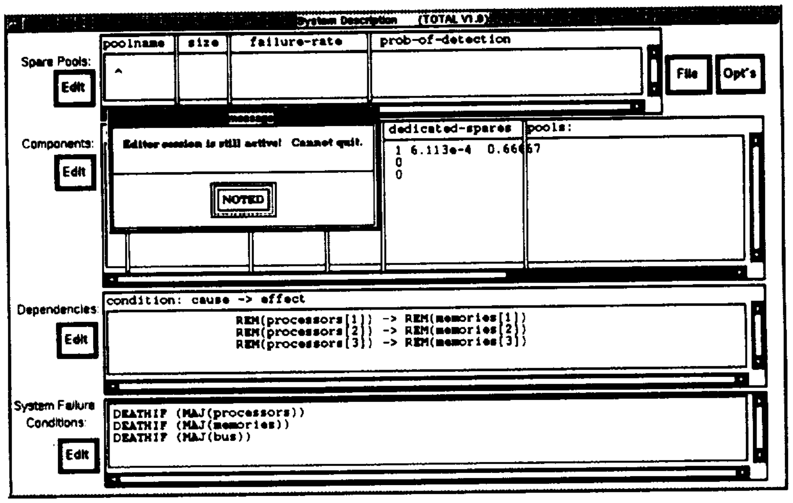
When the user is in the middle of an edit session, the TOTAL program will not allow the user



```
EDIT view.ast (emacs)
C_OPTION TOTAL_GENERATED;
C_OPTION WL=40;
C_OPTION WID=191;
C_OPTION WRAP_LONG_CONSTANT_EXPRS;
COMMENT OFF;

                    (*******************************************)
                    (*******************************************)
                    (***                                     ***)
                    (***   TABLE ORIENTED SYSTEM DESCRIPTION ***)
                    (***                                     ***)
                    (*******************************************)
                    (*******************************************)

(*=========================================================================

 This example system consists of a triplex set of processors with one spare
 processor, a triplex set of memory units, and a quadruplex bus.  All messages
 and calculations performed in the system are subject to majority voting to
 detect and mask failures.  Thus, a majority of the processors in the current
 configuration must be working, or system failure occurs.  Similarily, a
```

Figure 27: Editing the Generated ASSIST Code with EMACS

to exit. Figure 28 shows the system reminding the user to first quit the editor session before quitting the TOTAL session.



Editor session is still active! Cannot quit.

```
1 6.113e-4  0.66667
0
0
```

```
REM(processors[1]) -> REM(memories[1])
REM(processors[2]) -> REM(memories[2])
REM(processors[3]) -> REM(memories[3])
```

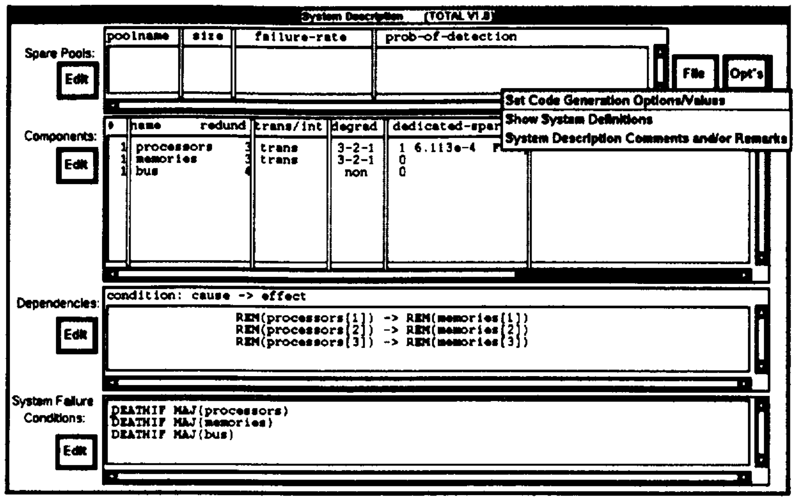
```
DEATHIF (MAJ(processors))
DEATHIF (MAJ(memories))
DEATHIF (MAJ(bus))
```

Figure 28: Cannot Quit When an Editor Session is Still Active

4.4 Setting Generation Options and Values

The options and values used when generating the ASSIST output file are specified by pulling down the "Opt's" menu as shown in Figure 29.

The Set Generation Values/Options panel is shown in Figure 30 with the default values given and in Figure 31 with the values used for the sample system. The panel contains the following items:


System Description (TOTAL V1.8)
Spare Pools:
Edit
Components:
Edit
Dependencies:
Edit
System Failure Conditions:
Edit
File
Opt's
Set Code Generation Options/Values
Show System Definitions
System Description Comments and/or Remarks

poolname	size	failure-rate	prob-of-detection

#	name	redund	trans/int	degrad	dedicated-spar
1	processors	3	trans	3-2-1	1 6.113e-4
1	memories	3	trans	3-2-1	0
1	bus	4		non	0

```
condition: cause -> effect
                REM(processors[1]) -> REM(memories[1])
                REM(processors[2]) -> REM(memories[2])
                REM(processors[3]) -> REM(memories[3])
```

```
DEATHIF MAJ(processors)
DEATHIF MAJ(memories)
DEATHIF MAJ(bus)
```

Figure 29: Using Pull-Down Menu to Set Generation Parameters

- Prune After Number of Component Failures: ⟶ Integer value representing the ASSIST pruning level. The default is to assume system failure when 4 or more of the components in the system have failed.

- List TOTAL tables in ASSIST comments? ⟶ Include a printout of the TOTAL system description tables in comments in the ASSIST-language file generated. The default is to include this table.

- Generate Assertions? ⟶ Include meaningful assertion statements in the ASSIST file to check correctness of the ASSIST description. The ASSIST assertions are described in detail in the ASSIST User's Manual. Assertions can give the user insight into the meaning of the ASSIST-language model descriptions, and may also be useful for uncovering possible errors in the system description. The default is not to generate assertions.

- Track Removed Components? ⟶ Create and update a variable (`TRASH_CAN_SIZE`) that tracks the number of components removed from the active configuration. This variable can be used in model pruning. The default is to track removed components.

- Additional Trace Comments? ⟶ Provide additional comments in the ASSIST-language description, such as noting the effects of dependencies on the TRANTO statements. The default is to include these additional comments.

- Keep but comment out any dead transitions? ⟶ This option, when selected, will comment out any generated `TRANTO` statements for which it can be determined that the source state will always be a death state. Use of this option will reduce the number of ASSIST warnings when the model is run. The default is to keep but comment out any dead transitions.

- Use explicit indicies when possible? ⟶ This option uses a numerical value when possible in generated TRANTO statements. For example, without explicit indicies, the generated "`IF (ix = 3) THEN ... TRANTO ABC[ix]`" will, with explicit indicies, be generated as "`IF (ix = 3) THEN ... TRANTO ABC[3]`". The default is to use explicit indicies when possible.

- Count Spare Failures for pruning? ⟶ Include spare failures when counting the number of components removed from the configuration for pruning. The default is to count spare failures for pruning.

- With trimming turned on? ⟶ Set TRIM=ON in the ASSIST-language file and calculate an appropriate TRIMOMEGA constant for ASSIST trimming. The default is to trim the model.

- Comment Conflicting Dependency Effects ⟶ When generating an ASSIST model, conflicting dependency effects, such as one or more dependencies causing both the incrementing and decrementing of the same variable, generate warning messages and are otherwise ignored. Although it is often desirable to ignore these, they can sometimes indicate an erroneously modeled file. If this option is selected, then a comment will be placed in the generated ASSIST input file in the destination of the TRANTO where the ignored effect would have gone had it not been ignored. The default is not to comment conflicting effects.

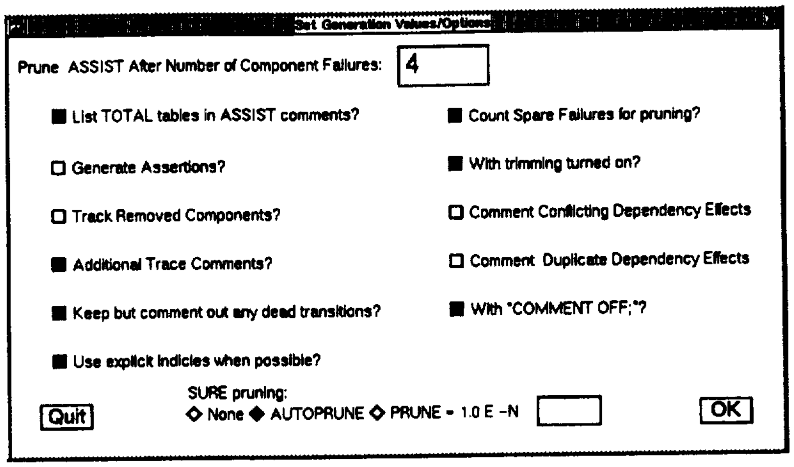
- Comment Duplicate Dependency Effects ⟶ When generating an ASSIST model, duplicated dependency effects, such as one or more dependencies causing the same effect specified more than once, are ignored. Although it is desirable to ignore these, they can sometimes indicate an erroneously modeled file. If this option is selected, then a comment will be placed in the generated ASSIST input file in the destination of the TRANTO where the ignored effect would have gone had it not been ignored. The default is not to comment duplicate effects.

- With "COMMENT OFF;"? ⟶ Include the statement "COMMENT OFF;" in the ASSIST model description, which will cause ASSIST to generate the SURE model without the state-space variable values in comments. Inclusion of SURE-level comments is usually not desired because it significantly increases the size of the SURE model file. The default is to turn "COMMENT OFF;".

- SURE pruning ⟶ This option determines how much SURE level pruning[2] is performed. With the choice "`PRUNE = 1.0E–N`", the value of "n" specified sets the prune level to 10^{-n}. The default is "AUTOPRUNE".

The user can select any combination of options from the panel.

The user unfamiliar with ASSIST model pruning and trimming is directed to [7] or [8] for a description of those techniques. The SURE program cannot solve models with competing transitions specified as means and standard deviations unless the probability of each path traversal is also given. Trimming fortunately has the effect of eliminating these competing transitions from the model. It is assumed that many models will require trimming, so trimming is turned on as



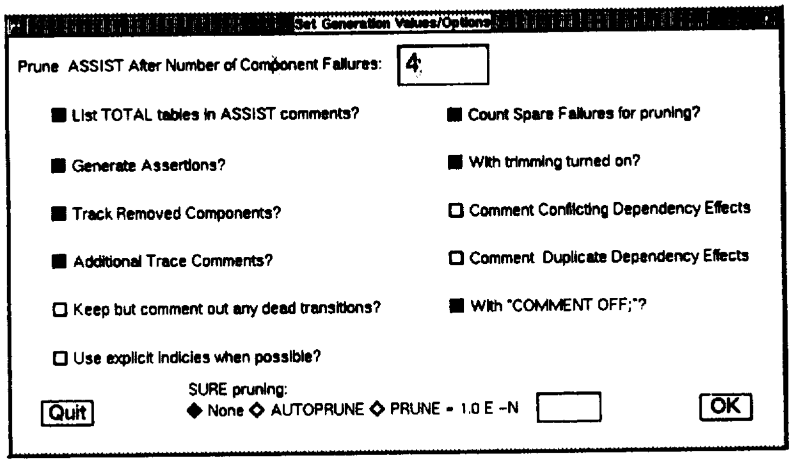
Set Generation Values/Options

Prune ASSIST After Number of Component Failures: 4

- ■ List TOTAL tables in ASSIST comments?
- ☐ Generate Assertions?
- ☐ Track Removed Components?
- ■ Additional Trace Comments?
- ■ Keep but comment out any dead transitions?
- ■ Use explicit indicies when possible?
- ■ Count Spare Failures for pruning?
- ■ With trimming turned on?
- ☐ Comment Conflicting Dependency Effects
- ☐ Comment Duplicate Dependency Effects
- ■ With "COMMENT OFF;"?

SURE pruning:
◇ None ◆ AUTOPRUNE ◇ PRUNE = 1.0 E −N

Figure 30: Default Options Used to Generate ASSIST Output File



Set Generation Values/Options

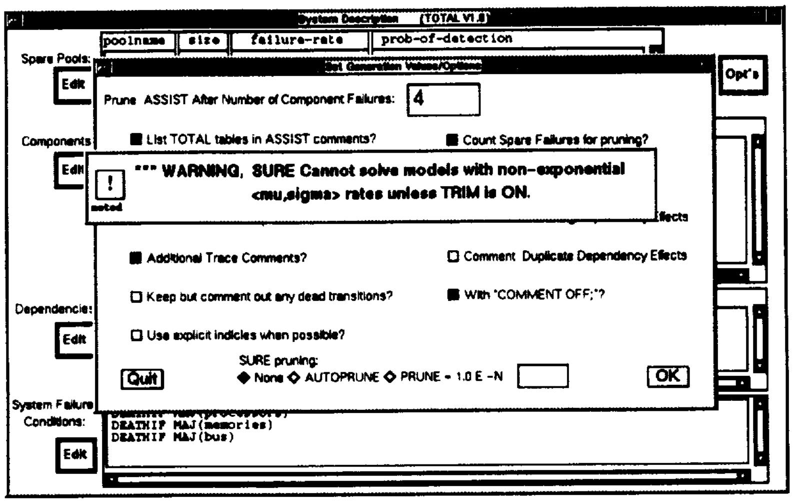
Prune ASSIST After Number of Component Failures: 4

- ■ List TOTAL tables in ASSIST comments?
- ■ Generate Assertions?
- ■ Track Removed Components?
- ■ Additional Trace Comments?
- ☐ Keep but comment out any dead transitions?
- ☐ Use explicit indicies when possible?
- ■ Count Spare Failures for pruning?
- ■ With trimming turned on?
- ☐ Comment Conflicting Dependency Effects
- ☐ Comment Duplicate Dependency Effects
- ■ With "COMMENT OFF;"?

SURE pruning:
◆ None ◇ AUTOPRUNE ◇ PRUNE = 1.0 E −N

Figure 31: Changing Options Used to Generate ASSIST Output File

the default. TOTAL automatically determines a suitable, conservative value for the trimming input parameter, TRIMOMEGA, so the trimming is done automatically. For the few models that have all **FAST** recoveries, the user can turn trimming off if he wishes to do so. If the user turns **TRIM OFF** and this is not appropriate, then the system will warn the user, as illustrated in Figure 32. The user can choose to ignore the warning, but the model will not be solvable as is.



Prune ASSIST After Number of Component Failures: 4

■ List TOTAL tables in ASSIST comments? ■ Count Spare Failures for pruning?

*** WARNING, SURE Cannot solve models with non-exponential <mu,sigma> rates unless TRIM is ON.

■ Additional Trace Comments? ☐ Comment Duplicate Dependency Effects

☐ Keep but comment out any dead transitions? ■ With "COMMENT OFF;"?

☐ Use explicit indicies when possible?

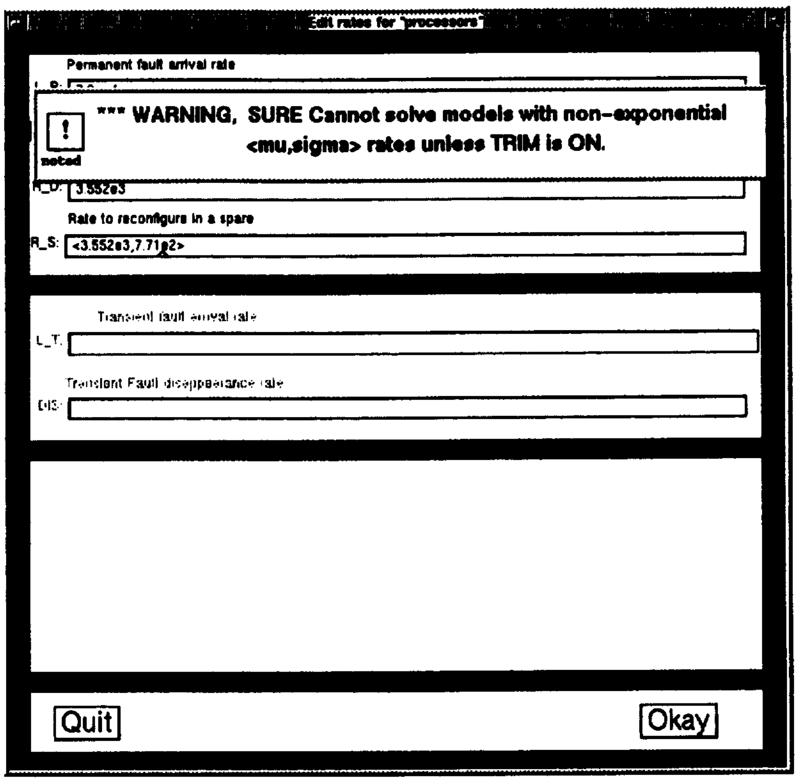
SURE pruning:
◆ None ◇ AUTOPRUNE ◇ PRUNE = 1.0 E -N

```
DEATHIF MAJ(memories)
DEATHIF MAJ(bus)
```

Figure 32: Warning for Systems for Which TRIM Should Remain On

If the user were to enter a system that had all **FAST** rates, turn trimming off, and then make a change to the rates that would require trimming in order to solve the modified model, the warning will appear again as illustrated in Figure 33. The user will be warned when a rate is changed from a safe rate to an unsafe rate and also when the "Okay" button is clicked.

Non-exponential rates are also disallowed when a component models transients as illustrated in Figure 34.



Edit rates for "processors"

Permanent fault arrival rate

L_P: [illegible]

*** WARNING, SURE Cannot solve models with non-exponential <mu,sigma> rates unless TRIM is ON.

noted

R_D: 3.552e3

Rate to reconfigure in a spare

R_S: <3.552e3,7.71e2>

Transient fault arrival rate

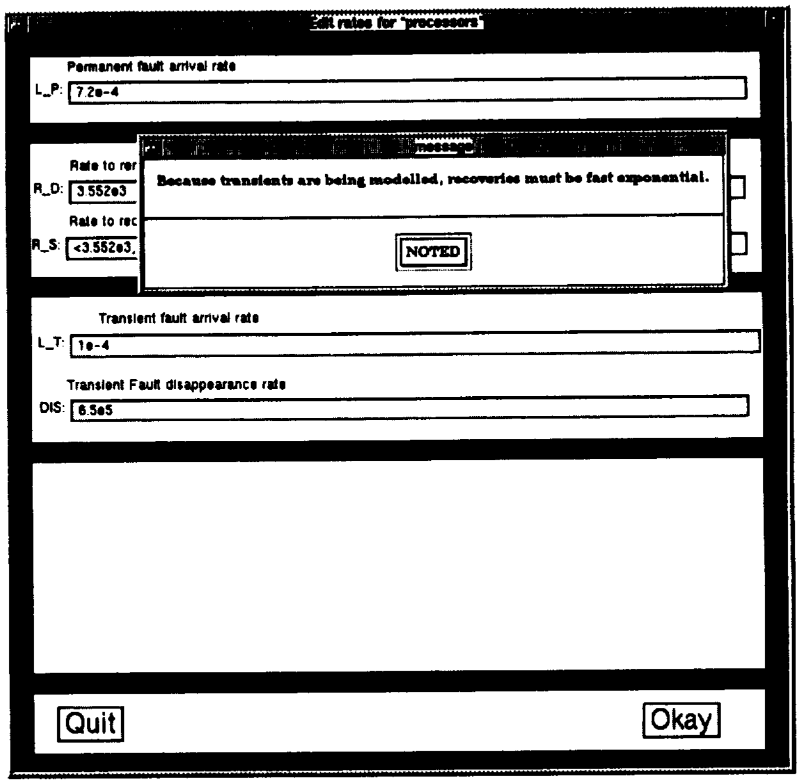
L_T:

Transient Fault disappearance rate

DIS:

Quit Okay

Figure 33: Warning Also Appears When Rates Are Changed


Edit rates for "processors"

Permanent fault arrival rate

L_P: 7.2e-4

Rate to rer

R_D: 3.552e3

Rate to rec

R_S: <3.552e3,

message

Because transients are being modelled, recoveries must be fast exponential.

Transient fault arrival rate

L_T: 1e-4

Transient Fault disappearance rate

DIS: 6.5e5

Figure 34: Warning Also Appears with non-exponential Rates when Transients are Modelled

5 Model Generation Logic and Mathematics

When defining a component, two types of recoveries can be specified. These are:

- recovery by swapping in a spare
- recovery by degrading the system

In either case, the faulty component is removed from the active configuration.

To specify the availability of spares, input a "Dedicated Spare Count" greater than zero and/or enter the name of at least the "Primary Spare Pool" on the "Component Configuration" panel as detailed in Section 3.2 on page 10.

To specify that a component will degrade, click on one of the radio buttons in the "Degradable" box on the "Component Configuration" panel as shown in Figure 10 on page 10.

5.1 Spare Component Logic and Mathematics

Whenever a component has available spares (either dedicated or pooled), then a faulty processor is removed and replaced with a spare. If there are dedicated spares, they will be used first. If there are spares available in the primary spare pool, they will be used first after all dedicated spares have been used. The secondary spares are used next, followed by the tertiary spares.

In the case of hot or warm spares, which can fail before ever being used, the spares always fail permanently. This is significant when transient faults are present. In the presence of a failed transient, replacement with a failed spare is modeled by removing the failed spare from the available spares, removing the failed transient, and adding a failed permanent to the configuration. In the presence of a failed permanent, this involves only removing the failed spare since the net effect of the replacement is to swap one failed permanent for another.

The generated transitions to replace a failed transient with a failed spare are:

```
NFS_component--,NFT_component--,NFP_component++,
  or
NFS_spare--,NFT_component--,NFP_component++,
```

whereas the generated transition to replace a failed permanent with a failed spare is:

```
NFS_component--,
  or
NFS_spare--,
```

where "component" is the name of any component that has a transient fault and "spare" is the name of any spare pool from which the component may draw spares. The "NFS_component" is used for the number of failed dedicated spares in the component and the "NFS_spare" for the number of failed spares in the spare pool named "spare".

When spares can fail, two transitions are generated instead of one; one for replacement with a working spare and one for replacement with a failed spare. The recovery rates are adjusted by the probability that the spare is working or failed, respectively. The probabilities are computed as:

```
IMPLICIT PRF_spare[NWS_spare,NFS_spare] = NFS_spare / NUC_spare;
IMPLICIT PRW_spare[NWS_spare,NFS_spare] = NWS_spare / NUC_spare;
```

where "NUC_spare" is the number of spares under consideration and is defined to be the total count of all spares that have not been detected as having failed.

In the case of partially detectable spares, these probabilities are computed as:

```
IMPLICIT PRF_spare[NWS_spare,NFD_spare,NFU_spare] = NFU_spare / NUC_spare;
IMPLICIT PRW_spare[NWS_spare,NFD_spare,NFU_spare] = NWS_spare / NUC_spare;
```

Where "NFU" stands for the "number of failed undetected" spares and "NFD" stands for the "number of failed detected" spares.

In the case where the probability of detecting a failed spare is 100%, the system will not generate a transition for swapping in a failed spare.

5.2 System Degradation Logic and Mathematics

Whenever the user defines a component as degradable by selecting some "Degradable" option other than "Non-degradable" from the "Component Configuration" panel, rules are generated to the ASSIST model output file in order to degrade the system according to the option specified. There are three options other than the "Non-degradable" option, namely:

- Triplex to duplex to simplex
- Triplex to simplex
- Duplex to zero

5.2.1 Degrade "Triplex to duplex to simplex"

This is a simple case of degradation, which requires no additional logic since every degradable component will always degrade by one until the system degrades to a triplex. This simple degrade-by-one logic is placed in the ELSE clause of the generated IF when one of the other two degradation schemes is selected as detailed in Sections 5.2.2-5.2.3.

5.2.2 Degrade "Triplex to simplex"

In the case where system degradation is from three components to one component, the generated transitions depend upon the current state of the component at the time at which the recovery by degradation occurs. The cases without and with transients will be considered separately. Because degradation is from three active to one active, degradation by removal of two components takes place only if the number active in the component in question is exactly three.

No transients occur for component

Whenever there are no transients for the component in question, the state space will be modeled as active and working.

When modeled collectively, the space will be (NA,NW). Because there is a fault present that has been detected, hence the need to degrade, there are only three possibilities for the current state space configuration:

```
(NA,NW) = (3,2)   (i.e, two working, one faulty active)
(NA,NW) = (3,1)   (i.e, one working, two faulty active)
(NA,NW) = (3,0)   (i.e, all three are faulty active)
```

Case where two are working

In the case where there are two working, since there are three active and one fault has been detected, the system must necessarily remove one of the two working with the failed one when it degrades from three to one. This happens 100% of the time:

```
(3,2) ⟶ (1,1)   NA=NA-2,NW--   100% of the time  (throw away one of the working ones)
```

When modeled individually, for each of the cases, a loop and if test will be generated in order to figure out which are the two working. In this case, there is a 50% probability of pulling the first working one and 50% probability of pulling the other working one.

Case where one is working

In the case where there is only one working, the system will remove the detected failed and one other processor. The other one removed could be the other failed one or it could be the working one. There is a 50% probability in either case. The transitions will be:

```
(3,1) ⟶ (1,0)   NA=NA-2,NW--   50% of the time  (throw away the working one)
(3,1) ⟶ (1,1)   NA=NA-2        50% of the time  (throw away the other failed one)
```

When modeled individually, for each of the cases, a loop and if test will be generated in order to figure out which of the other two is the working one and which is the failed one.

Case where all have failed

In the case where there are no working, the system will merely remove two faulty with 100% probability:

```
    (3,2) ⟶ (1,1)   NA=NA-2   100% of the time  (throw away one of the other two faulty ones)
```

When modeled individually, for each of the cases, a loop and if test will be generated in order to figure out which are the other two faulty. In this case, there is a 50% probability of pulling the first one of the other two faulty and a 50% probability of pulling the last one of the other two faulty.

Transients occur for component

Whenever there are transients for the component in question, the state space will be modeled as working, failed permanent, and failed transient.

When modeled collectively, the space will be `(NW,NFP,NFT)`. Because there is one fault present that has been detected, hence the need to degrade, the number working can be no more than two. There are many possibilities for the current state space configuration:

```
    NW = 2
        (NW,NFP,NFT) = (2,1,0)
        (NW,NFP,NFT) = (2,0,1)
    NW = 1
        (NW,NFP,NFT) = (1,2,0)
        (NW,NFP,NFT) = (1,0,2)
        (NW,NFP,NFT) = (1,1,1)
    NW = 0
        (NW,NFP,NFT) = (0,3,0)
        (NW,NFP,NFT) = (0,0,3)
        (NW,NFP,NFT) = (0,2,1)
        (NW,NFP,NFT) = (0,1,2)
```

Case where two are working

In the case where two are working, the system will remove the detected failed and one of the two working. In either case, there is a 100% probability of removing one of the working ones. The transitions will be:

```
    If the failed one is a permanent:
       (2,1,0) ⟶ (1,0,0) 100%  NW--,NFP--  (throw away one working)
    If the failed one is a transient:
       (2,0,1) ⟶ (1,0,0) 100%  NW--,NFT--  (throw away one working)
```

When modeled individually, for each of the cases, a loop and if test will be generated in order to figure out which are the two working. In this case, there is a 50% probability of pulling the first one of the two working and a 50% probability of pulling the last one of the two working.

Case where one is working

In the case where only one is working, the system will remove the detected failed and one of the other two. One of the other two that could be removed is the working one; however, the other one that could be removed might be either failed transient or failed permanent.

If both are failed permanents, then the system could throw away a working with the detected failed permanent or it could throw away the other failed permanent with the detected failed permanent. Each possibility would happen with 50% probability.

If both are failed transients, then the system could throw away a working with the detected failed transient or it could throw away the other failed transient with the detected failed transient. Each possibility would happen with 50% probability.

If there is one each of failed transient and failed permanent, then the detected failed could be the failed permanent or it could be the failed transient. There is a 50% chance of either case. When the detected one is a failed permanent, then there is a conditional 50% chance of removing the working and a conditional 50% chance of removing the failed transient with the failed permanent. When the detected one is a failed transient, then there is a conditional 50% chance of removing the working and a conditional 50% chance of removing the failed permanent with the failed transient. The unconditional probabilities multiply to 25% with the case of removing both a failed permanent and a failed transient occurring twice to make 50%.

The transitions will be:

```
If both failed are permanents:
   (1,2,0) ⟶ (0,1,0) 50%  NW--,NFP--   (throw away the working)
   (1,2,0) ⟶ (1,0,0) 50%  NFP=NFP-2    (throw away other failed)
If both failed are transients:
   (1,0,2) ⟶ (0,0,1) 50%  NW--,NFT--   (throw away the working)
   (1,0,2) ⟶ (1,0,0) 50%  NFT=NFT-2    (throw away other failed)
If one of the two failed is permanent and the other transient:
   (1,1,1) ⟶ (0,1,0) 25%  NW--,NFT--   (throw away working)
   (1,1,1) ⟶ (1,0,0) 50%  NFP--,NFT--  (throw away other failed)
   (1,1,1) ⟶ (0,0,1) 25%  NW--,NFP--   (throw away working)
```

When modeled individually, for each of the cases, a loop and if test will be generated in order to figure out which is the other failed and which is the working.

Case where all are failed

In the case where all have failed, the system will remove the detected failed and one of the other two failed.

If all three have failed permanently, then the detected fault must be a permanent fault and another failed permanent will be thrown out with it. The probability is 100%.

If all three have failed transiently, then the detected fault must be a transient fault and another failed transient will be thrown out with it. The probability is 100%.

If two of the failed have failed permanently, then the other will have failed transiently. If the fault detected is a permanent, which will happen two thirds of the time, then the other fault could be the other failed permanent or it could be the failed transient. The conditional probabilities are 50% each. If the fault detected is a transient, which will happen one third of the time, then the other fault could be the other failed transient or it could be the failed permanent. The conditional probabilities are 50% each. Multiplying the probabilities, the result is a 67% probability of removing one of the two permanents with the only transient and a 33% probability of removing both of the permanents.

If two of the failed have failed permanently, then the other will have failed permanently. If the fault detected is a transient, which will happen two thirds of the time, then the other fault could be the other failed transient or it could be the failed permanent. The conditional probabilities are 50% each. If the fault detected is a permanent, which will happen one third of the time, then the other fault could be the other failed permanent or it could be the failed transient. The conditional probabilities are 50% each. Multiplying the probabilities, the result is a 67% probability of removing one of the two transients with the only permanent and a 33% probability of removing both of the transients.

The transitions will be:

```
If all three have failed permanently:
   (0,3,0) ──→ (0,1,0) 100% NFP=NFP-2   (throw away another permanent)
If all three have failed transiently:
   (0,0,3) ──→ (0,0,1) 100% NFT=NFT-2   (throw away another transient)
If two have failed permanently:
   (0,2,1) ──→ (0,1,0) 67%  NFP--,NFT-- (throw away perm and trans)
   (0,2,1) ──→ (0,0,1) 33%  NFP=NFP-2   (throw away both permanents)
If two have failed transiently:
   (0,1,2) ──→ (0,0,1) 67%  NFP--,NFT-- (throw away perm and trans)
   (0,1,2) ──→ (0,1,0) 33%  NFT=NFT-2   (throw away both transients)
```

When modeled individually, for each of the cases, a loop and if test will be generated in order to figure out which are the failed transients and which are the failed permanents. When all three have failed permanently, then the detected one is failed permanent and there is a 50% chance of removing the first of the other two failed permanents and a 50% chance of removing the last of the other two permanents. When all three have failed transiently, then the detected one is failed transient and there is a 50% chance of removing the first of the other two failed transients and a 50% chance of removing the last of the other two transients.

5.2.3 Degrade "Duplex to zero" (Self-Checking Pairs)

In the case where system degradation is from two components to no components, the number active, number working, number failed permanent (in the case of transients), and number failed transient (in the case of transients) are all set to zero.

When the component is modeled individually, each active, working, failed permanent (in the case of transients), and failed transient (in the case of transients) is set to `FALSE`.

The rate used for this single transition is the normal recovery rate for degradation, namely "`R_D_comp`" for component "comp".

Because degradation is from two active to no active, degradation by removal of two components takes place only if the number active in the component in question is exactly two.

5.3 Recoveries and Death States

In many systems, many of the cases allowed for during the spare replacement and degradation recoveries can never happen because the states are death states. The cases must be considered, however, because this depends upon which system failure conditions were specified. For some of the more common system failure conditions, TOTAL is able to determine that the current state is a death state and the transitions are automatically commented out of the generated ASSIST file so as to decrease the number of warnings such as:

```
[WARNING] NO TRANSITIONS GENERATED USING TRANTO ON LINE : nnn
```

However, users employing textual `DEATHIF` descriptions or complex combinations of system failure conditions may see this warning when ASSIST executes. The warning can be safely ignored.

6 System Requirements

The TOTAL prototype program was written using the Transportable Applications Environment (TAE) developed at NASA Goddard Space Flight Center in Maryland. The TAE system executes on top of the Motif windowing system. The Motif system, which is based on X-Windows, is available on quite a few different systems, including both SUN and VAX systems.

The TAE package was used in order to be portable between different systems and in order to save development time.

6.1 Hardware/Operating System Requirements

6.1.1 Requirements to run pre-compiled versions

Executable files compiled at version 4.1.2 of SUN OS on the SUN SPARCSTATION[27] architecture are available for sites without compilers or TAE. To run the pre-compiled SUN OS version, some version of the X (MIT or OpenWindows 3.0) must be executing. The Motif (mwm) window manager must also be executing.

The TOTAL program has been ported to the VAX. Executable files compiled at VAX VMS V5.5 are available for VAX sites without compilers or TAE. Executables are built with version V3.1-051 of the VAX C compiler and with V3.8-273 of the VAX PASCAL compiler. To run the pre-compiled VAX VMS, the DECWINDOWS MOTIF window manager must be executing.

6.1.2 Requirements to re-compile

The TOTAL program was written in ANSI-standard "C" by David Boerschlein. It currently compiles on a SUN SPARCSTATION with the TAE V5.2 libraries executing the MOTIF windowing system or on a VAX with the TAE V5.2 libraries under the VMS operating system executing DECWINDOWS MOTIF.

An ANSI-standard "C" compiler is available from the Free Software Foundation ("gcc"). The "C" compiler that is available with the VAX VMS operating system is also ANSI. The current SUN "C" compiler will not compile TOTAL. SUN may release an ANSI "C" compiler in the future.

Some of the programs (such as SURE, STEM, and PAWS) that are used to process ASSIST output files require a Pascal compiler in order to re-build. The VMS version of TOTAL also requires the Pascal compiler.

[27] a trademark of SPARC International

6.2 Memory Requirements

The TOTAL program will not execute on a system with less than 16MB of memory, and 24MB is recommended for efficiency. This is because TOTAL makes use of a lot of windows. Both MOTIF and DECWINDOWS MOTIF use a good deal of memory.

6.3 Software System Requirements

The user must be executing under either the DECWINDOWS MOTIF or the MOTIF windowing system on top of SUN OS.

Features of ASSIST referenced in TOTAL require ASSIST revision 7.1 or higher as well as SURE, STEM, and/or PAWS revisions 7.9.8 or higher. Model files produced with TOTAL can be processed with older revisions of SURE, STEM, and/or PAWS provided that the combinatorial functions (`COMB`, `PERM`, `FACT`, and `GAM`) and the modulo operators (`MOD`, `DIV`, and `CYC`) are not referenced. Use of dependencies that reference components of different redundancy counts can generate ASSIST code that references the `CYC` operator. In this case, SURE version 7.9.3 is required.

6.4 Monochrome vs Color

The TOTAL program will run on either a monochrome or color terminal. Some of the features of its look are more pronounced on a color terminal.

7 Concluding Remarks

A prototype spreadsheet interface for a reliability analysis tool set has been described. The TOTAL interface program allows the user to describe a system at a level of abstraction appropriate for designers of fault-tolerant systems, even those with no background knowledge in reliability analysis calculations. We do not plan to take this concept beyond the prototyping stage; however, we believe that the interface method we have developed is powerful, and we hope that other reliability analysis tool developers will gain new insights from our research and perhaps develop a commercially available tool with the capabilities we envision.

References

[1] R. W. Butler, "The SURE approach to reliability analysis," *IEEE Transactions on Reliability*, vol. 41, pp. 210–218, June 1992.

[2] R. W. Butler and A. L. White, "SURE reliability analysis: Program and mathematics," NASA Technical Paper 2764, Mar. 1988.

[3] R. W. Butler and P. H. Stevenson, "The PAWS and STEM reliability analysis programs," NASA Technical Memorandum 100572, Mar. 1988.

[4] R. W. Butler, "An abstract language for specifying markov reliability models," *IEEE Transactions on Reliability*, vol. R-35, pp. 595–601, Dec. 1986.

[5] S. C. Johnson, "ASSIST user's manual," NASA Technical Memorandum 87735, Aug. 1986.

[6] A. L. White and D. L. Palumbo, "State reduction for semi-markov reliability models," in *The 36th Annual Reliability and Maintainability Symposium*, (Los Angeles, CA), Jan. 1990.

[7] S. C. Johnson, "Reliability analysis of large, complex systems using ASSIST," in *AIAA/IEEE 8th Digital Avionics Systems Conference*, (San Jose, California), Oct. 1988.

[8] R. W. Butler and S. C. Johnson, "The art of fault-tolerant system reliability modeling," NASA Technical Memorandum 102623, Mar. 1990.

A Command Line Interface

For both UNIX and VMS systems, the TOTAL program is invoked via:

> **total**
>
> or
>
> **total** <resource-file-name>
>
> or
>
> **total** <total-file-name>
>
> or
>
> **total** <total-file-name> <resource-file-name>
>
> or
>
> **total** <resource-file-name> <total-file-name>

Options can appear anywhere on the command line after the name of the command "total". They may precede or they may follow the optional resource and/or TOTAL file names.

The user must specify a suffix of ".res" for the resource file name. The user may explicitly specify the suffix of ".tot" for the TOTAL file name but this is not required.

If more than one resource file name is specified, then the last one specified on the command line is used and the prior ones are ignored.

If more than one TOTAL file name is specified, then the last one specified on the command line is used and the prior ones are ignored.

A.1 Command Line Options File

The user may define an options file to override the default options. The options specified in the total options file are parsed first before any command line options.

The name of the total options file on UNIX systems is:

```
.total_options
```

and its name on VMS systems is:

```
total_options.cfg
```

The file is located in the user's home directory.

Any and all of the command line options can be defaulted in the total options file.

No option in the total options file may span two lines and backslashes are not allowed. The user is free to put more than one option on a line as long as the options are separated by whitespace. The user is also free to use as many lines as desired.

A.2 Command Line Options

The TOTAL command line allows the user to specify options. These options control a number of parameters and allow the user more control over how the TOTAL program executes.

Options must be preceded by a slash under VMS as in:

```
/batch
```

and must be preceded by a dash under UNIX as in:

```
—batch
```

The following sample command lines are equivalent and specify the use of the "emacs" editor instead of "vi", use of the "MyResource.res" resource file and initialization by reading in the existing (old) system called "sample":

```
total MyResource.res sample.tot  —ema
total MyResource.res sample  —ema
total sample MyResource.res  —ema
total MyResource.res  —ema  sample
total  —ema  MyResource.res sample
```

Options may be specified either in upper or lower case. The normal UNIX case sensitivity does not apply to the TOTAL command line options.

The following options are available:

- **—audio** ⟶ This option tells TOTAL to play sounds when certain actions take place. This can be useful, for example, with large models that take a fair amount of time to generate and solve. In order for sounds to play, the operating system must have been installed with the demos option and the sounds files must exist in "/usr/demo/SOUND/sounds". The default is not to play any sounds.
- **—batch** *filename* ⟶ Specifies that the program is to execute in batch mode instead of in workstation mode. The typical user will not be interested in batch (".tba") files, however, many of the operations that can be performed with the panels can also be done via special "commands" in a batch file. Batch files are not fully supported. The default is interactive processing mode.
- **—comp** =*nnn* ⟶ This option specifies the maximum number of components in the system. The default is 100.
- **—con** =*nnn* ⟶ This option specifies the maximum number of conditional dependency condition **AND** list items allowed in the system. This number is the cumulative total of the lengths of each condition **AND** list for each conditional dependency in the system. The default is 200.
- **—dand** =*nnn* ⟶ This option specifies the maximum number of system failure **AND** list items allowed in the system. This number is the cumulative total of the lengths of each **DEATHIF**. The default is 200.
- **—dep** =*nnn* ⟶ This option specifies the maximum number of dependencies allowed in the system. Each dependency counts only once even though it may have several effects. The default is 50.
- **—dif** =*nnn* ⟶ This option specifies the maximum number of system failure conditions (**DEATHIF's**) allowed in the system. Each condition counts only once even though it may have several **AND's** in the list. The default is 50.
- **—dl** =*nnn* ⟶ This option specifies the maximum number of dependency lines which a system is allowed to have. Note that each effect increases the number of lines required by one. One dependency will therefore require one line for each of its effects. The default is 180.
- **—eff** =*nnn* ⟶ This option specifies the maximum number of dependency effect list items allowed in the system. This number is the cumulative total of the number of effects for each dependency in the system. The default is 200.

- **−ema** ⟶ This option specifies that the **EMACS** editor is to be used when the user requests to edit the ASSIST output file. This editor is available on both **VMS** and **Unix** systems as well as on many other systems. One version of **EMACS** is available from the Free Software Foundation. The default is to use the system editor and not to use **EMACS**. The **Unix** version of TOTAL assumes the version from the foundation in that it uses the **−nw** option when invoking the editor.
- **−eve** ⟶ This option specifies that the **VMS EVE** editor is to be used when the user requests to edit the ASSIST output file. If this option is specified and the user is running on a **Unix** system, then TOTAL assumes that the user really meant to specify **−vi** and will use the **Unix vi** editor instead. This (**−eve**) is the default on **VMS** systems.
- **−nc** =*nnn* ⟶ This option specifies the maximum number of named constant lines allowed. Named constants are not generated in TOTAL 1.0. The default is 100.
- **−noaudio** ⟶ This option turns off the **−audio** option.
- **−noeve** ⟶ This option turns off the **−eve** option.
- **−novi** ⟶ This option turns off the **−vi** option.
- **−pool** =*nnn* ⟶ This option specifies the maximum number of shared spare pools in the system. The default is 50.
- **−prb** =*nnn* ⟶ This option specifies the maximum length of the probability buffer. This buffer is used to store the rate expressions for the **TRANTO's** before they are written to the ASSIST output file. The default of 512 bytes should be much more than sufficient for virtually all system descriptions.
- **−pro** =*nnn* ⟶ This option is the same as **−prb** .
- **−rc** =*nnn* ⟶ This option specifies the maximum number of rate constant lines allowed. Rate constants are generated for inclusion in the ASSIST output file and are displayed in the "System Definitions" panel. Rate constants include means and standard deviations as well as fast and slow rates as specified for each component, dedicated spare failure rates for each component, and pooled spare failure rates for each spare pool. The rate constants also include the definition of TRIMOMEGA when trimming is turned on. The definition of TRIMOMEGA takes many lines (one line per component fault arrival rate). The default is 800.
- **−sc** =*nnn* ⟶ This option specifies the maximum number of space constant lines allowed. Space constants are constants that are required in the generated **SPACE** and **START** statements as displayed in the "System Definitions" panel for inclusion into the ASSIST output file. These constants include the initial numbers of components and spares. Also included is the grand total of all of the components used in the system. The default is 300, which allows for approximately 150 initial numbers of components and spares.
- **−sp** =*nnn* ⟶ This option specifies the maximum number of space lines allowed for a generated **SPACE** statement as displayed in the "System Definitions" panel. The default is 102, which corresponds to 100 state space variables plus a leading and trailing line.
- **−ssv** =*nnn* ⟶ This option specifies the maximum number of auxiliary state-space variables that are allowed in the system. The default is 50.
- **−tto** =*nnn* ⟶ This option specifies the length of the transition buffer. This buffer is used to store the **TRANTO's** before they are written to the ASSIST output file. When there is a big "domino effect" with dependency causes and effects, the **TRANTO** can get rather large. The default is 4096, which corresponds to about 50 lines in the ASSIST output file.
- **−vi** ⟶ This option specifies that the **Unix vi** editor is to be used when the user requests to edit the ASSIST output file. If this option is specified and the user is running on a **VMS** system, then TOTAL assumes that the user really meant to specify **−eve** and will use the **VMS EVE** editor instead. This (**−vi**) is the default on **Unix** systems.

A.3 The Resource File

If the user has access to the Transportable Applications Environment (TAE) utility[28], then the user can use the TAE workbench to customize the resource file. This would be useful, for example, to change the colors used in the displays. To do this, copy the resource file and run the workbench. For example:

```
cp /usr/local/bin/total_resource_file.res myres.res
taewb -f myres.res
```
[29]

or

```
COPY RELIABILITY$DIR:TOTAL_RESOURCE_FILE.RES MYRES.RES
RUNWB MYRES.RES
```
[30]

The TAE utility is not required to run TOTAL (just to re-build and link the executable and customize the resource file).

A.4 Converting Between ASCII and TOTAL Description Files

Because TOTAL description files (".tot" files) are stored in system binary format, a description file cannot be transferred to another system and used directly. For example, a total description file created on a SUN system cannot be directly read on a VAX system.

In order to transfer TOTAL description files from one system to another, two commands are provided:

- tot2txt ⟶ Converts from binary ".tot" format to ASCII ".txt" format.
- txt2tot ⟶ Converts from ASCII ".txt" format to binary ".tot" format.

For example, to transfer a file named "myfile.tot" from the SUN to the VAX, do the following:

```
sun% tot2txt myfile
sun% $then copy the file "myfile.txt" to the vax$
vax$ txt2tot myfile
```

[28]Available from COSMIC, # COS-10033 or COS-10034
[29]This corresponds to TAE V5.2 on SUN OS
[30]This corresponds to TAE V4.1 on VMS

B Installation of TOTAL

Installation of TOTAL is automated by the Makefile and or the VMSINSTAL procedure. There is a default resource file that is accessed if one is not specified on the command line. This resource file must reside in the system directories.

B.1 Under SUN OS

The default resource file must be accessed via the path:

```
/usr/local/bin/total_resource_file.res
```

Each user can override this default by listing a resource file pathname in his/her own home-directory ".total_options" file.

B.2 Under VMS

On VMS, the default resource file must be accessed via the path:

```
RELIABILITY$DIR:TOTAL_RESOURCE_FILE.RES
```

Each user can override this default by listing a resource file pathname in his/her own home-directory "total_options.cfg" file.

C Icons for TOTAL under Motif

The icons for the panels in TOTAL are illustrated in Figure 35.

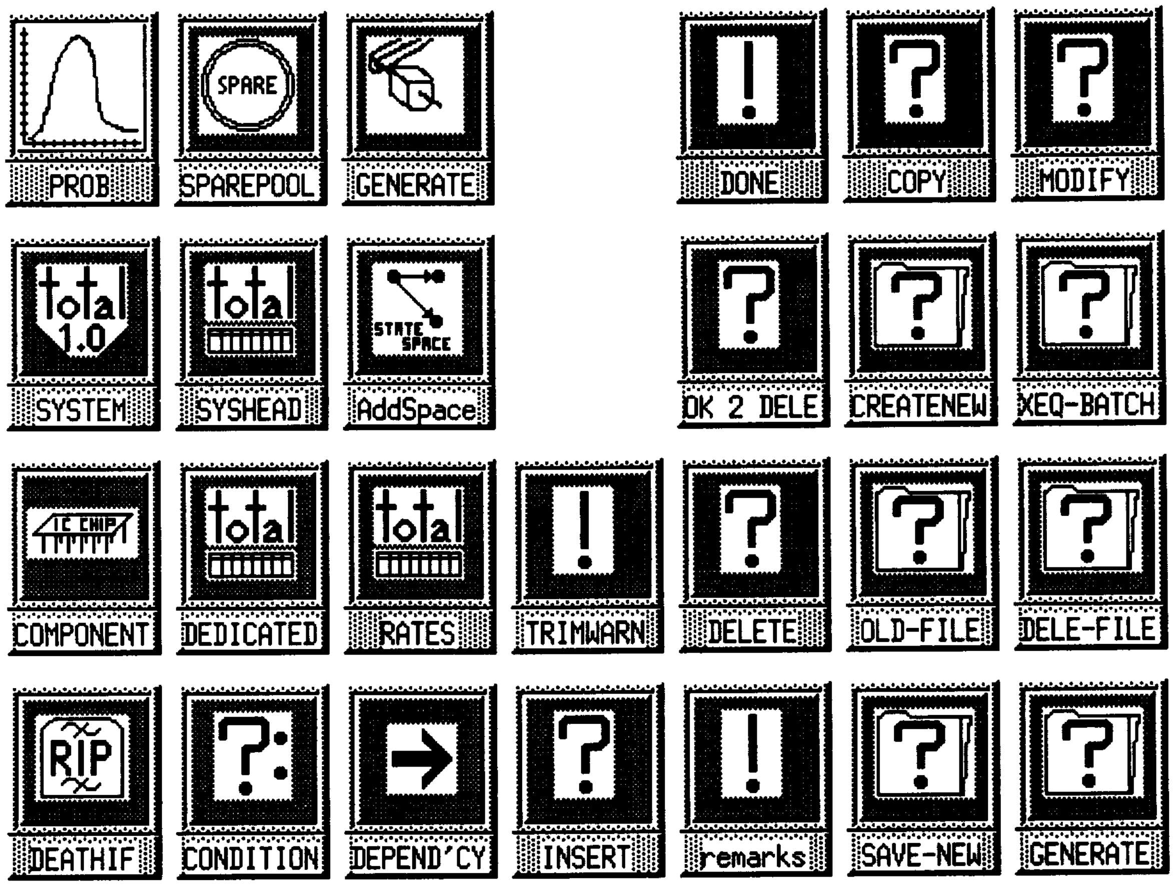


Figure 35: Icons that appear when TOTAL panels are minimized

D Sample ASSIST file generated by TOTAL

The following is the ASSIST input file generated by TOTAL that corresponds to the sample system in this manual:

```
C_OPTION TOTAL_GENERATED;
C_OPTION WL=40;
C_OPTION WID=192;
C_OPTION WRAP_LONG_CONSTANT_EXPRS;
COMMENT OFF;


                (**********************************************)
                (**********************************************)
                (***                                        ***)
                (***   TABLE ORIENTED SYSTEM DESCRIPTION    ***)
                (***                                        ***)
                (**********************************************)
                (**********************************************)

(*=============================================================================

 This example system consists of a triplex set of processors with one spare
 processor, a triplex set of memory units, and a quadruplex bus.  All messages
 and calculations performed in the system are subject to majority voting to
 detect and mask failures.  Thus, a majority of the processors in the current
 configuration must be working, or system failure occurs.  Similarily, a
 majority of the memories and a majority of the buses must be working.  Upon
 detection of the first processor failure, the spare processor is brought into
 the configuration to replace the faulty processor.  Upon the second processor
 failure, the faulty processor is removed and the remaing two working
 processors continue in duplex mode.  Failure of one of the remaining two
 working processors is assumed to immediately defeat the majority voter.  Each
 memory unit is attached to one of the processors and removal of a failed
 processor also results in removal of its attached memory.  The memory units
 degrade from triplex to duplex to simplex as failed memory units are detected
 in the configuration or are removed because of processor removals.  The bus
 is non-reconfigurable.


No spare pools defined.

list of components:

    #     name redund trans/int degrad  dedicated-spares
   -- ------------ -- --------- ------ ------------------
    1 processors   3 trans     3-2-1   1 6.113e-4  Fully
    1 memories     3 trans     3-2-1   0
    1 bus          4            non    0

      pools:
      -------- -------- --------



        L_P      R_D      R_S      L_T      DIS
      -------- -------- -------- -------- --------
      7.2e-4   3.552e3  3.552e3  1e-4     6.5e5
      4.4e-3   12.213e3 n/a      5.403e-3 6.8e-4
      2.202e-3 n/a      n/a      n/a      n/a

list of dependencies:
```

```
                                cause -> effect
    ----------------------------------    ----------------------------------
                   REM(processors[1]) -> REM(memories[1])
                   REM(processors[2]) -> REM(memories[2])
                   REM(processors[3]) -> REM(memories[3])

Deathif list:

    DEATH   expression
    ------- -----------------------------------------------------------------
    DEATHIF MAJ(processors)
    DEATHIF MAJ(memories)
    DEATHIF MAJ(bus)

No extra state-space variables defined.

===============================================================================*)

                            (****************************)
                            (****************************)
                            (***                      ***)
                            (***    SPACE CONSTANTS   ***)
                            (***                      ***)
                            (****************************)
                            (****************************)

Prune_After  = 4;    (* Prune after 4'th component failure *)
NI_processors= 3;    (* Redundancy count for "processors" *)
NSI_processors= 1;   (* Dedicated spare count for "processors" *)
NI_memories  = 3;    (* Redundancy count for "memories" *)
NI_bus       = 4;    (* Redundancy count for "bus" *)
NTOT = NI_processors + NSI_processors + NI_memories + NI_bus;
                     (* total number of components initially *)
NCFMX = NTOT;        (* Maximum possible component failure count *)

                            (****************************)
                            (****************************)
                            (***                      ***)
                            (***    RATE  CONSTANTS   ***)
                            (***                      ***)
                            (****************************)
                            (****************************)

L_P_processors= 7.2e-4; (* Permanent fault arrival rate for "processors" *)
R_D_processors= 3.552e3; (* Rate to degrade for "processors" *)
R_S_processors= 3.552e3; (* Rate to reconfigure spare into "processors" *)
L_T_processors= 1e-4;   (* Transient fault arrival rate for "processors" *)
DIS_processors= 6.5e5;  (* Rate at which "processors" transient disappears *)
L_P_memories = 4.4e-3;  (* Permanent fault arrival rate for "memories" *)
R_D_memories = 12.213e3; (* Rate to degrade for "memories" *)
L_T_memories = 5.403e-3; (* Transient fault arrival rate for "memories" *)
DIS_memories = 6.8e-4;  (* Rate at which "memories" transient disappears *)
L_P_bus      = 2.202e-3; (* Permanent fault arrival rate for "bus" *)

TRIMOMEGA = NI_processors*L_P_processors +
            NI_processors*L_T_processors +
            NI_memories*L_P_memories +
            NI_memories*L_T_memories +
            NI_bus*L_P_bus;
TRIM ON;

AUTOPRUNE = 0;
PRUNE = 0;
```

```
L_S_processors= 6.113e-4; (* Dedicated "processors" spare fail rate *)

                         (*********************************)
                         (*********************************)
                         (***                           ***)
                         (***    STATE SPACE DEFINED    ***)
                         (***                           ***)
                         (*********************************)
                         (*********************************)

SPACE =
  (
  (W_processors : ARRAY[1..NI_processors] OF BOOLEAN),
                  (* Each working in "processors" *)
  (FP_processors: ARRAY[1..NI_processors] OF BOOLEAN),
                  (* Each permanent failed in "processors" *)
  (FT_processors: ARRAY[1..NI_processors] OF BOOLEAN),
                  (* Each transient faulty in "processors" *)
   NWS_processors: 0..NSI_processors,
                  (* dedicated Working spares count,  "processors" *)
   NFS_processors: 0..NSI_processors,
                  (* dedicated Failed spares count,  "processors" *)
  (W_memories   : ARRAY[1..NI_memories] OF BOOLEAN),
                  (* Each working in "memories" *)
  (FP_memories  : ARRAY[1..NI_memories] OF BOOLEAN),
                  (* Each permanent failed in "memories" *)
  (FT_memories  : ARRAY[1..NI_memories] OF BOOLEAN),
                  (* Each transient faulty in "memories" *)
   NW_bus       : 0..NI_bus, (* Count of working in "bus" *)
   TRASH_CAN_SIZE : 0..NTOT, (* removed components *)
   NCF          : 0..NCFMX (* Number of component failures for PRUNEIF *)
  );

IMPLICIT NW_processors[W_processors] = COUNT(W_processors);
IMPLICIT NFP_processors[FP_processors] = COUNT(FP_processors);
IMPLICIT NFT_processors[FT_processors] = COUNT(FT_processors);

IMPLICIT NWOT_processors[W_processors,FT_processors] =
        (* Number Working or Transient "processors" *)
        NW_processors + NFT_processors;
IMPLICIT WOT_processors[W_processors,FT_processors](K) =
        (* Working or Transient each "processors" *)
        W_processors[K] OR FT_processors[K];
IMPLICIT TFA_processors[FP_processors,FT_processors] =
        (* Total active failed in "processors" *)
        NFP_processors + NFT_processors;
IMPLICIT NF_processors[FP_processors,FT_processors] =
        (* Total active failed in "processors" *)
        NFP_processors + NFT_processors;
IMPLICIT TWA_processors[W_processors] =
        (* Total active working in "processors" *)
        NW_processors;
IMPLICIT TA_processors[W_processors,FP_processors,FT_processors] =
        (* Total active in "processors" *)
        TFA_processors + TWA_processors;
IMPLICIT FA_processors[FP_processors,FT_processors](K) =
        (* Active failed each "processors" *)
        FP_processors[K] OR FT_processors[K];
IMPLICIT A_processors[W_processors,FP_processors,FT_processors](K) =
        (* Test if active in "processors" *)
        W_processors[K] OR FP_processors[K] OR FT_processors[K];

IMPLICIT NW_memories[W_memories] = COUNT(W_memories);
IMPLICIT NFP_memories[FP_memories] = COUNT(FP_memories);
IMPLICIT NFT_memories[FT_memories] = COUNT(FT_memories);

IMPLICIT NWOT_memories[W_memories,FT_memories] =
        (* Number Working or Transient "memories" *)
        NW_memories + NFT_memories;
IMPLICIT WOT_memories[W_memories,FT_memories](K) =
        (* Working or Transient each "memories" *)
        W_memories[K] OR FT_memories[K];
IMPLICIT TFA_memories[FP_memories,FT_memories] =
        (* Total active failed in "memories" *)
        NFP_memories + NFT_memories;
IMPLICIT NF_memories[FP_memories,FT_memories] =
        (* Total active failed in "memories" *)
        NFP_memories + NFT_memories;
IMPLICIT TWA_memories[W_memories] = (* Total active working in "memories" *)
        NW_memories;
IMPLICIT TA_memories[W_memories,FP_memories,FT_memories] =
        (* Total active in "memories" *)
        TFA_memories + TWA_memories;
IMPLICIT FA_memories[FP_memories,FT_memories](K) =
        (* Active failed each "memories" *)
        FP_memories[K] OR FT_memories[K];
IMPLICIT A_memories[W_memories,FP_memories,FT_memories](K) =
        (* Test if active in "memories" *)
        W_memories[K] OR FP_memories[K] OR FT_memories[K];

IMPLICIT TFA_bus[NW_bus] = (* Total active failed in "bus" *)
        NI_bus - NW_bus;
IMPLICIT NF_bus[NW_bus] = (* Total active failed in "bus" *)
        NI_bus - NW_bus;
IMPLICIT TWA_bus[NW_bus] = (* Total active working in "bus" *)
        NW_bus;
IMPLICIT TA_bus[NW_bus] = (* Total active in "bus" *)
        NI_bus;

IMPLICIT NS_processors[NWS_processors,NFS_processors] =
        NWS_processors + NFS_processors;
IMPLICIT NUC_processors[NWS_processors,NFS_processors] =
        NWS_processors;
IMPLICIT PRF_processors[NWS_processors,NFS_processors] =
        NFS_processors / NUC_processors;
IMPLICIT PRW_processors[NWS_processors,NFS_processors] =
        NWS_processors / NUC_processors;
IMPLICIT TS_processors[NWS_processors,NFS_processors] =
        (* Total spares available to "processors" *)
        NWS_processors + NFS_processors;
IMPLICIT TUC_processors[NWS_processors,NFS_processors] =
        (* Total spares under consideration for "processors" *)
        NUC_processors;

START =
  (
   (NI_processors OF TRUE), (* Each working in "processors" *)
   (NI_processors OF FALSE), (* Each permanent failed in "processors" *)
   (NI_processors OF FALSE), (* Each transient faulty in "processors" *)
   NSI_processors, (* dedicated Working spares count,  "processors" *)
   0,              (* dedicated Failed spares count,  "processors" *)
   (NI_memories OF TRUE), (* Each working in "memories" *)
   (NI_memories OF FALSE), (* Each permanent failed in "memories" *)
   (NI_memories OF FALSE), (* Each transient faulty in "memories" *)
   NI_bus,         (* Count of working in "bus" *)
   0,              (* removed components *)
   0               (* Number of component failures for PRUNEIF *)
  );

ASSERT NTOT =
       COUNT(W_processors) +
       COUNT(FP_processors) +
       COUNT(FT_processors) +
       COUNT(W_memories) +
       COUNT(FP_memories) +
       COUNT(FT_memories) +
       NI_bus +
       NWS_processors +
       NFS_processors +
       TRASH_CAN_SIZE;

                      (************************************)
                      (************************************)
                      (***                              ***)
                      (***   COMPONENT: "processors"    ***)
                      (***                              ***)
                      (************************************)
                      (************************************)


(*
 *    Assertions for component "processors"
 *    -------------------------------------
 *)

ASSERT TA_processors >= TWA_processors;
ASSERT TA_processors >= TFA_processors;
ASSERT (TWA_processors + TFA_processors) <= NI_processors;
ASSERT (TWA_processors + TFA_processors) = TA_processors;


(*
 *    For each part in component "processors"
 *    ----------------------------------------
 *)

FOR ix IN [1..NI_processors]
    ASSERT A_processors(ix) OR (NOT W_processors[ix]);
    ASSERT COUNT(FP_processors[ix]) + COUNT(FT_processors[ix]) <= 1;
    ASSERT COUNT(W_processors[ix]) +
           COUNT(FP_processors[ix]) + COUNT(FT_processors[ix])
           <= 1;


    (*
     *    Fault Arrivals for component "processors"
     *    -----------------------------------------
     *)


    IF (W_processors[ix]) THEN  (* Fault arrivals *)

       (*
        *    Permanent fault arrivals
        *    ------------------------
        *)

       TRANTO W_processors[ix]=FALSE,FP_processors[ix]=TRUE,FT_processors[ix]=FT_processors[ix],
              NCF++
              BY L_P_processors;  (* Permanent fault arrival *)
    (*
     *    Transient fault arrivals
     *    ------------------------
     *)

    TRANTO W_processors[ix]=FALSE,FT_processors[ix]=TRUE,FP_processors[ix]=FP_processors[ix],
           NCF++
           BY L_T_processors;  (* Transient fault arrival *)
ENDIF;

IF (FT_processors[ix])  (* Transient faults go permanent *)
   TRANTO FT_processors[ix]=FALSE,FP_processors[ix]=TRUE,
          NCF++
          BY L_P_processors;  (* Transient to Permanent fault arrival *)

(*
 *    Transients disappear for "processors"
 *    -------------------------------------
 *)


IF (FT_processors[ix]) THEN  (* Transient faults *)
   TRANTO FT_processors[ix]=FALSE,W_processors[ix]=TRUE
          BY FAST DIS_processors;  (* Transient disappears *)
ENDIF;

(*
 *    Grab a spare for "processors" fault
 *    -----------------------------------
 *)


IF (FT_processors[ix]) THEN  (* Transient faults *)
   IF (NUC_processors > 0) THEN   (* Try dedicated spares for "processors" first *)
      IF (NWS_processors > 0)
         TRANTO NWS_processors--,FT_processors[ix]=FALSE,W_processors[ix]=TRUE,
                TRASH_CAN_SIZE++
                BY FAST (PRW_processors)*R_S_processors;
   ENDIF;
ENDIF;

IF (FP_processors[ix]) THEN  (* Permanent faults *)
   IF (NUC_processors > 0) THEN   (* Try dedicated spares for "processors" first *)
      IF (NWS_processors > 0)
         TRANTO NWS_processors--,FP_processors[ix]=FALSE,W_processors[ix]=TRUE,
                TRASH_CAN_SIZE++
                BY FAST (PRW_processors)*R_S_processors;
   ENDIF;
ENDIF;

(*
 *    Degrade component "processors"
 *    ------------------------------
 *)

IF (FA_processors(ix)) AND (TUC_processors = 0) THEN
   (*
    *    degrade by one
    *)
   IF (FP_processors[ix]) THEN
      IF (ix = 1) THEN
         IF (W_memories[1]) THEN (* able to: WRM(memories[1]) *)
            TRANTO FP_processors[1]=FALSE,
                   W_memories[1]=FALSE (* #1: REM(processors[1]) -> WRM.REM(memories[1]) *),
               TRASH_CAN_SIZE=TRASH_CAN_SIZE+2
               BY FAST R_D_processors;
    ENDIF;
    IF (FP_memories[1]) THEN (* able to: PRM(memories[1]) *)
       TRANTO FP_processors[1]=FALSE,
               FP_memories[1]=FALSE (* #1: REM(processors[1]) -> PRM.REM(memories[1]) *),
               TRASH_CAN_SIZE=TRASH_CAN_SIZE+2
               BY FAST R_D_processors;
    ENDIF;
    IF (FT_memories[1]) THEN (* able to: TRM(memories[1]) *)
       TRANTO FP_processors[1]=FALSE,
               FT_memories[1]=FALSE (* #1: REM(processors[1]) -> TRM.REM(memories[1]) *),
               TRASH_CAN_SIZE=TRASH_CAN_SIZE+2
               BY FAST R_D_processors;
    ENDIF;
    IF (TA_memories <= 0) THEN (* not able to: REM(memories[1]) *)
       TRANTO FP_processors[1]=FALSE,
               TRASH_CAN_SIZE++
               BY FAST R_D_processors;
    ENDIF;
 ELSE IF (ix = 2) THEN
    IF (W_memories[2]) THEN (* able to: WRM(memories[2]) *)
       TRANTO FP_processors[2]=FALSE,
               W_memories[2]=FALSE (* #2: REM(processors[2]) -> WRM.REM(memories[2]) *),
               TRASH_CAN_SIZE=TRASH_CAN_SIZE+2
               BY FAST R_D_processors;
    ENDIF;
    IF (FP_memories[2]) THEN (* able to: PRM(memories[2]) *)
       TRANTO FP_processors[2]=FALSE,
               FP_memories[2]=FALSE (* #2: REM(processors[2]) -> PRM.REM(memories[2]) *),
               TRASH_CAN_SIZE=TRASH_CAN_SIZE+2
               BY FAST R_D_processors;
    ENDIF;
    IF (FT_memories[2]) THEN (* able to: TRM(memories[2]) *)
       TRANTO FP_processors[2]=FALSE,
               FT_memories[2]=FALSE (* #2: REM(processors[2]) -> TRM.REM(memories[2]) *),
               TRASH_CAN_SIZE=TRASH_CAN_SIZE+2
               BY FAST R_D_processors;
    ENDIF;
    IF (TA_memories <= 0) THEN (* not able to: REM(memories[2]) *)
       TRANTO FP_processors[2]=FALSE,
               TRASH_CAN_SIZE++
               BY FAST R_D_processors;
    ENDIF;
 ELSE IF (ix = 3) THEN
    IF (W_memories[3]) THEN (* able to: WRM(memories[3]) *)
       TRANTO FP_processors[3]=FALSE,
               W_memories[3]=FALSE (* #3: REM(processors[3]) -> WRM.REM(memories[3]) *),
               TRASH_CAN_SIZE=TRASH_CAN_SIZE+2
               BY FAST R_D_processors;
    ENDIF;
    IF (FP_memories[3]) THEN (* able to: PRM(memories[3]) *)
       TRANTO FP_processors[3]=FALSE,
               FP_memories[3]=FALSE (* #3: REM(processors[3]) -> PRM.REM(memories[3]) *),
               TRASH_CAN_SIZE=TRASH_CAN_SIZE+2
               BY FAST R_D_processors;
    ENDIF;
    IF (FT_memories[3]) THEN (* able to: TRM(memories[3]) *)
       TRANTO FP_processors[3]=FALSE,
               FT_memories[3]=FALSE (* #3: REM(processors[3]) -> TRM.REM(memories[3]) *),
               TRASH_CAN_SIZE=TRASH_CAN_SIZE+2
               BY FAST R_D_processors;
    ENDIF;
    IF (TA_memories <= 0) THEN (* not able to: REM(memories[3]) *)
```

```
         TRANTO FP_processors[3]=FALSE,
                 TRASH_CAN_SIZE++
                 BY FAST R_D_processors;
      ENDIF;
   ENDIF; ENDIF; ENDIF;
ENDIF;
IF (FT_processors[ix]) THEN
   IF (ix = 1) THEN
      IF (W_memories[1]) THEN (* able to: WRM(memories[1]) *)
         TRANTO FT_processors[1]=FALSE,
                 W_memories[1]=FALSE (* #1: REM(processors[1]) -> WRM.REM(memories[1]) *),
                 TRASH_CAN_SIZE=TRASH_CAN_SIZE+2
                 BY FAST R_D_processors;
      ENDIF;
      IF (FP_memories[1]) THEN (* able to: PRM(memories[1]) *)
         TRANTO FT_processors[1]=FALSE,
                 FP_memories[1]=FALSE (* #1: REM(processors[1]) -> PRM.REM(memories[1]) *),
                 TRASH_CAN_SIZE=TRASH_CAN_SIZE+2
                 BY FAST R_D_processors;
      ENDIF;
      IF (FT_memories[1]) THEN (* able to: TRM(memories[1]) *)
         TRANTO FT_processors[1]=FALSE,
                 FT_memories[1]=FALSE (* #1: REM(processors[1]) -> TRM.REM(memories[1]) *),
                 TRASH_CAN_SIZE=TRASH_CAN_SIZE+2
                 BY FAST R_D_processors;
      ENDIF;
      IF (TA_memories <= 0) THEN (* not able to: REM(memories[1]) *)
         TRANTO FT_processors[1]=FALSE,
                 TRASH_CAN_SIZE++
                 BY FAST R_D_processors;
      ENDIF;
   ELSE IF (ix = 2) THEN
      IF (W_memories[2]) THEN (* able to: WRM(memories[2]) *)
         TRANTO FT_processors[2]=FALSE,
                 W_memories[2]=FALSE (* #2: REM(processors[2]) -> WRM.REM(memories[2]) *),
                 TRASH_CAN_SIZE=TRASH_CAN_SIZE+2
                 BY FAST R_D_processors;
      ENDIF;
      IF (FP_memories[2]) THEN (* able to: PRM(memories[2]) *)
         TRANTO FT_processors[2]=FALSE,
                 FP_memories[2]=FALSE (* #2: REM(processors[2]) -> PRM.REM(memories[2]) *),
                 TRASH_CAN_SIZE=TRASH_CAN_SIZE+2
                 BY FAST R_D_processors;
      ENDIF;
      IF (FT_memories[2]) THEN (* able to: TRM(memories[2]) *)
         TRANTO FT_processors[2]=FALSE,
                 FT_memories[2]=FALSE (* #2: REM(processors[2]) -> TRM.REM(memories[2]) *),
                 TRASH_CAN_SIZE=TRASH_CAN_SIZE+2
                 BY FAST R_D_processors;
      ENDIF;
      IF (TA_memories <= 0) THEN (* not able to: REM(memories[2]) *)
         TRANTO FT_processors[2]=FALSE,
                 TRASH_CAN_SIZE++
                 BY FAST R_D_processors;
      ENDIF;
   ELSE IF (ix = 3) THEN
      IF (W_memories[3]) THEN (* able to: WRM(memories[3]) *)
         TRANTO FT_processors[3]=FALSE,
                 W_memories[3]=FALSE (* #3: REM(processors[3]) -> WRM.REM(memories[3]) *),
                 TRASH_CAN_SIZE=TRASH_CAN_SIZE+2
                 BY FAST R_D_processors;
      ENDIF;
      IF (FP_memories[3]) THEN (* able to: PRM(memories[3]) *)
         TRANTO FT_processors[3]=FALSE,
                        FP_memories[3]=FALSE (* #3: REM(processors[3]) -> PRM.REM(memories[3]) *),
                        TRASH_CAN_SIZE=TRASH_CAN_SIZE+2
                        BY FAST R_D_processors;
                  ENDIF;
                  IF (FT_memories[3]) THEN (* able to: TRM(memories[3]) *)
                     TRANTO FT_processors[3]=FALSE,
                        FT_memories[3]=FALSE (* #3: REM(processors[3]) -> TRM.REM(memories[3]) *),
                        TRASH_CAN_SIZE=TRASH_CAN_SIZE+2
                        BY FAST R_D_processors;
                  ENDIF;
                  IF (TA_memories <= 0) THEN (* not able to: REM(memories[3]) *)
                     TRANTO FT_processors[3]=FALSE,
                        TRASH_CAN_SIZE++
                        BY FAST R_D_processors;
                  ENDIF;
               ENDIF; ENDIF; ENDIF;
            ENDIF;
         ENDIF;

ENDFOR;

(*
 *    Assertions for dedicated "processors" spares
 *    --------------------------------------------
 *)

ASSERT NUC_processors <= NS_processors;
ASSERT (NWS_processors + NFS_processors) <= NSI_processors;

IF (NWS_processors > 0) THEN (* Arrival of "processors" spare fault *)
   TRANTO NWS_processors--,NFS_processors++,NCF++
          BY NWS_processors*L_S_processors;
ENDIF;

(*
 *    General Death's for component "processors"
 *    ------------------------------------------
 *)

DEATHIF TFA_processors >= TWA_processors; (* MAJ(processors) *)

                        (***********************************)
                        (***********************************)
                        (***                             ***)
                        (***    COMPONENT: "memories"    ***)
                        (***                             ***)
                        (***********************************)
                        (***********************************)


(*
 *    Assertions for component "memories"
 *    -----------------------------------
 *)

ASSERT TA_memories >= TWA_memories;
ASSERT TA_memories >= TFA_memories;
ASSERT (TWA_memories + TFA_memories) <= NI_memories;
ASSERT (TWA_memories + TFA_memories) = TA_memories;


(*
 *    For each part in component "memories"
 *    -------------------------------------
*)

FOR ix IN [1..NI_memories]
    ASSERT A_memories(ix) OR (NOT W_memories[ix]);
    ASSERT COUNT(FP_memories[ix]) + COUNT(FT_memories[ix]) <= 1;
    ASSERT COUNT(W_memories[ix]) +
           COUNT(FP_memories[ix]) + COUNT(FT_memories[ix])
           <= 1;


     (*
      *    Fault Arrivals for component "memories"
      *    ---------------------------------------
      *)


     IF (W_memories[ix]) THEN  (* Fault arrivals *)

        (*
         *    Permanent fault arrivals
         *    ------------------------
         *)

        TRANTO W_memories[ix]=FALSE,FP_memories[ix]=TRUE,FT_memories[ix]=FT_memories[ix],
               NCF++
               BY L_P_memories;  (* Permanent fault arrival *)


        (*
         *    Transient fault arrivals
         *    ------------------------
         *)

        TRANTO W_memories[ix]=FALSE,FT_memories[ix]=TRUE,FP_memories[ix]=FP_memories[ix],
               NCF++
               BY L_T_memories;  (* Transient fault arrival *)
     ENDIF;

     IF (FT_memories[ix])  (* Transient faults go permanent *)
        TRANTO FT_memories[ix]=FALSE,FP_memories[ix]=TRUE,
               NCF++
               BY L_P_memories;  (* Transient to Permanent fault arrival *)

     (*
      *    Transients disappear for "memories"
      *    -----------------------------------
      *)


     IF (FT_memories[ix]) THEN  (* Transient faults *)
        TRANTO FT_memories[ix]=FALSE,W_memories[ix]=TRUE
               BY FAST DIS_memories;  (* Transient disappears *)
     ENDIF;

     (*
      *    Degrade component "memories"
      *    ----------------------------
      *)

     IF (FA_memories(ix)) THEN
        (*
         *    degrade by one
         *)
        IF (FP_memories[ix]) THEN
           TRANTO FP_memories[ix]=FALSE,
                TRASH_CAN_SIZE++
                BY FAST R_D_memories;
         ENDIF;
         IF (FT_memories[ix]) THEN
            TRANTO FT_memories[ix]=FALSE,
                TRASH_CAN_SIZE++
                BY FAST R_D_memories;
         ENDIF;
      ENDIF;

ENDFOR;

(*
 *    General Death's for component "memories"
 *    -----------------------------------------
 *)

DEATHIF TFA_memories >= TWA_memories; (* MAJ(memories) *)

                         (*******************************)
                         (*******************************)
                         (***                         ***)
                         (***    COMPONENT:  "bus"    ***)
                         (***                         ***)
                         (*******************************)
                         (*******************************)


(*
 *    Assertions for component "bus"
 *    ------------------------------
 *)

ASSERT TA_bus >= TWA_bus;
ASSERT TA_bus >= TFA_bus;


IF (NW_bus > 0)  (* Fault arrivals *)
   TRANTO NW_bus--,
          NCF++
          BY NW_bus*L_P_bus;  (* Permanent fault arrival *)

(*
 *    General Death's for component "bus"
 *    ------------------------------------
 *)

DEATHIF TFA_bus >= TWA_bus; (* MAJ(bus) *)

                         (*********************************)
                         (*********************************)
                         (***                           ***)
                         (***   MISC. DEATH & PRUNE     ***)
                         (***                           ***)
                         (*********************************)
                         (*********************************)

PRUNEIF NCF > Prune_After;
```

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704-0188

1. AGENCY USE ONLY *(Leave blank)*	2. REPORT DATE April 1994	3. REPORT TYPE AND DATES COVERED Technical Memorandum

4. TITLE AND SUBTITLE TOTAL User Manual	5. FUNDING NUMBERS WU 505-64-10-07
6. AUTHOR(S) Sally C. Johnson David P. Boerschlein	
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) NASA Langley Research Center Hampton, VA 23681-0001	8. PERFORMING ORGANIZATION REPORT NUMBER
9. SPONSORING / MONITORING AGENCY NAME(S) AND ADDRESS(ES) National Aeronautics and Space Administration Washington, DC 20546-0001	10. SPONSORING / MONITORING AGENCY REPORT NUMBER NASA TM-109101

11. SUPPLEMENTARY NOTES

Sally C. Johnson: Langley Research Center, Hampton, VA
David P. Boerschlein: Lockheed Engineering & Sciences Company, Hampton, VA

12a. DISTRIBUTION / AVAILABILITY STATEMENT Unclassified-Unlimited Subject Category 62	12b. DISTRIBUTION CODE

13. ABSTRACT *(Maximum 200 words)*

Semi-Markov models can be used to analyze the reliability of virtually any fault-tolerant system. However, the process of delineating all of the states and transitions in the model of a complex system can be devastatingly tedious and error-prone. Even with tools such as the Abstract Semi-Markov Specification Interface to the SURE Tool (ASSIST), the user must describe a system by specifying the rules governing the behavior of the system in order to generate the model. With the Table Oriented Translator to the ASSIST Language (TOTAL), the user can specify the components of a typical system and their attributes in the form of a table. The conditions that lead to system failure are also listed in a tabular form. The user can also abstractly specify dependencies with causes and effects. The level of information required is appropriate for system designers with little or no background in the details of reliability calculations. A menu-driven interface guides the user through the system description process, and the program updates the tables as new information is entered. The TOTAL program automatically generates an ASSIST input description to match the system description.

14. SUBJECT TERMS Reliability Analysis, Fault Tolerance	15. NUMBER OF PAGES 73
	16. PRICE CODE A04

17. SECURITY CLASSIFICATION OF REPORT Unclassified	18. SECURITY CLASSIFICATION OF THIS PAGE Unclassified	19. SECURITY CLASSIFICATION OF ABSTRACT	20. LIMITATION OF ABSTRACT