NASA Technical Paper 3452
Volume 3

HiRel: Hybrid Automated Reliability Predictor (HARP) Integrated Reliability Tool System (Version 7.0)

HARP Graphics Oriented (GO) Input User's Guide

Salvatore J. Bavuso, Elizabeth Rothmann, Nitin Mittal, and Sandra Howell Koppen

November 1994

NASA Technical Paper 3452
Volume 3

HiRel: Hybrid Automated Reliability Predictor (HARP) Integrated Reliability Tool System (Version 7.0)

HARP Graphics Oriented (GO) Input User's Guide

Salvatore J. Bavuso
Langley Research Center • Hampton, Virginia

Elizabeth Rothmann and Nitin Mittal
Duke University • Durham, North Carolina

Sandra Howell Koppen
Lockheed Engineering & Sciences Company • Hampton, Virginia

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

November 1994

This publication is available from the following sources:

NASA Center for AeroSpace Information
800 Elkridge Landing Road
Linthicum Heights, MD 21090-2934
(301) 621-0390

National Technical Information Service (NTIS)
5285 Port Royal Road
Springfield, VA 22161-2171
(703) 487-4650

Contents

Chapter 1

Introduction

The Hybrid Automated Reliability Predictor (HARP) integrated Reliability (HiRel) workstation tool system marks another accomplishment toward the goal of producing a totally integrated computer-aided design (CAD) workstation design capability (ref. 1). Since a reliability engineer must generally graphically represent a reliability model before solving it, the use of a graphical input description language increases productivity and decreases the incidence of error. The captured image displayed on a cathode-ray tube (CRT) screen serves as a documented copy of the model and provides the data for automatic input to the HARP reliability model solver. The introduction of dependency gates to fault tree notation allows large fault-tolerant system models to be modeled with a concise and visually recognizable and familiar graphical language. In addition to aiding in the verification of the reliability model, the concise graphical representation presents company management, regulatory agencies, and customers a means of expressing a complex model that is readily understandable.

The HARP program (ref. 2) consists of three main software programs that constitute a workstation capability: the graphics oriented (GO) program (ref. 3), the textual HARP program typically referred to as HARP or textual HARP, and the HARP Output (HARPO) program. All these programs execute on a VAX under VMS, SUN under UNIX, and IBM-compatible 286, 386, and 486 PC's under DOS. Textual HARP also executes under OS/2. The graphical input and output programs require the Graphical Kernal System (GKS) support modules with C and FORTRAN compilers, while textual HARP only requires an ANSI standard FORTRAN 77 compiler. All three programs are stand-alone programs that operate on compatible files. For example, files created under the PC environment can be executed by a VAX. In this way, a PC can be used as a workstation for input and output processing, while a VAX or some other workstation can be used for large computations.

Textual HARP when used as a stand-alone capability is compatible with a wide range of computing platforms because it was written in ANSI standard FORTRAN 77 for wide portability. Textual HARP has an interactive prompting input capability and is composed of three stand-alone programs: *tdrive*, *fiface*, and *harpeng*. As the user successively executes the programs in this order, files are created that are required by downstream programs. The programs also accept user-generated files created with a text editor. Thus, the user has the option to use the interactive capability or simply input the user-generated files.

The input to *tdrive* can also come from files generated by GO. The output of textual HARP is in the format of tabular structured files. These files can be used by HARPO to display the HARP tabular data in a wide variety of forms in an interactive mode.

The HARP Graphics Oriented (GO) program is the graphical user interface (GUI) to the HARP (ref. 2) program which is a member of the HiRel tool system. (See vols. 1 and 2 of this TP.) The GO program creates all required files that describe the user's reliability/availability models and provides an alternate input format to HARP's textual input language. The ASCII files can be modified with a text editor for additional flexibility. Before using the GO program, the user should understand the following HARP concepts: the fault-occurrence/repair model (FORM), fault/error handling model (FEHM), and the function of the HARP dictionary.

The GO program supports the creation of Markov chain models in two notations: directly as a Markov chain and indirectly as a fault tree. When a model is described as a fault tree, the

program *tdrive* automatically converts the fault tree to its equivalent Markov chain for solution. A Markov chain is solved directly if no FEHM's are specified; otherwise, for both notations the model's Markov chain is automatically altered to include the effects of user-specified fault/error handling. The model that is eventually solved by HARP is always a Markov chain.

The choice of which modeling language to use depends on several factors, such as the size of the model, the user's familiarity with a particular notation, and the degree of modeling detail desired. These factors are highly user dependent; however, for all but the most simple models (less than 50 states), the fault tree notation is by far the easiest to describe and comprehend. An important limitation of the fault tree notation is its inability to model systems with repair; however, there is no limitation for Markov chains. Thus, describing a repair model as a fault tree initially to automatically create its Markov chain may be possible. With a text editor, the *.int file can be subsequently altered to reflect the required repair transitions and repair rates. Direct input of a Markov chain will appropriately model systems with repair.

This document describes the GO program. Chapter 2 provides information on how to run the graphics program and gives an example session. Chapter 3 provides details about fault tree and Markov chain icons available for modeling, and chapter 4 describes the use for each function key. Chapter 5 gives detailed information about the output files GO created for input HARP. The appendix describes the hardware and software requirements for each computing platform.

Chapter 2

Program Initiation and Example Session

The initiation of the GO program may differ slightly on each computing platform. On a Sun Microsystems Sun 3 workstation, the Sunview environment must be invoked first. GO also executes on Sun 4 workstations under Sun OpenWindows. DEC VAX workstations also require a window environment.

On an IBM-compatible PC, if the GO program is in the path statement in autoexec.bat, typing `go` at the C prompt invokes the fault tree menus and `gom` invokes the Markov chain menus. The appendix provides the details.

To initiate the program on a Sun or VAX workstation, type the word `go` on your terminal, which gives you the default FORM of a fault tree. (You can also type `go f`.) If you want to create a Markov chain, type `go m`. (See appendix for parameter passing under the software requirements section.) You cannot switch back and forth between the two FORM types during program execution.

Your workstation requires a few seconds to set up the menus and screen. The first screen that appears is the credits screen. Pressing the left mouse button clears the screen for model entry. Your screen should then look like figure 1.

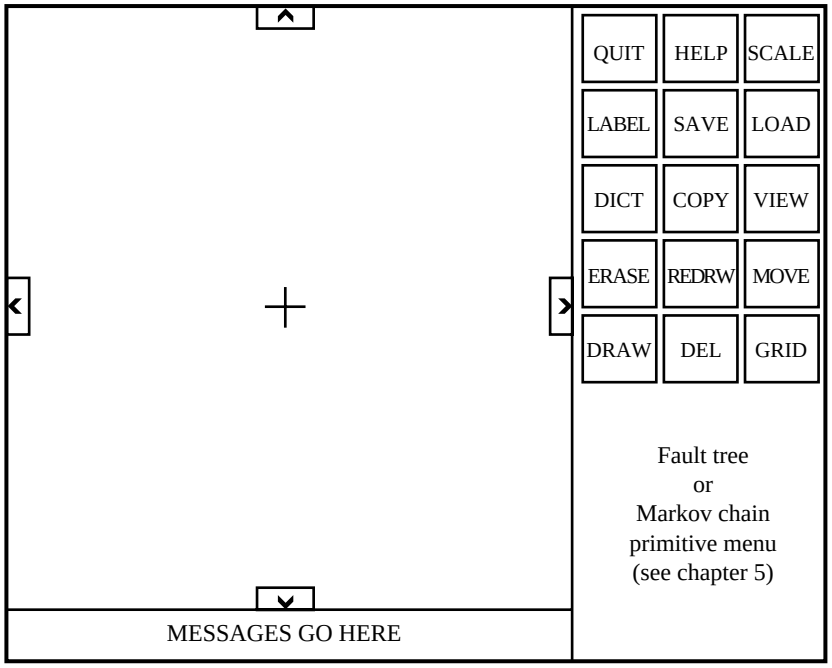


Figure 1. Initial drawing screen.

The drawing window is the large square to the left of the screen. In the middle of each edge of the drawing window is a small box with an arrow key. These SCROLL boxes allow you to scroll the picture up, down, left, or right. Directly below the drawing window is the message window

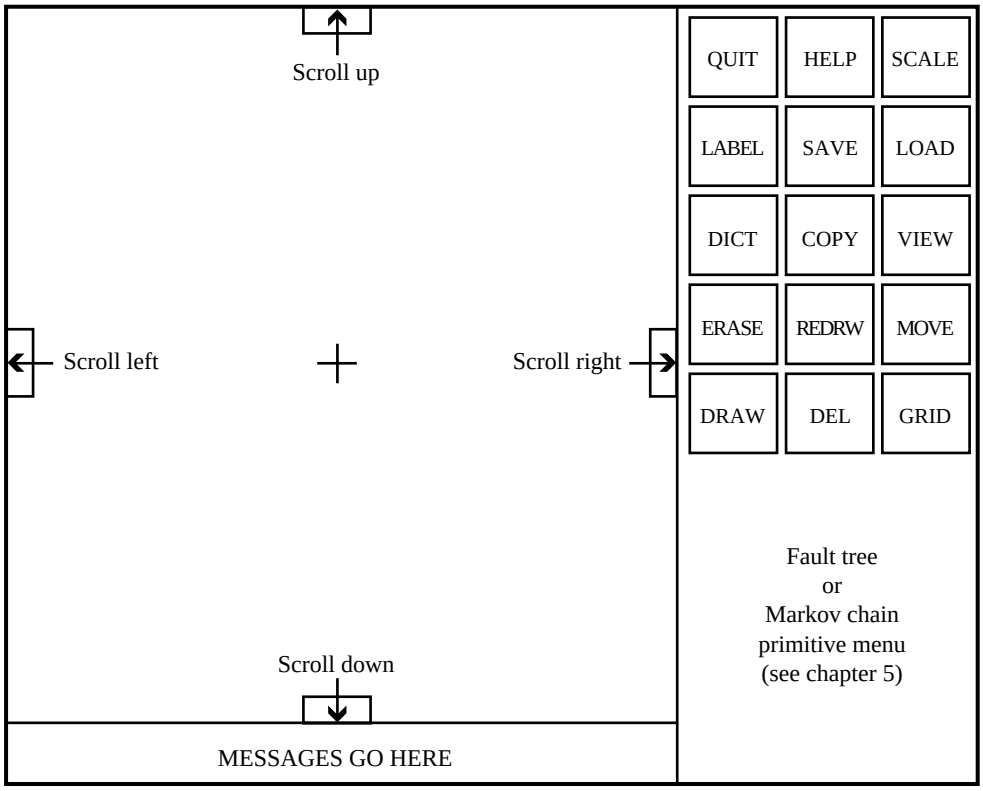


Figure 2. Screen scrolling buttons.

(`MESSAGES GO HERE`). User information, instructions, warnings, or error messages are displayed here. To the right of the drawing window is the menu area that includes the menu icons and the graphic primitive icons. The 15 menu icons shown have various function names, that is, QUIT, HELP, SCALE, etc. These menu icons are referenced herein as function keys. When the instructions call for you to use a particular function key, you should move the cursor over that icon with the mouse and press and release the left button on the mouse (tagging and dragging are not implemented). The graphic primitive icons are located below the function keys. If the FORM is a fault tree, there are eight function keys. If the FORM is a Markov Chain, there are four function keys. These icons are discussed in chapter 4.

The cursor is centered in the drawing window. Use the mouse to move the cursor in the direction desired. To make a selection, press the left mouse button while the cursor is over the selected area.

Keyboard input is not case sensitive, even though GO prompts may indicate otherwise. GO generated filenames are always uppercase but can be input as lowercase. GO executing on a Sun requires the user to select the keyboard input panel with the left mouse button. The panel is located below the message window.

2.1. Scrolls

The scroll buttons are located along each side of the main drawing window. (See fig. 2.) Anytime you want to scroll to a different screen, either to create a model or to view it, move the cursor over the correct box and press and release the left mouse button. To scroll a quarter screen, press and release the mouse button once. A number is written in the appropriate box representing the number of whole screens that have been scrolled in that direction. This scheme

makes returning to the home location easy. For example, if the screen has been moved two screens to the right, returning to home can be accomplished by pressing and releasing the mouse button eight times in quick succession.

2.2. Example Session

2.2.1. Model Input

To input a model in the GO program, perform the following steps:

1. Invoke GO by entering *go* or *go f* for inputting fault trees or *go m* for Markov chains. On a PC, the parameter *f* or *m* must be passed to the GO program. You must use a batch file, for example, go.bat or gom.bat, to pass the parameters to pcgo.exe and to invoke the GKS drivers. (See appendix under the section for PC software requirements.)
2. Clear the credits screen by pressing and releasing the left mouse button.
3. At any time when the menu is displayed, selecting HELP with the mouse displays context sensitive information. Menus are selected by moving the mouse to place the cursor over the menu item and pressing and releasing the left mouse button. To get help on drawing, select HELP then select DRAW from the menu. Selecting HELP then QUIT terminates help.
4. If this is an initial session, begin by selecting DRAW from the menu with the mouse (see chapter 4 for options). If this is a continuation of a previous session, then select LOAD from the menu.
5. After the primitives are displayed and before labeling the graph begins, you must create a dictionary file if the model is a fault tree. You can create this file by selecting DICT. If the model is a Markov chain without FEHM's (AS IS model) no dictionary is required. If FEHM's are used, a dictionary is required.
6. Before labeling can begin, you must save the model by selecting SAVE. Labeling can then proceed.
7. On completion of the model drawing, once again save the model before selecting QUIT. The session ends when QUIT is invoked.
8. Processing of the model data continues by executing *tdrive* unless an AS IS Markov chain is drawn. In this latter case, *fiface* is executed directly after GO termination. The programs *tdrive* or *fiface* prompt you to read existing files. Then, *tdrive* or *fiface* reads the files that GO created. These files (*.FTR and *.DIC for fault trees and *.MKV for Markov chains) must be available to *tdrive* or *fiface* for reading. Processing now proceeds identically to that of textual HARP. (See vol. 2 of this TP.)

2.2.2. Sample Session

After the credits screen is cleared (press the left mouse button), one of two screen images are displayed. Figure 1 appears if *go* or *go f* was invoked or figure 3 appears with a blank drawing window if *go m* was invoked. Since the more practical use of HiRel uses the fault tree notation, this session concentrates on its use. Markov chains are easier to draw than fault trees, but they become very large and untractable for most practical systems.

You can now select the context sensitive HELP screens by pointing the mouse cursor at HELP and pressing and releasing the left mouse button followed by selecting the function key of interest, for example, DRAW. Help is terminated by selecting QUIT. The purpose and use of

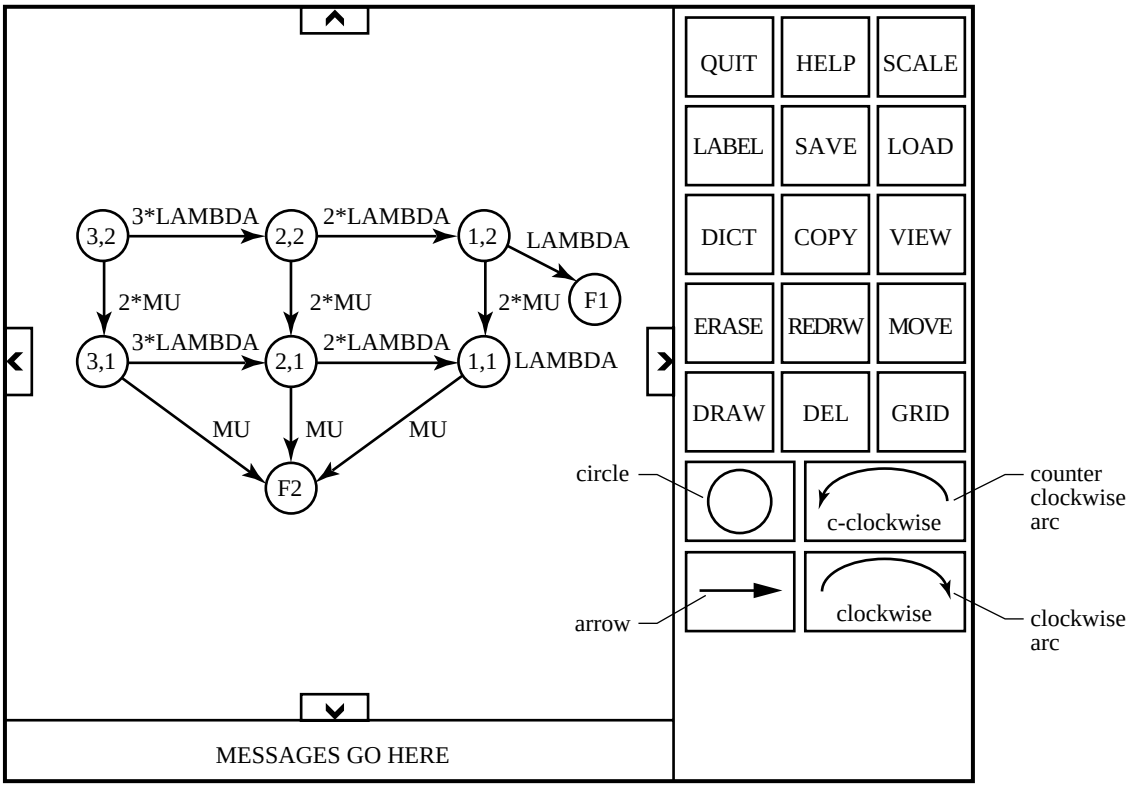


Figure 3. Markov chain primitive menu.

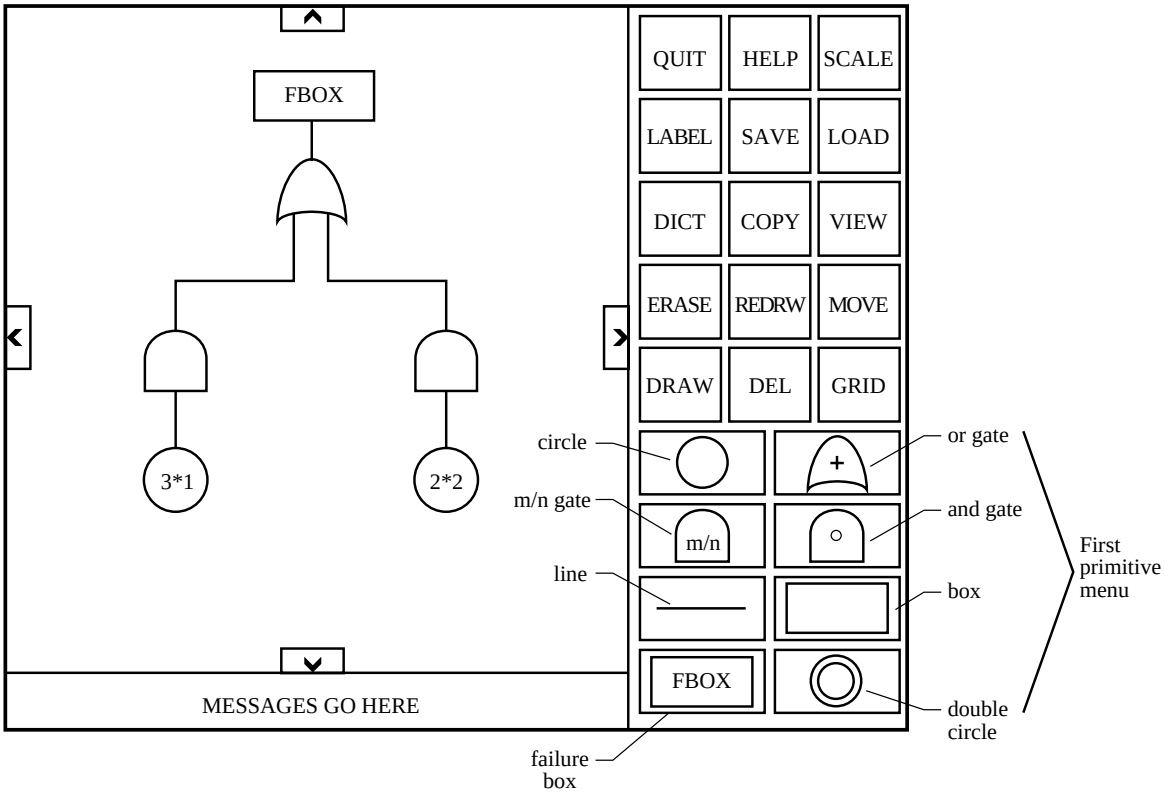


Figure 4. Default fault tree primitive menu.

all function keys except DRAW is straightforward. To demonstrate the drawing function, begin by selecting DRAW after terminating help.

The DRAW function key is now surrounded by a red border designating the selected function. Next point to the the function key to be placed on the screen. Figure 4 shows the fault tree that is currently being described. The order of selecting and placing the function keys is arbitrary with one exception, the line. Lines should be drawn last. To place the FBOX, place the cursor in the FBOX function key and press and release the left mouse button. Move the cursor to the desired position and press and release the left mouse button. The FBOX function key appears on the screen. Continue this process for all functions required. You need not select DRAW for each new function key in the primitive menu. The Circle function keys are empty when initially drawn. To draw lines, select line and select the source, then move the cursor to the destination and select it. Connect lines from the bottom of the screen upward. Figure 5 shows the second primitive menu that is obtained by pointing to the primitive menu and pressing and releasing the right mouse button. This menu is a toggle switch; thus, repeating the operation restores the initial menu.

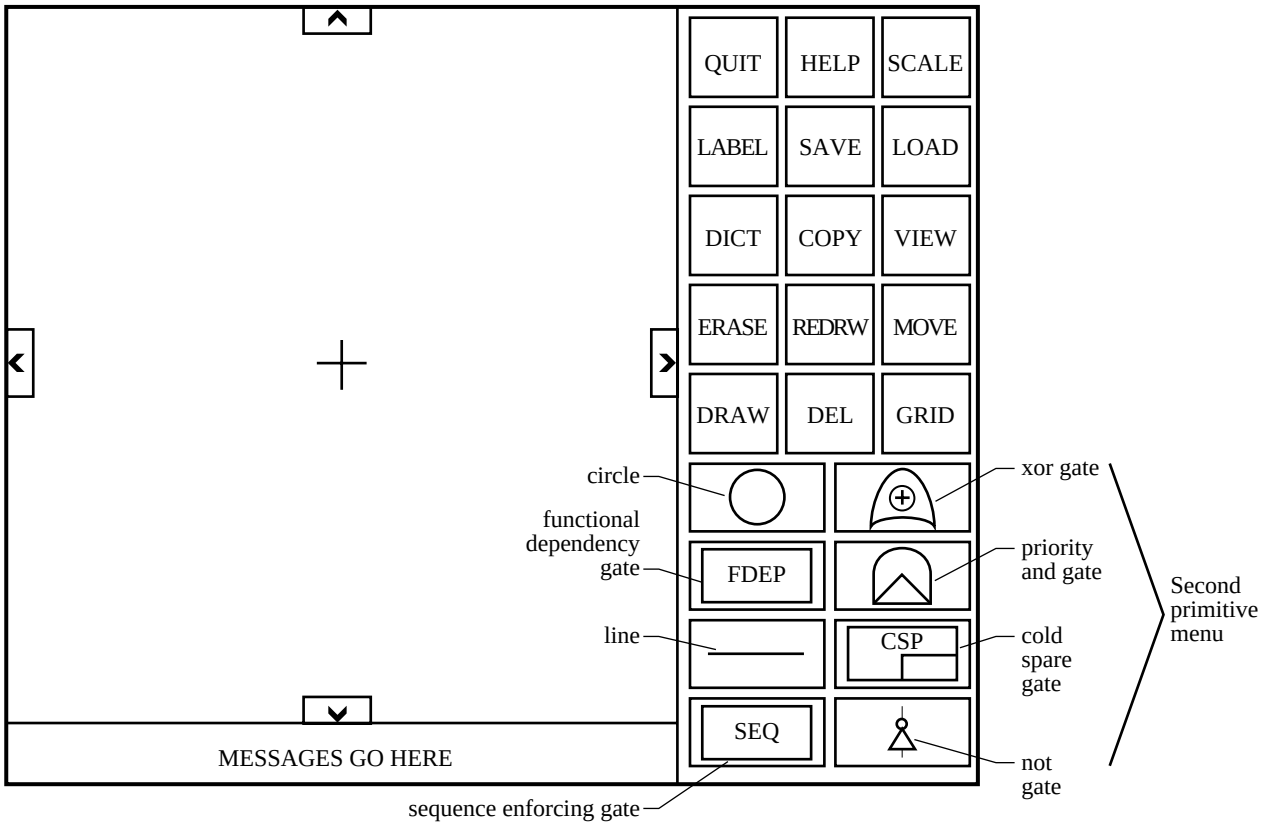


Figure 5. Second fault tree primitive menu.

The dictionary should be created next by selecting DICT. GO prompts you for the required information: the names of the system components, a symbolic failure rate name, and a FEHM model. GO assigns a unique positive integer to each component that is used later during the labeling of the basic event function keys.

When the dictionary is complete, the model needs to be temporarily saved. Use the SAVE function key to do that. When specifying the filename for the model, do not use a file extension. GO attaches an appropriate one for you. Now labeling can begin. When LABEL is selected,

GO places the dictionary integer associated with each component basic event in a circle function key. GO asks you for a replication factor, a positive integer. In figure 4, a replication factor of 3 was given for component 1 and a replication factor of 2 was given for component 2. Unlike textual HARP, you do not need to keep track of node numbers. GO does that bookkeeping for you.

At any time during the drawing operation, GRID, SCALE, SAVE, VIEW, REDRW, DEL, HELP, COPY, MOVE, and even QUIT or ERASE, can be selected. REDRW is useful for cleaning up the screen when much editing has been done. ERASE terminates the drawing but not the session. QUIT does both. In each case, you are warned of your requested action before it is executed. Typically, SAVE is selected before quitting. QUIT does not save automatically for you.

When the session is terminated, you should execute the *tdrive* program to continue the analysis. Then *tdrive* prompts you to enter the modelname of the system you drew. The analysis is now in the HARP environment and stays there until the HARPO program is invoked after *harpeng* is executed.

Chapter 3

Graphics Primitives

3.1. Markov Chain Primitives

Markov chains are stochastic mathematical models that constitute a superset of combinatorial fault tree models. Markov chains are particularly useful for modeling systems that contain various dependencies not easily modeled or at all possible with combinatorial fault trees. In contrast to combinatorial models, the solution of Markov chains is computationally more expensive. However, for many applications involving fault trees, the additional computer time is trivial compared with the benefits of the fault tree notation. When fault/error handling modeling is required, a Markov chain is often the only reasonable choice. The HiRel capability makes the choice of selecting a modeling notation much easier because the HARP program converts a fault tree into a Markov chain even when failure dependencies and fault/error handling are modeled. (See vol. 1 of this TP.)

Four graphics primitives are used to create a Markov chain: circle (representing a system state and also called a node), arrow, clockwise arc, and counterclockwise arc (representing state transitions). The connectors (arcs or arrows) must go from a node to a node. HARP can solve repairable systems; thus, no restrictions exist on having a connector going from node A to node B and another going from node B to node A. For clarity on the screen, you should not make both of these arrows because they will overwrite one another. Use the arcs primitives because they are curved rather than straight.

3.1.1. Circle

A circle represents a state in the Markov chain. To add (or delete) circles to your model (once you have used both the DRAW (or DEL) and the CIRCLE function keys), move the cursor to the desired location in the drawing window and press the left mouse button. When labeling the states, a parameter of 13 characters containing numerals or characters is allowed. If two nodes have the identical label, they are considered to be the same state. (This condition applies only if there are labels. Null states, states without labels, cannot be used in the remainder of the HARP program.)

3.1.2. Clockwise and Counterclockwise Arc

As the name implies, the clockwise arc is drawn clockwise from the head node to the tail node. Once the CLOCKWISE ARC function key has been highlighted, you should first specify the tail node from which the arc emanates and then specify the head node to which the arc goes. To specify a node, move the cursor anywhere within the node and press the left mouse button. You are given an error message if either node cannot be found. If the desired direction of the arc is counterclockwise, use the arc function key that shows the arc going in a counterclockwise direction with the word c-clockwise in the box. The tail node is again designated first and the head node second. However, the arc is drawn in a counterclockwise motion. While arcs look different on the screen than arrows (arcs are curved and arrows are straight), the arc is not interpreted any differently than the arrow during the run of the HARP engine. Arcs are useful for modeling repair transitions. Labels of up to 24 characters can be entered with the following restrictions:

1. Only one level of parentheses
2. Only addition and subtraction within parentheses
3. Only addition, subtraction, and multiplication outside parentheses

3.1.3. Arrow

The arrow connects two states in the Markov chain. Like an arc, it must have both a valid head and tail node to be entered. While it looks different on the screen (it is straight and arcs are curved), the arrow is not interpreted any differently than the arc during the run of the HARP engine. Arcs are used exclusively for repair transitions. After both the DRAW (or DEL) function key and the ARROW function key have been specified, designate the tail node by moving the cursor anywhere within it and press the left mouse button. Then, specify the head node in the same manner. Labels of up to 24 characters can be entered with the following restrictions:

1. Only one level of parentheses
2. Only addition and subtraction within parentheses
3. Only addition, subtraction, and multiplication outside parentheses

3.2. Fault Tree Primitives

The following 14 graphics primitives are used by the fault tree GO program: circle, or gate, and gate, m/n gate, priority and gate, functional dependency gate, cold spare gate, sequence-enforcing gate, double circle, box, failure box (FBOX), line, xor gate, and not gate. (All primitives except line are also called nodes.) These primitives are divided into two menus, as shown in figures 4 and 5. To view the second primitive menu, click the right mouse button. This button works as a toggle from one menu to the other.

3.2.1. Circle

The circle represents a basic failure event in the fault tree. It can have no incoming lines. To add (or delete) circles to your model (once you have enabled the DRAW (or DEL) and the circle function keys), move the cursor to the desired location in the drawing menu and press the left mouse button. When labeling the basic events, the dictionary file must exist. The label is the number of the corresponding dictionary. For example, if dictionary entry 2 corresponds to actuators and this basic event is an actuator, then the label is 2. HARP allows the combination of statistically identical components into single basic events. These replicated basic events are labeled with an expression of the form $m * n$, representing m replications of redundant, functionally identical components of type n. Therefore, you are asked how many components are represented by this particular basic event circle. (See vol. 1 of this TP.)

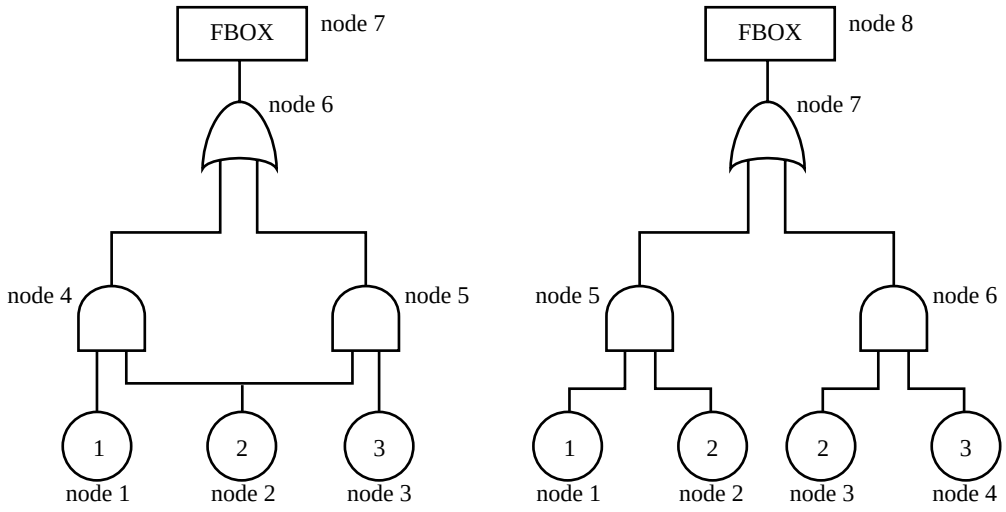
3.2.2. Double Circle

The double circle function key is used to simplify the graphics display. The double circle notation is provided for drawing convenience and to simplify the drawing by reducing connecting arcs. In a model, two or more basic event nodes (circles) of the same component type represent two or more distinct basic events. To represent a single basic event that is used more than once in the fault tree (shared event or common mode/cause failure) use the double circle function key. (See vol. 1 of this TP.) The shared basic event is initially drawn as a single circle. All other multiple occurring events associated with the initial basic event are then drawn as double

circles. The double circles are referenced to the initial single-circle basic event during the labeling process. When labeling commences, GO first prompts (red X) for the labels of all distinct basic events, if they were drawn first. GO then marks the double circle (red X) and prompts for the single-circle label corresponding to the double circle, etc. If a double circle is drawn before its corresponding basic event, GO asks whether the basic event was previously defined. If the answer is no and you respond accordingly, GO skips the double circle labeling and allows all basic events to be labeled. You should then select the LABEL function key to label the double circles.

When labeling the model, you are asked to identify the basic event node (circle) to which the double circle corresponds. The *.TRE and *.FTR files are generated by the GO program and represent the drawn model. The *.FTR or *.TRE files are read by program *tdrive*, which causes all lines that emanated from the double circle to emanate from the corresponding basic event. The double circle is removed from the internal data structure after the line connectors have been switched. This primitive keeps the fault tree neat and easy to follow. When DELETE and DOUBLE CIRCLE function keys are selected, GO warns you that all connecting arcs will also be deleted. At some time during the drawing session, a double circle must be associated with a basic event node. Also, a double circle must not be associated with other double circles.

The same reliability model can be obtained without the use of the double circles and can be used as a check to insure that the double circle model is correctly drawn. Volume 1 of this Technical Paper shows an example of a fault tree in figure 26. This figure is repeated in the following sketch. A double circle implementation of the fault tree on the left can be visualized by observing the fault tree on the right. Node 3 would be drawn as a double circle and hence would be assigned as node 2 internally by GO. GO would call node 4 node 3, and all following nodes would have a node value of one less than is depicted. All node numerals are internal to GO and are not displayed. Only the component type numbers are displayed in the circles. The disadvantage of not using the double circle drawing scheme is the need for displaying more arcs than the double circle technique.



3.2.3. or Gate

The or gate can have both incoming and outgoing connectors. The output is true if at least one input is true. For example, if an input is a basic failure event, then the or gate output is a failure event. The or gates are not labeled.

3.2.4. xor Gate

The xor gate can have both incoming and outgoing connectors. The output is true if an odd number of inputs is true. The xor gates are not labeled.

3.2.5. not Gate

The not gate has one incoming and one outgoing connector. The output is false when the input is true. The not gates are not labeled.

3.2.6. and Gate

The and gate can have both incoming and outgoing connectors. The output is true if all inputs are true. For example, if all inputs are basic failure events, then the and gate output is a failure event. Also, and gates are not labeled.

3.2.7. m/n Gate

The m/n gate can have both incoming and outgoing connectors. At least m out of the n events must occur in order for the output of the gate to occur. Both m and n must be specified during the labeling task.

3.2.8. Functional Dependency Gate

The functional dependency gate has one input (the trigger input), one or more dependent outputs, and a normal output. The dependent outputs are basic events that depend on the trigger event. When the trigger event occurs, the dependent basic events are forced to occur. The normal output at the top of the gate reflects the status of the trigger event. To connect a line from another primitive to the trigger event, point to the area near the tip of the FDEP gate stub to the left of the gate and click the left mouse button. In the PC version, the cursor is a plus (+) sign. Aim the center of the cursor at the stub's end and do not allow any part of the cursor to touch the box. Doing so directs the connection away from the stub. To connect the dependent events, click the left mouse button when the cursor is inside the functional dependency node. The ordering of the displayed dependent events is unimportant as the trigger event causes all dependent events to occur.

3.2.9. Priority and Gate

The priority and gate has both incoming and outgoing connectors. It is essentially an and gate with two inputs with the added restriction that the input events have to occur in order. If the two inputs are A and B, then the gate fires if both A and B occur and A occurs before B. When drawing the two input lines, draw the left one first then the right one. Only one line entering the gate is shown on the screen; thus, it is important to get the order of input correct the first time. Verifying the order is impossible after they are drawn. ASCII files MODELNAME.FTR or MODELNAME.TRE can be viewed for that information after quitting GO.

3.2.10. Cold Spare Gate

The cold spare gate has both incoming and outgoing connectors. It has one primary input (the functional unit) and one or more secondary inputs (the cold spare units). The gate produces an output when all input events (spare failures) have occurred. To connect the secondary inputs, click the left mouse button on the line extending from the bottom of the node. To connect the

primary event, click the left mouse button when the cursor is inside the cold spare node (larger rectangle). GO orders cold spare events based on the left-to-right order of the displayed nodes (spare basic event circles) and NOT connections. Although figure 3 in volume 1 of this Technical Paper shows the incoming arcs as separate arcs, GO implemented the connections differently. A separate arc is drawn for the primary unit, but only one arc is drawn for the spare units. Arcs from the spare basic event circles will merge into one stub entering the cold spare gate primitive. Since the arcs merge into one, distinguishing the order of the incoming arcs is visually impossible. Thus, the order of the basic event circles and not the arcs is used to determine the order of the spare event failures.

3.2.11. Sequence-Enforcing Gate

The sequence-enforcing gate has more than one input connector and an output connector. The input events are constrained to occur in the left-to-right order in which they appear under the gate; that is, the leftmost event must occur before the event on its immediate right is allowed to occur. The GO program orders the input events based on the relative locations of the circles representing basic events and not on the order of incoming connectors (see the cold spare gate explanation for this scheme). For the sequencing-enforcing gate, all inputs merge into one stub that enters the primitive.

3.2.12. Box

A box node represents a subsystem tree output (a subsystem failure event) and is labeled by text describing the subsystem (not yet implemented in the graphics or HARP packages). Boxes are usually placed at the outputs of gates to serve as comments.

3.2.13. Failure Box

The failure box node (FBOX) represents system failure. It must be the top node of the fault tree. If it is omitted, a warning is displayed when you save the file. The *tdrive* program will not complete without this node present. Only one FBOX is allowed.

3.2.14. Line

The LINE function key connects two nodes (nonline primitives) in the fault tree. Lines must be entered from the node lower in the tree to the higher one. A line cannot go between two basic events; basic events can only be connected to other node types.

Chapter 4

Function Keys

4.1. HELP

Help prints out instructions on how to use the various function keys. To obtain information on any function, first select the HELP key, and then select the function key designating the desired information. (See fig. 6.) To exit the help session, use the QUIT function key. As an example, suppose you want information on the GRID function key, then the COPY function, and then you want to leave the HELP session. Perform the following steps:

1. Select the HELP key with the mouse
2. Select the GRID key with the mouse
3. Select the COPY key with the mouse
4. Select the QUIT key with the mouse

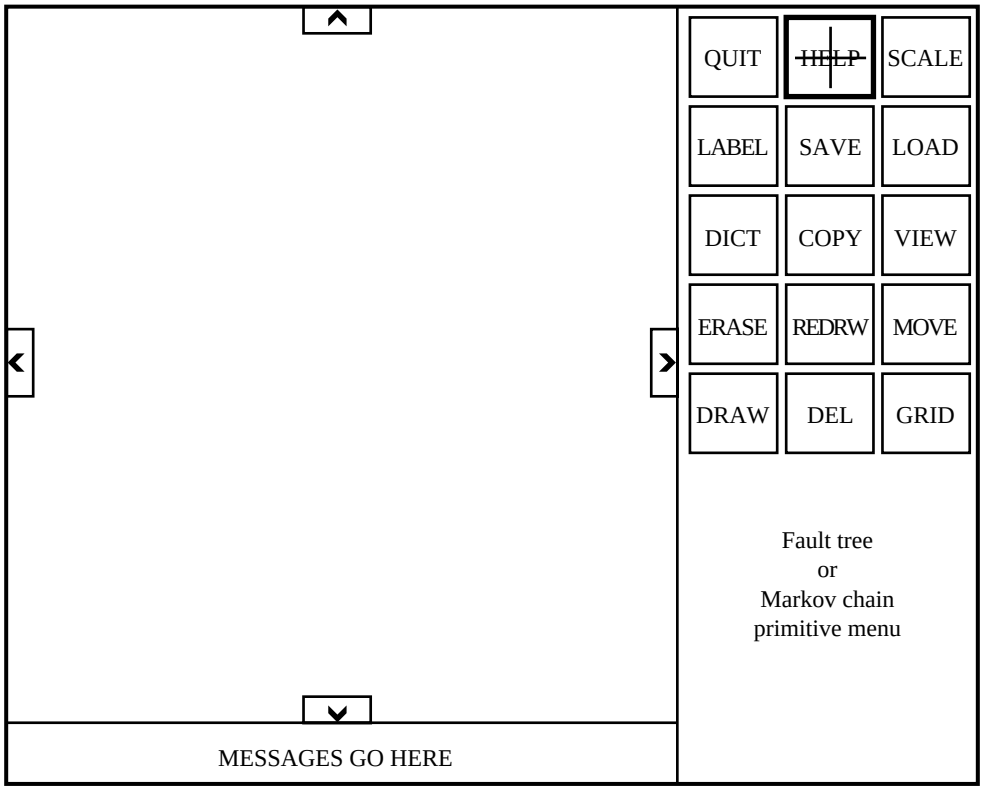


Figure 6. Selecting HELP from the menu.

If you accidently selected DRAW or DEL first to get help and you do not want to draw a primitive, select REDRW to abort DRAW. Now select HELP then DRAW, etc.

4.2. LOAD

If a model had been previously created and saved, select LOAD function key to restore the model on the screen. (See fig. 7.) If this is a new session and a new model is to be drawn, skip this command and proceed to DRAW.

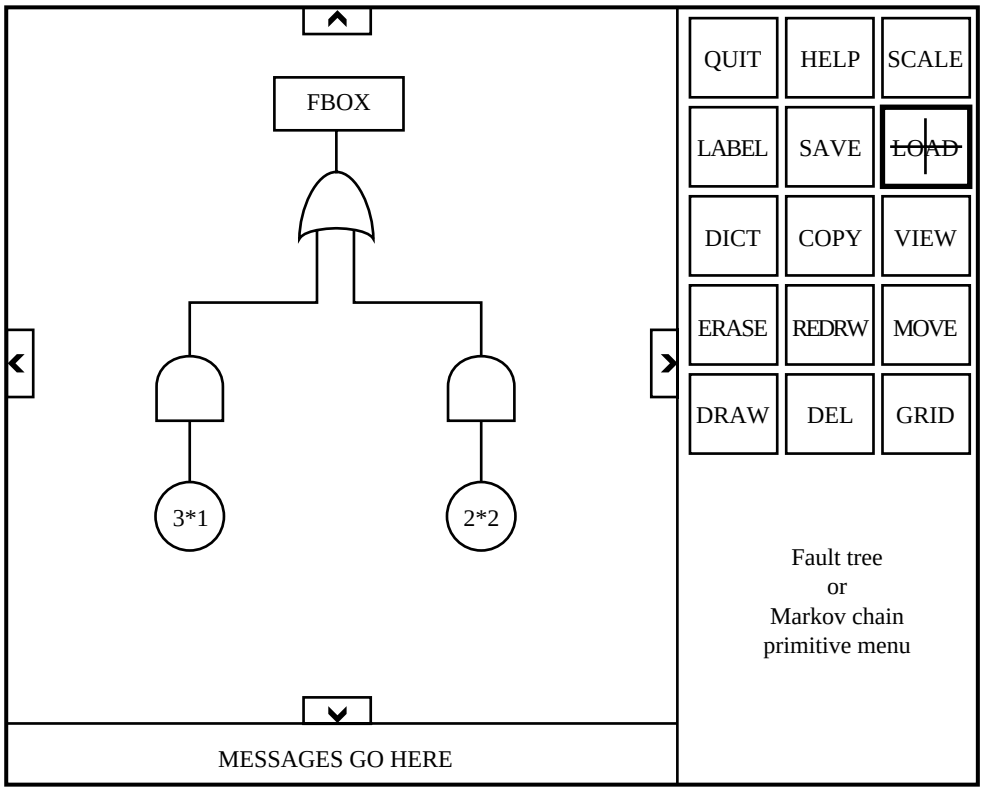


Figure 7. Selecting LOAD from the menu.

LOAD reads a model into the data structure and displays it on the screen. When prompted for the model name, type in the name with or without the extension. (If used, the extension must be .TRE for a fault tree or .MKV for a Markov chain.) If another model was already in the data structure, you are given the option of returning to it or using the new model. Once you decide, the model not selected is removed from the data structure.

4.3. DRAW

The DRAW function key allows you to create a model or modify an existing model that was invoked by the LOAD command. (See fig. 8.) To draw primitives in your model (e.g., a circle), first select the DRAW function key by clicking the left mouse button and then select the graphics primitive function key (CIRCLE) with the mouse using the same technique as before. Notice that both the DRAW function key and the CIRCLE key are highlighted. If they are not, you probably did not select the DRAW key first.

Once the primitive has been chosen, simply move the cursor into the drawing window and place the cursor where you want the primitive to appear and press the left mouse button. The primitives are copied (not dragged) into the drawing area. To enter more of the same primitive, move the cursor in the drawing area and press the left mouse button—you do not need to select

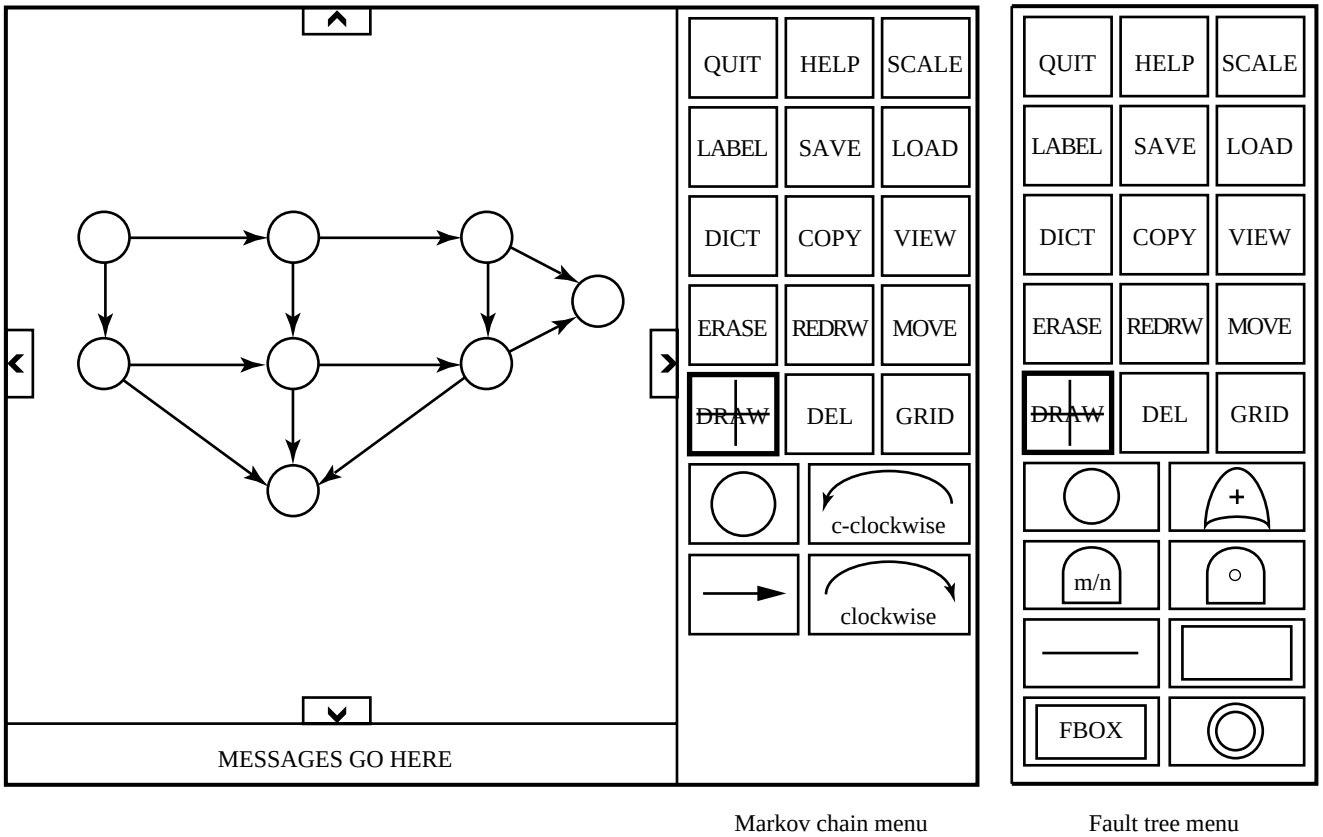


Figure 8. Selecting DRAW from the menu.

the primitive function key again. If you want to enter a different primitive, simply move the cursor over that graphics primitive function key and press the left mouse button (the DRAW function key remains selected). Then repeat this process in the drawing window. Until you select one of the other main function keys, you can add as many different primitives to your model as you like. If the DRAW function key is selected and you want to abort the selection without selecting a primitive, select REDRW.

4.4. DICT

The DICT function key is used to enter the dictionary. (See vol. 1 of this TP for information on the dictionary.) If the model type is a fault tree, the dictionary file must be created. To create the dictionary, first select the DICT function key. (See fig. 9.) You are prompted for the component type name, its failure rate, and the FEHM filename. Help is available by entering a <CR> at the prompt. If the FEHM parameter file does not exist, you can create one for any of the allowed HARP FEHM's. The ARIES, CARE, and ESPN models are displayed graphically as well.

If you make an error anywhere in the current component description, you can return to the beginning of that component description by selecting Escape <CR> for the PC DOS version and selecting \ <CR> for the Sun and VAX VMS versions at any of the prompts. This action returns you to the prompt for the component name. You cannot jump from one component description to another. To change a previous description, the dictionary file *.dic must be edited, or the entire dictionary file may be recreated. Once you have entered all the component information,

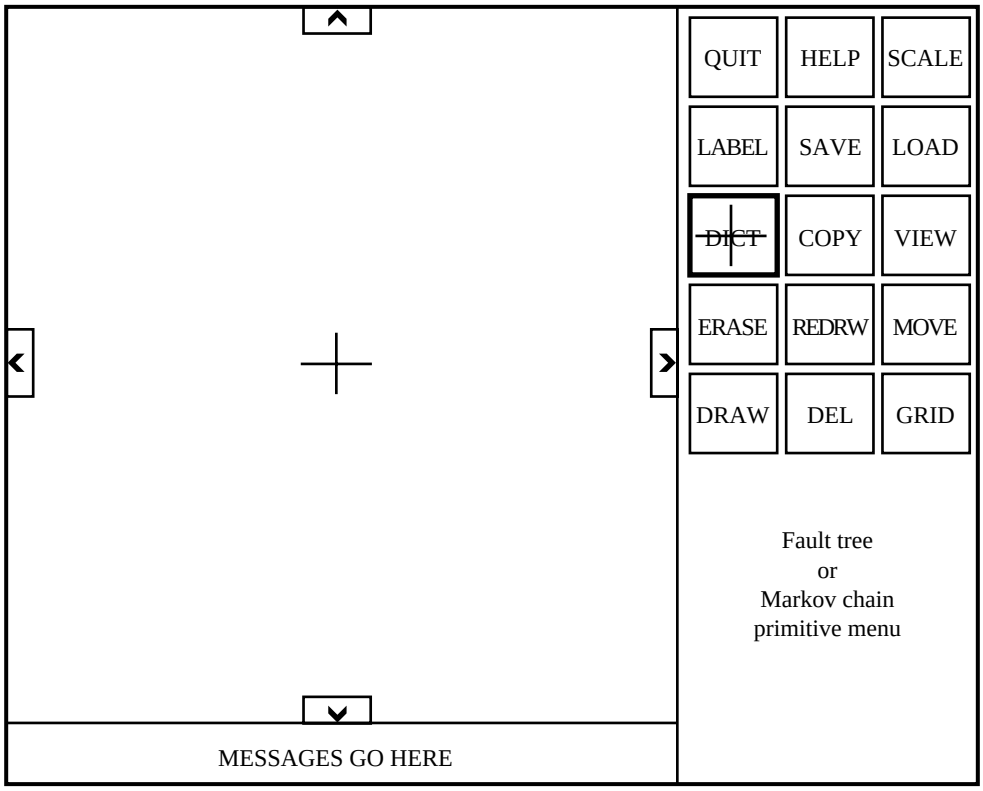


Figure 9. Selecting DICT from the menu.

type \, \d, or /d to terminate. If you have entered FEHM filenames other than NONE or VALUES, you are also asked about the user-defined near-coincident fault rates.

The deletion of a basic event node using the DEL function key doesn't remove its entry from the dictionary. No matter how many basic events are deleted, adding another dictionary item proceeds from the last one entered. This scheme precludes the renumbering of the labels as well as the dictionary entries each time a basic event is deleted. Unused dictionary entries do not cause any problems with the program.

4.5. LABEL

The LABEL function key is used to label the nodes, and if the model type is a Markov chain, it is used to label the connectors. (See fig. 10.) You are first asked whether you want to change existing labels or enter new labels. In either case, you are prompted for a label by a red X in or on each graphics primitive in the drawing window. To move the red X from one node to the next node without making any entries or changes, simply hit <CR>. A model must be saved before it can be labeled. If you are labeling a fault tree, you are prompted for the dictionary entry number. The prompt also allows you to enter a ? for help so that you can see the dictionary. Enter \, \d, or /d to exit labeling, or enter c if you forgot to create a dictionary before entering labeling. If you are labeling a Markov chain, you are prompted for the state name. Enter the label on the keyboard, maximum of 13 characters, and press Enter or <CR>. You are then prompted for the next one. When labeling, you are first asked for all the node labels, then the connectors. To quit the labeling session before all the labels have been entered, type \, \d, or /d then <CR>. When selecting *change existing*, if you do not want to change the red X, press <CR> to select the next basic event for editing. Basic events must be displayed on the screen

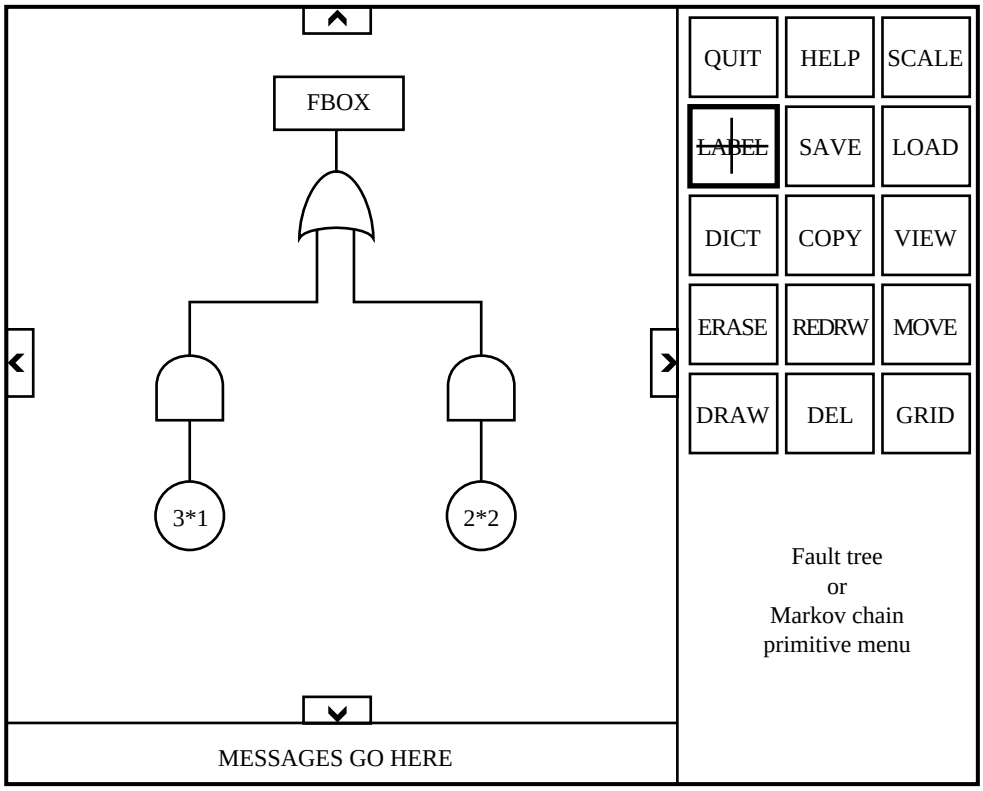


Figure 10. Selecting LABEL from the menu.

to do labeling. Labeling details for each graphic primitive are found in their respective graphic primitive description sections.

HARP places the following restrictions on rate parameters, which only affect the Markov chain input:

1. Only one level of parentheses
2. Only addition and subtraction within parentheses
3. Only addition, subtraction, and multiplication outside parentheses

Greek symbols, new line markers, and subscripts must be entered with an escape code invoked by the \ key. To enter any Greek symbol, preface the Greek name with a \ and also add a \ suffix. The way to enter λ is

\lambda\

All Greek symbols are available in lowercase only. The symbols \- within the label signify a new line. All characters entered following \S and before \U are written as subscripts.

In a Markov chain, two nodes having an identical label are considered to be the same state. (This condition applies only if there are labels. However, to run the *fiface* and *harpeng* programs, the states must be labeled.)

4.6. SAVE

The SAVE function key allows you to write your model to a disk file. (See fig. 11.) When prompted for the model name, do not enter an extension. If the model type is a fault tree,

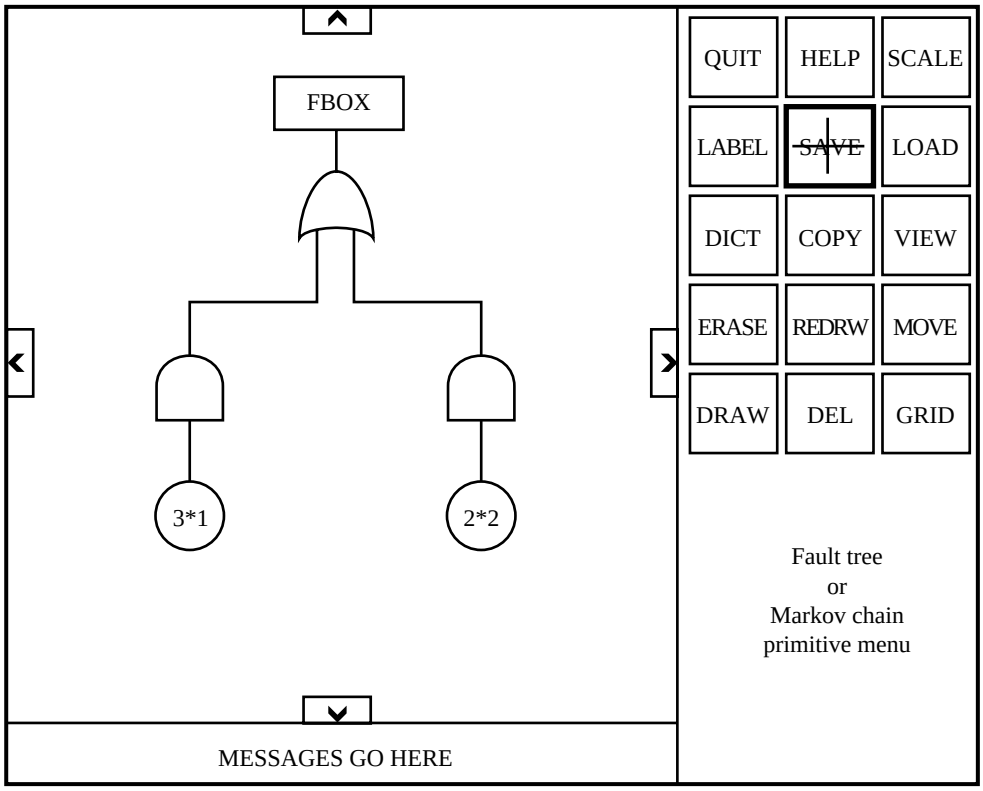


Figure 11. Selecting SAVE from the menu.

extensions of .TRE and .FTR are appended. For a fault tree, a *.FTR file is created to be used by *tdrive*. It is identical to the *.TRE file unless double circle graphics primitives are used. (See section 3.2.2 for details on this primitive.) If the type is a Markov chain, the extension .MKV is appended. Be sure to save a labeled model before quitting a session. Your session will still continue after saving the model.

4.7. QUIT

The QUIT function key is used to exit the current session. (See fig. 12.) After selecting QUIT, a prompt appears that gives you a chance to change your selection and continue or to really quit the session. Be sure to save your model before you quit because it is lost upon exiting. If you enter Y to really quit, GKS is terminated and your screen returns to normal.

4.8. COPY

The COPY function key allows you to copy portions of your model from one part of the screen to another. (See fig. 13.) Consider the area to be copied as a rectangle. After selecting COPY, you are prompted for the lower left corner and the upper right corner of the area to be copied. Using these two points, a box is drawn around the portion of the model to be copied and the area within the box is copied to the location specified. Only one drawing area screen can be copied at a time. However, it can be copied to another screen segment; that is, you can use the scroll keys to get to the location you want the model copied.

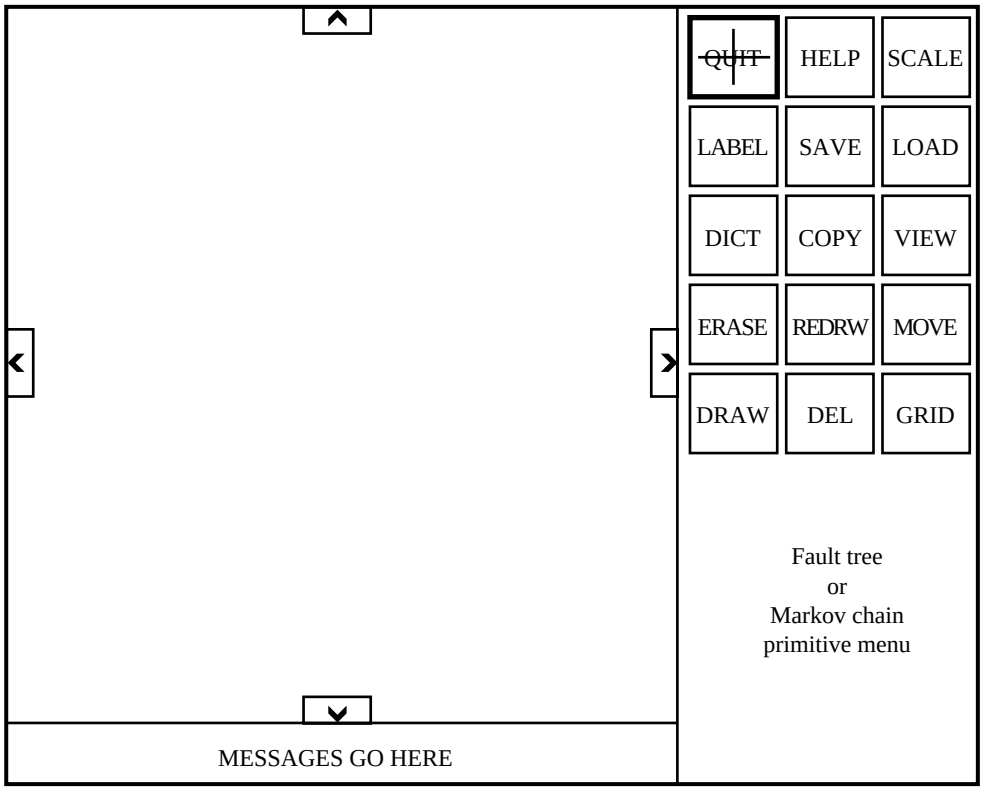


Figure 12. Selecting QUIT from the menu.

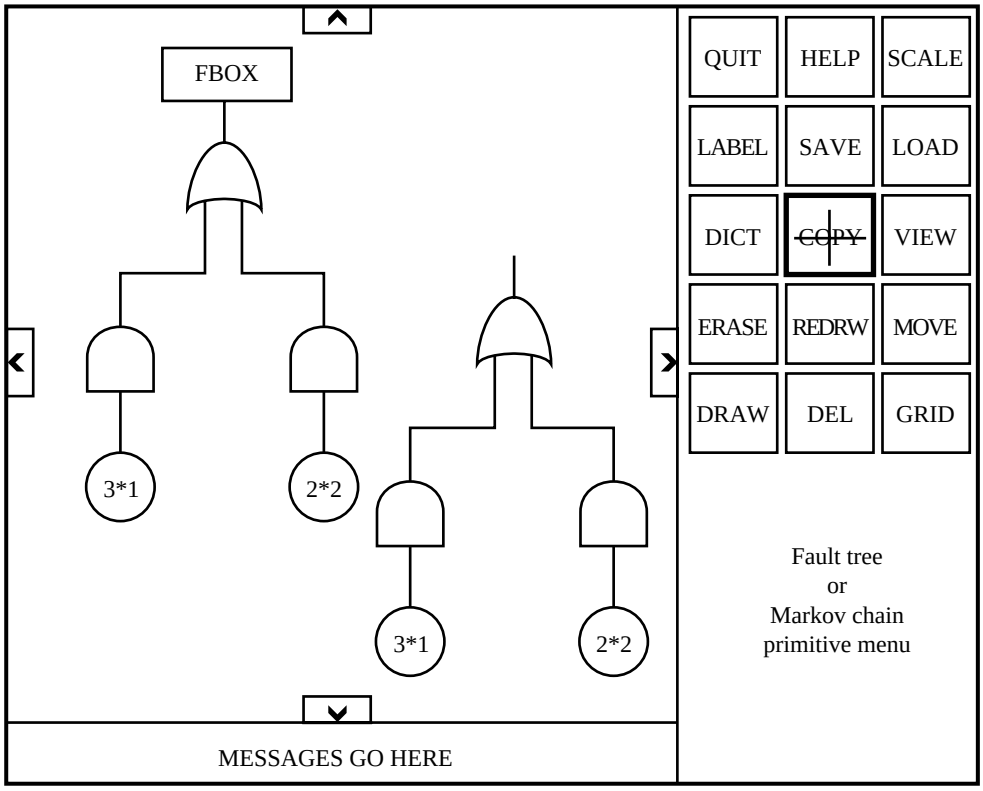


Figure 13. Selecting COPY from the menu.

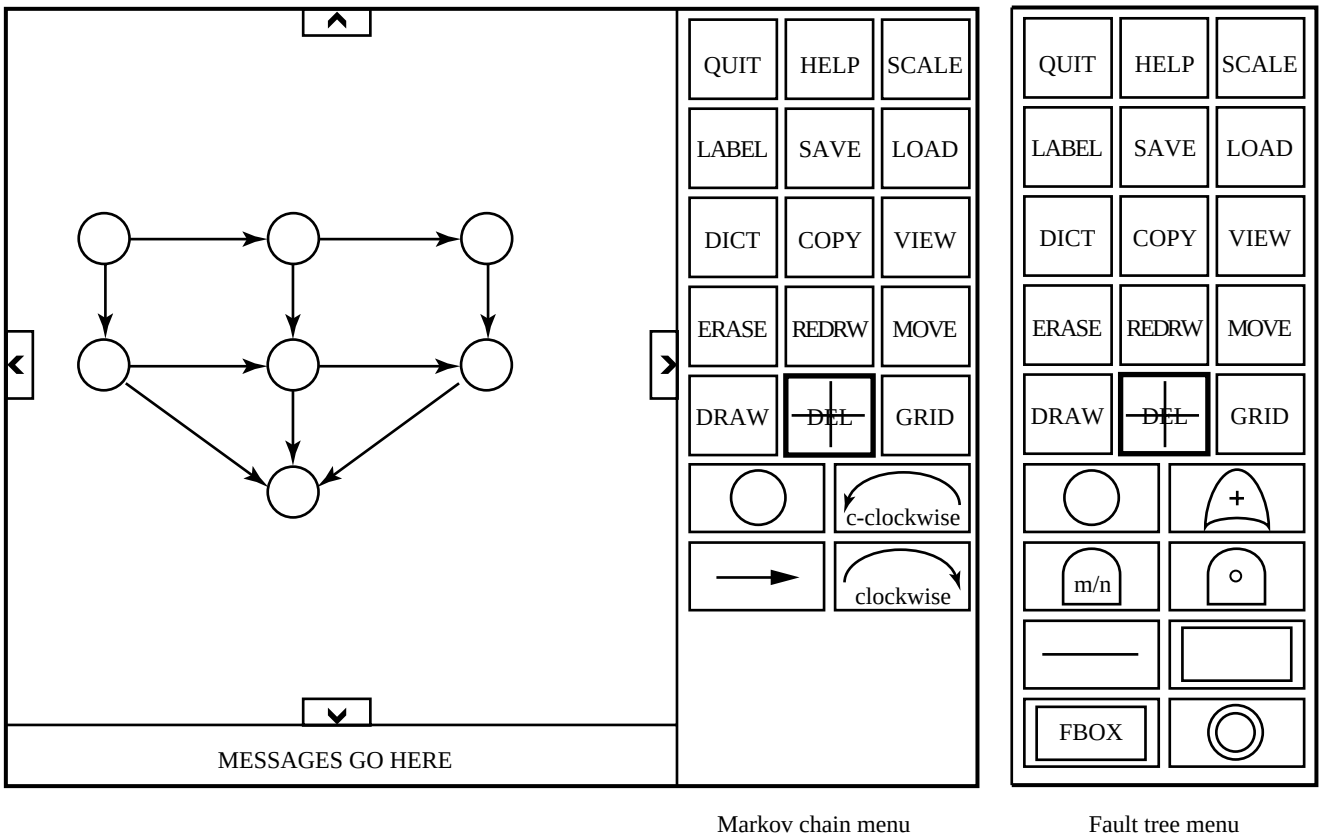


Figure 14. Selecting DEL from the menu.

4.9. DEL

The DEL function key is used to remove primitives from your model. (See fig. 14.) To abort a DEL selection prior to selecting a primitive, select REDRW. To delete a primitive from your model, first select DEL and then select the graphics primitive in the menu.

Both the DEL function key and the graphics primitive function key should be highlighted. If they are not, you probably did not select the DEL key first. Once the primitive has been chosen, simply move the cursor into the drawing window and place the cursor within the primitive that you want to delete, then press the left mouse button. To delete more of the same primitive, move the cursor to the primitive in the drawing window and press the left mouse button—you do not need to enable the primitive function key again. If you want to delete a different primitive, select the graphics primitive from the menu with the mouse (the DELETE function key remains enabled), then repeat the process in the drawing window. Until you select one of the other function keys, you can delete as many different primitives in your model as you want. However, with the exception of the circle and double circles, you cannot delete a node if there are any connectors entering or leaving it. The connectors must first be removed. If a label is associated with the node or connector, it is removed. You can delete circles or double circles like any other primitive; however, you are warned that all connectors will also be deleted.

The deletion of a basic event node using the DEL function key doesn't remove its entry from the dictionary. No matter how many basic events are deleted, adding another dictionary item proceeds from the last one entered. This scheme precludes the renumbering of the labels as well

as the dictionary entries each time a basic event is deleted. Unused dictionary entries do not cause any problems with the program.

4.10. ERASE

The ERASE function key not only erases the model on the screen but also destroys it in the data structure. (See fig. 15.) Once this function is used, you cannot retrieve your model unless it was previously saved on disk.

4.11. GRID

If you want a grid in the drawing window screen, use the GRID function key. (See fig. 16.) If the grid is on, invoking the GRID function key turns the grid off. Regardless of the grid being visible or not, nodes in the model are drawn to the nearest half-grid location. The grid size is fixed, and drawing in scaled mode is prohibited.

4.12. MOVE

The MOVE function key allows you to move any node in the model to another location. (See fig. 17.) You are prompted for the node to move (place the cursor anywhere within the node and press the left mouse button) and where to move it (specify the the location in the drawing window with the cursor and press the left mouse button). Any connectors—both in and out of the node—are altered to reflect the new location of the node as well.

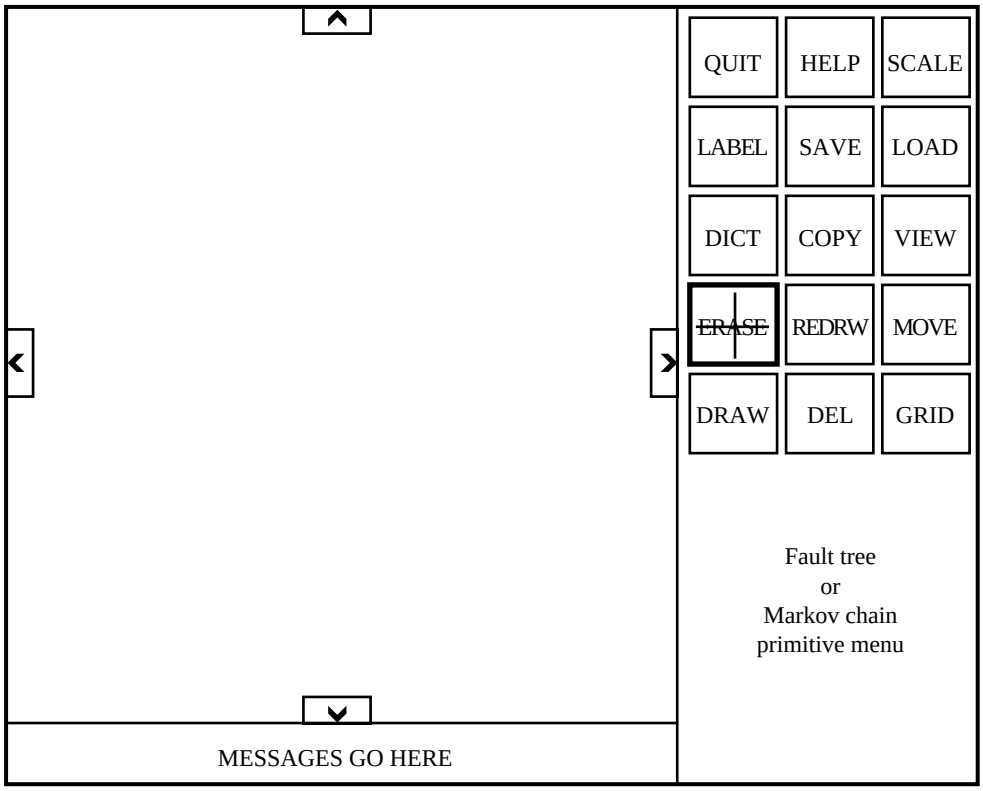


Figure 15. Selecting ERASE from the menu.

4.13. REDRW

To redraw the model on the screen, use the REDRW function key. (See fig. 18.) All nodes, connectors (arcs, arrows, or lines), and labels are redrawn. This function does not alter the data

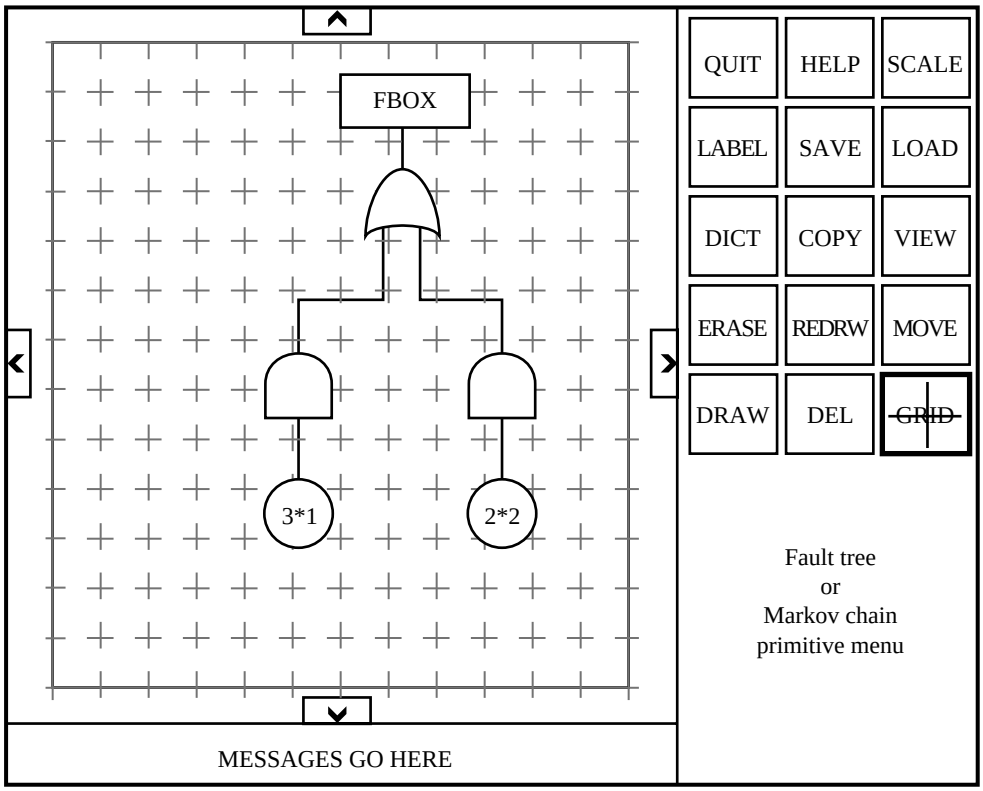


Figure 16. Selecting GRID from the menu.

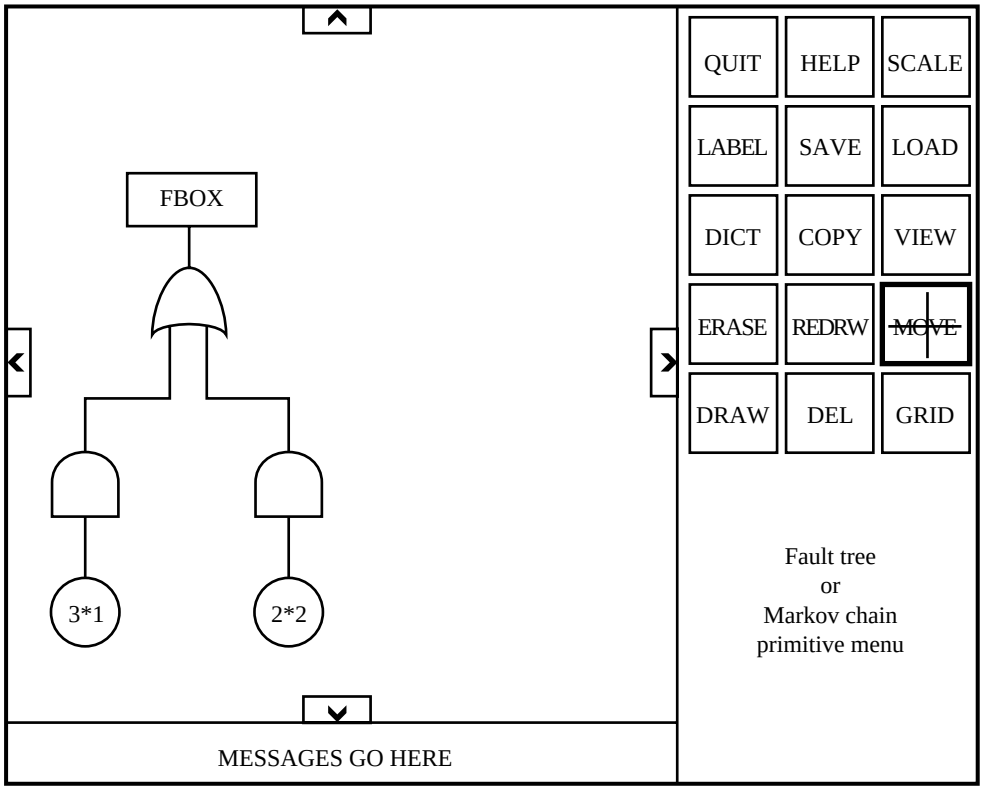


Figure 17. Selecting MOVE from the menu.

structure in any way. After an editing session, you may want to redraw your model because parts of desired lines or symbols can be erased from the screen as other undesired ones are deleted.

4.14. SCALE

The SCALE function key is used to scale the model either bigger or smaller. (See fig. 19.) You are prompted for a scaling factor that can be any real number greater than 0 and less than or equal to 3.0. When using numbers greater than 1.0, you may not see anything on the screen once it is redrawn because you have magnified the model such that it is no longer visible on the screen. Scale the model again with a smaller value. If your model is currently scaled and you use the COPY, MOVE, LABEL, DRAW, or DEL functions keys, the scale is changed to 1.0 and the model is redrawn on the screen. Drawing in a scaled mode is prohibited.

4.15. VIEW

The VIEW function key is used to show the entire model in a small window in the lower right corner of the drawing area. (See fig. 20.) The model is centered, scaled, and drawn in the window. The portion of the model currently being displayed in the main drawing area window is highlighted by a red box. If you have not created any portion of the model in more than one drawing screen window (i.e., you have not used the SCROLL function keys and then created more of the model), the VIEW function is not necessary.

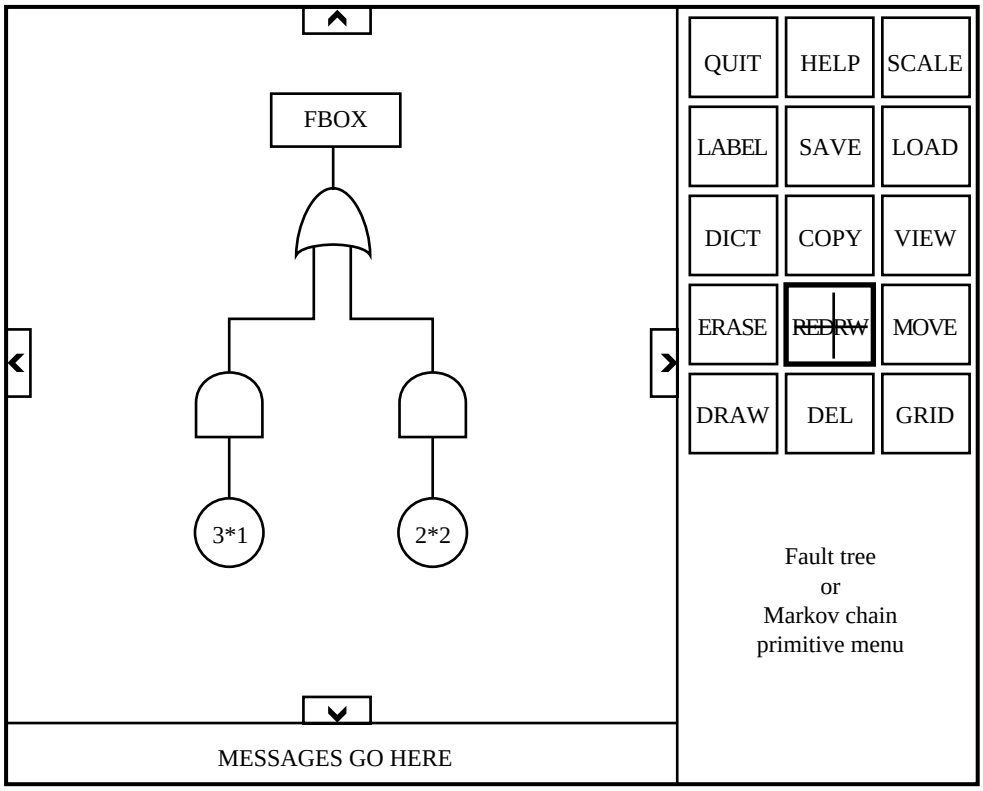


Figure 18. Selecting REDRW from the menu.

When VIEW is selected, a prompt appears that gives you the option of viewing the entire model or viewing dictionary information (component number, component name, and component failure rate name) of a selected basic event in a fault tree. The dictionary information option is

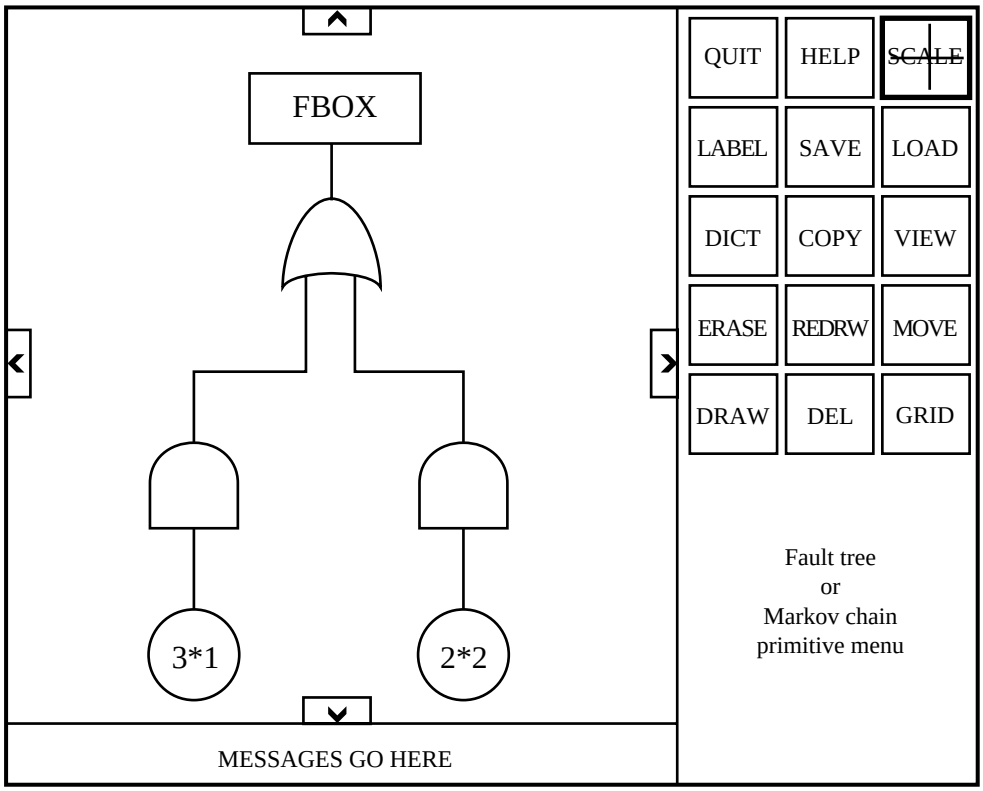


Figure 19. Selecting SCALE from the menu.

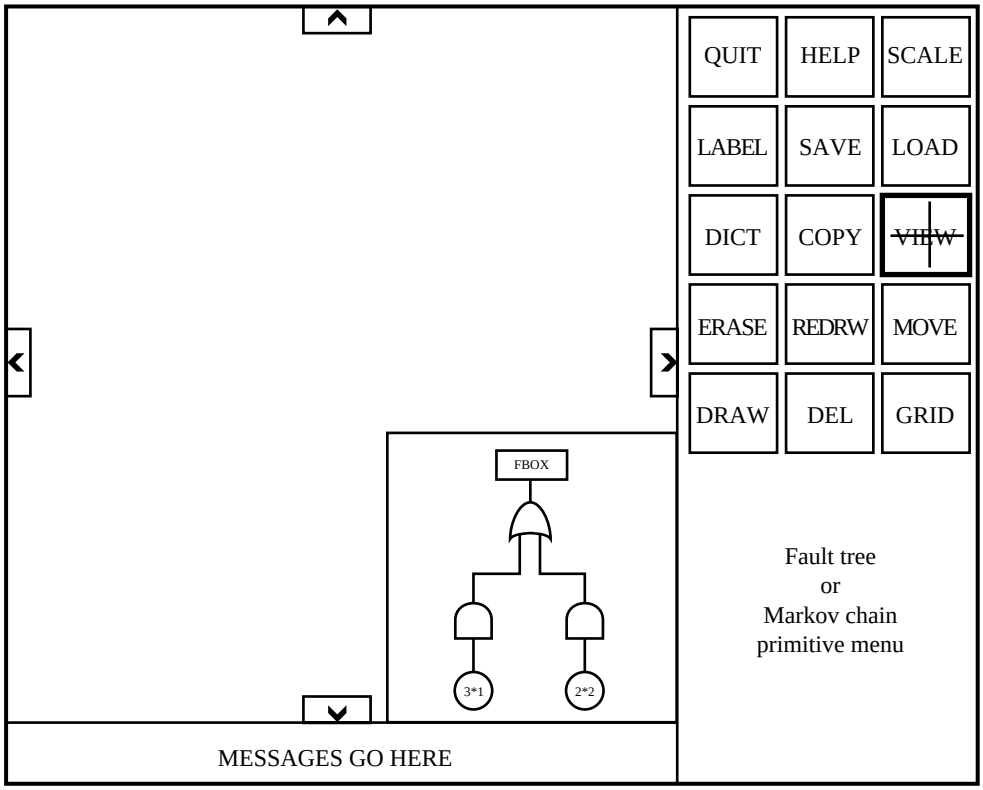


Figure 20. Selecting VIEW from the menu.

selected by entering d <CR> at the prompt. Next, select with the mouse the node of interest. The dictionary information appears in a highlighted box below the node. Click again to remove the box. To use this option, the model must have a dictionary file and must be labeled. To view the entire model, enter m <CR> at the prompt.

Chapter 5

Output Files

Once you have saved the output from your graphics session, the information is stored in a file with the .TRE extension (if the FORM is a fault tree) or .MKV (if the FORM is a Markov chain). In addition, for fault tree model, *.FTR (see chapter 5) and *.DIC files are created, and for a Markov chain model, a *.INT file is created. For the following Markov chain, the output

*.MKV file appears as follows:

```
N 100 200 16 3PROC

A 300 200 0 0 2 18 3*\LAMBDA\

N 300 200 16 2PROC

A 100 200 300 200 2 18 0

A 500 200 0 0 2 18 2*\LAMBDA\

N 500 200 16 F1

A 300 200 500 200 2 18 0
```

The lines beginning with an "N" represent nodes and those beginning with an "A" designate arcs, arrows, or lines (connectors). The fields for the nodes are as follows:

```
N xcoor ycoor type_node node_label
```

The (xcoor,ycoor) pair represents the location of the node on a 480-by-480 screen (PC version). Although this coordinate scale is not used by the graphics implementation (it uses a 1.0-by-1.0 representation), the scale is easy to decipher in the file. The node can be a circle if the FORM is a Markov chain. If the FORM is a fault tree, the node can be a circle, or gate, and gate, m/n gate, box, double circle, fbox, cold spare gate, functional dependency gate, sequence-enforcing gate, priority and gate, xor gate, or not gate. The node label can be a maximum of 13 characters in length. The fields for the connectors are as follows:

```
A x1 y1 x2 y2 direc type_connector next_node rate_paramter
```

A connector can either be an incoming or an outgoing connector. For an outgoing connector, $x2$ and $y2$ are both 0. In this case, $x1$, $y1$ represents the location of the node at the head of the connector. If the connector is an incoming one, $x1$, $y1$ represents the location of the node at the tail of the connector. In this case, $x2$, $y2$ is the same as the node coordinates. This condition flags it as an incoming connector. The integer direc specifies the direction of the connector. For arcs, a 1 signifies an upward arc, 2 signifies a downward arc, 4 signifies an arc that curves toward the right, and an 8 signifies curves toward the left. Arcs that do not connect two nodes on the horizontal or vertical have direction numbers of 5, 6, 9, and 10. Arrows have direction numbers

of 1 for a vertical line, 2 for a horizontal line, 4 for a positive slope line, and 8 for a negative slope line. Lines in fault trees have a default direction number of 2.

The type_connector can be a clockwise arc, arrow, or counterclockwise arc for a Markov chain. For a fault tree, the only connector types are *lines*. The interpretations for the types of node and types of connector numbers are as follows: circle, 16; or gate, 17; m/n gate, 18; and gate, 19; xor gate, 24; cold spare gate, 25; not gate, 26; functional dependency gate, 29; sequence-enforcing gate, 28; priority and gate, 27; box, 21; failure box, 22; double circle, 23; clockwise arc, 17; arrow, 18, counterclockwise arc, 19; line, 20. As shown in the *.MKV file, the nodes are all circles and the connectors are all arrows.

In addition, a dictionary file (*.DIC) can be created. It contains the name of each component in the system, its failure rate, and any coverage information. The program *tdrive*, which converts the fault tree to a Markov chain requires the total number of component types in the model and the symbolic failure rate parameter for each. For this reason, the *.DIC file must exist for a fault tree model.

When the FORM is a fault tree, you must run the programs *tdrive*, *fiface*, and *harpeng* (in that order). For Markov chain input, only the programs *fiface* and *harpeng* need be run. When the FORM is a Markov chain, a *.INT file is also output. This file is declared to be SORTED or UNSORTED and contains the list of transitions in the model and the rate parameter associated with each transition. SORTED is the declaration only if the states names (node names) are entered in an increasing integer order (meaning none are symbolic). If inputs are performed in this manner, you should append the *X to all transitions going to the failure states. The *fiface* program normally appends *X when it sees transitions entering a state beginning with an F. For a fault tree, a *.FTR file is created to be used by *tdrive*. The file is identical to the *.TRE file unless double circle graphics primitives are used. (See section 3.2.2 for details on this primitive.)

The programs of the HARP package—*tdrive*, *fiface*, and *harpeng*—limit the size of the model on the PC under the 16-bit DOS version but not under OS/2 or the 32-bit DOS version. A maximum of 500 states[1] without truncation (6000 with level three truncation) and 4000 transitions (without truncation) are permitted on the PC. If more states or transitions are created by the graphics program than can be accepted by the 16-bit version, you will need to upload the graphics files to your mini or mainframe computer and execute your model using an unscaled version of HARP (non PC version). The 32-bit DOS version and the OS/2 version do not have the state limitation, so the entire analysis can be performed on the PC.

NASA Langley Research Center
Hampton, VA 23681-0001
August 8, 1994

[1] Based on 512K of memory. More states are possible with more memory. (See vol. 1 of this TP.)

Appendix

Hardware and Software Configurations

This appendix is provided as a guide for the proper hardware and software configuration to execute the Graphics Oriented (GO) program. The Graphical Kernal System (GKS) was selected because it is the most portable graphics standard currently available. The GKS standard, however, is not universally conformed to as are other standards. Consequently, some differences in installation and operation are present. Substantial effort has been expended to minimize the possible confusion that may result. Each computing platform installation requires some knowledge of that system for proper installation.

Section A1 addresses IBM-compatible hardware and software requirements. Section A2 describes how to configure your PC to run the graphics program, section A3 describes how to configure your Sun workstation, and section A4 describes how to configure your VAX workstation. Section A5 discusses known bugs.

A1. PC Requirements

The GO graphics program was designed using the Graphic Software Systems Graphical Kernal System (ref. 4) package and has been ported to an IBM-compatible 286, 386, and 486 computing platforms.

An IBM PC AT compatible with the enhanced graphics adapter (EGA) display was used for development. However, the video graphics adapter (VGA) display mode is currently supported.

A1.1. Hardware Requirements

Minimum hardware needed to run the program for the PC is as follows:

- 512K of memory
- Two disk drives with 360K capacity
- Graphics display device (EGA or VGA)
- Mouse

To enhance the performance:

- Fixed disk
- 1.2 or 1.4 MB floppy drives
- Math coprocessor
- VGA

A1.2. Software Requirements

To invoke the Markov chain primitive menus, the m parameter must be passed to pcgo.exe. This action can be done with the batch file gom.bat. Passing the f parameter or no parameter (default case) invokes the fault tree primitive menus. The file go.bat is used for this purpose.

For program execution, you need DOS version 2.1 or later. All other files are provided. Program development (not required for execution) requires the following:

- Graphics Software Systems Graphical Kernal System Development package (GSS*GKS Version 2.02). This package includes GSS Computer Graphics Interface (GSS*CGI) and device drivers.
- Microsoft C Compiler version 5.1
- MS-DOS Linker version 3.2

Reference 5 is an excellent reference on the GKS system.

A2. Configuring the PC

Several files are needed to run (execute) the graphics program. These files are pcgo.exe, sr.z, sg.z, aries.m, espn.m, care.m, go.bat, and gom.bat. The pcgo.exe file is the actual executable program, the *.z files are the font files, and the *.m files are the FEHM graphics files. The *.bat files are required to invoke the GKS drivers and the GO executable. The file go.bat invokes the fault tree capability, and gom.bat invokes the Markov chain capability. Other support required files are given in the following listing.

To run the program correctly with GKS, you need the following lines added to your AUTOEXEC.BAT file. (No changes are required for your CONFIG.SYS file.)

```
SET CGIPATH = C:\CGI

C:\CGI\DRIVERS.EXE
```

You need to make a CGI subdirectory that contains all necessary GKS support code:

```
CGI.CFG             Contains driver spec., device names, & environ. vars.

KERNEL.SYS          Contains workstation configuration

DRIVERS.EXE         GKS driver

IBMEGA.SYS          Display driver

MSMOUSE.SYS         Mouse driver

GSSCGI.SYS          GSSCGI driver

IBMVGA12.SYS        VGA driver (optional in lieu of IBMEGA.SYS)

META.SYS            Meta driver for interfacing with I/O devices (optional)

FONT101.TBL         GKS font files

FONT102.TBL

     .

     .

     .

FONT106.TBL
```

To change devices or drivers, the user needs to edit the CGI.CFG file with a common ASCII editor. Upgrading from an EGA to a VGA display or adding a hardcopy output capability are such examples. KERNELOB.SYS can also be edited when workstation parameters change.

To help you configure your PC, the following example CONFIG.SYS and AUTOEXEC.BAT files are given.

```
CONFIG.SYS file contents and comments
files=20                        The max. number of files allowed open
buffers=15                      The max. number of buffers
break on                        Allows ^C to terminate process
shell=c:\command.com            OS is on drive c:\
REM Include your normal statements

AUTOEXEC.BAT
echo off
verify on
set comspec=c:\command.com
REM  Run GSS*GKS Device Driver Management Utility to initialize
REM  the system and to install the Transient Drivers
REM
SET CGIPATH = C:\CGI
C:\CGI\DRIVERS.EXE
REM  Include your normal path
path
cls
```

The following is a list of files included in PC GO. The *.z, *.m, go.bat, and gom.bat files are needed at run time with pcgo.exe. The file go.bat invokes the GKS drivers for fault tree entry, and gom.bat is used for Markov chain entry. The other files are used for development. Please read the read.me file.

```
adraw.c      arclab.c     carcs.c     cdisp.c      gssgo.c
gssinit.c    cmd_prim.c   coorarc.c   copygr.c     dict.c
drwprims.c   video.c      fhmfils.c   filein.c     fileout.c
formats.c    gendrw.c     glcent.c    greek.c      redraw.c
help.c       label.c      matrix.c    mouse.c      movepl.c
petri.c      snap.c       space.c     str.c        trans.c
upkeep.c     viewport.c   wrlab.c     aries.m      espn.m
stiff.m      sg.z         sr.z        defvar       dos.h
defs.h       devices.h    fonts.h     gdefines.h   memngt.h
menu.h       shapes.h     typedef.h   vars.h       vec.h
linkit.bat   lnfil        makefile    makeit.bat   go.bat
gom.bat      pcgo.exe     read.me     list
```

A3. Configuring the SUN Workstation

The HARP graphics program was designed to work with Sun GKS 4.1 under OpenWindows. To execute the HARP graphics program, perform the following steps:

- Type /usr/openwin/bin/openwin at the prompt to place yourself in the openwin environment
- The setenv GKSHOME path of the GKS library files and OPENWINHOME = usr/openwin must be in your .cshrc file. For example, setenv GKSHOME /usr/gks4.1
- Make sure your Sun workstation has the following libraries:

```
%cc -Bstatic -g -I$GKSHOME/include/gks -I$OPENWINHOME/include foo.c -o foo
-L$OPENWINHOME/lib -L$GKSHOME/lib -lgks -lxview -lolgx -lX11 -lm
(Note: ``-g'' is required only for dbxtool, the source code debugging tool.)
```

- The GKS libraries are supplied with the executable and source files.
- The following files must be found in a directory:

```
adraw.c        arclab.c    aries.m    carcs.c      cmd_prim.c
coorarc.c      copygr.c    defs.h     defvar       dev.h
dict.c         drwprims.c  espn.m     fhmfils.c    filein.c
fileout.c      formats.c   gendrw.c   glcent.c     gssgo.c
gssinit.c      help.c      label.c    matrix.c     memngt.h
menu.h         mouse.c     movepl.c   node_defs.h  petri.c
place_label.c  redraw.c    shapes.h   space.c      stiff.m
str.c          trans.c     upkeep.c   vars.h       viewport.c
wrlab.c        read.me     list
```

- In addition to the previous files, Makefile is needed to compile and link the HARP programs.
- To compile and link the HARP programs, type make.
- After typing make, an executable file named *go* should appear along with the object modules for the source codes.

A4. Configuring the VAX Workstation

The HARP graphics program was designed to work with DEC GKS version 4.0 with the C language binding and VMS 4.7 or higher. It has successfully been tested using the VAXstation II windowing software version 3.1 or higher and DEC windows version 1.0. If you are using DEC Windows, be sure to define logical names for the devices.

```
Example:  define gks$wstype 211
          define gks$conid wsa0
```

These lines can be included in your login.com file. To execute the HARP graphics program, perform the following steps:

- Set up foreign command $go := $full path name harp.exe in your login.com file. For example, $go := $vax1$dub0:[jane.harp]harp.exe

- The following files must be found in a directory:

adraw.c	arclab.c	aries.m	carcs.c	cmd_prim.c
coorarc.c	copygr.c	defs.h	defvar	dev.h
dict.c	drwprims.c	espn.m	fhmfils.c	filein.c
fileout.c	formats.c	gendrw.c	glcent.c	gssgo.c
gssinit.c	help.c	label.c	matrix.c	memngt.h
menu.h	mouse.c	movepl.c	node_defs.h	petri.c
place_label.c	redraw.c	shapes.h	space.c	stiff.m
str.c	trans.c	upkeep.c	vars.h	vec.h
viewport.c	where_gks.h	wrlab.c		

- In addition to the previous files, compile_harp.com and link_harp.com procedure files have been included. To execute these procedures, type @compile_harp followed by @link_harp.
- After executing these two procedure files, an executable file entitled harp.exe should appear. To execute harp type go.

During certain portions of the interactive session, a string input window will appear on the screen to request input. If this window should ever impair viewing important graphic information, click the left button of the mouse on the upper right corner of the string window to display a menu. This menu lets you shrink the window to an icon or push the window behind the graphics display. To continue, bring back the string window for response.

A5. Known Bugs and Suggested Improvements

Although all versions of the GO program were translations of the original PC version, the versions differ as a result of differences in their GKS implementations. Even though the developers made every attempt to maintain a common source code for all computing platforms, the different implementations precluded that aim (ref. 1). Consequently, each computing platform version of GO has its unique bugs. In delineating the known bugs, the GO version is identified, when possible. For the Sun workstation, the priority and gate accepts more than two inputs.

References

1. Bavuso, Salvatore J.; Koppen, Sandra V.; and Haley, Pamela J.: *Graphical Workstation Capability for Reliability Modeling.* NASA TM-4317, 1992.

2. Geist, Robert; Trivedi, Kishor; Dugan, Joanne Bechta; and Smotherman, Mark: Design of the Hybrid Automated Reliability Predictor. *Proceedings of the IEEE/AIAA 5th Digital Avionics Systems Conference*, 1983, pp. 16.5.1–16.5.8.

3. Bavuso, Salvatore J.; and Dugan, Joanne B.: HiRel—Reliability/Availability Integrated Workstation Tool. *Proceedings of the Annual Reliability and Maintainability Symposium*, IEEE, Jan. 1992, pp. 491–500.

4. McKay, Lucia: *GKS Primer.* Nova Graphics International Corp., 1984.

5. Sproull, Robert F.; Sutherland, W. R.; and Ullner, Michael K.: *Device-Independent Graphics.* McGraw-Hill Book Co., 1985.


3PROC
3*λ
2PROC
2*λ
F1

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704-0188

1. AGENCY USE ONLY *(Leave blank)*

2. REPORT DATE
November 1994

3. REPORT TYPE AND DATES COVERED
Technical Paper

4. TITLE AND SUBTITLE
HiRel: Hybrid Automated Reliability Predictor (HARP)
Integrated Reliability Tool System (Version 7.0)
HARP Graphics Oriented (GO) Input User's Guide

5. FUNDING NUMBERS
WU 505-66-21-02

6. AUTHOR(S)
Salvatore J. Bavuso, Elizabeth Rothmann, Nitin Mittal,
and Sandra Howell Koppen

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)
NASA Langley Research Center
Hampton, VA 23681-0001

8. PERFORMING ORGANIZATION REPORT NUMBER
L-16553C

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 TP-3452, Vol. 3

11. SUPPLEMENTARY NOTES
Bavuso: Langley Research Center, Hampton, VA; Rothmann and Mittal: Duke University, Durham, NC; Koppen: Lockheed Engineering & Sciences Company, Hampton, VA.

12a. DISTRIBUTION/AVAILABILITY STATEMENT
Unclassified–Unlimited

Subject Category 61

12b. DISTRIBUTION CODE

13. ABSTRACT *(Maximum 200 words)*
The Hybrid Automated Reliability Predictor (HARP) integrated Reliability (HiRel) tool system for reliability/availability prediction offers a toolbox of integrated reliability/availability programs that can be used to customize the user's application in a workstation or nonworkstation environment. HiRel consists of interactive graphical input/output programs and four reliability/availability modeling engines that provide analytical and simulative solutions to a wide host of highly reliable fault-tolerant system architectures and is also applicable to electronic systems in general. The tool system was designed at the outset to be compatible with most computing platforms and operating systems, and some programs have been beta tested within the aerospace community for over 8 years. This document is a user's guide for the HiRel graphical preprocessor Graphics Oriented (GO) program. GO is a graphical user interface for the HARP engine that enables the drawing of reliability/availability models on a monitor. A mouse is used to select fault tree gates or Markov graphical symbols from a menu for drawing.

14. SUBJECT TERMS
Reliability; Availability; Fault tree; Markov chain; Coverage; Faults; Errors; Fault tolerant; Graphical user interface (GUI)

15. NUMBER OF PAGES
37

16. PRICE CODE
A03

17. SECURITY CLASSIFICATION OF REPORT
Unclassified

18. SECURITY CLASSIFICATION OF THIS PAGE
Unclassified

19. SECURITY CLASSIFICATION OF ABSTRACT
Unclassified

20. LIMITATION OF ABSTRACT