Chitra93:
A System to Model Ensembles of Trace Data

(Last updated for Version 4.122.)

Marc Abrams
Department of Computer Science
Virginia Polytechnic Institute and State University
Blacksburg, VA 24061-0106
703-231-8457
chitra@cs.vt.edu

1 August 1994

Contents

List of Figures

List of Tables

Acknowledgements

CHITRA was conceived and supervised by Marc Abrams. Naganand Doraswamy implemented CHITRA91. Krishna Ganugapati implemented CHITRA92 with help from Tim Lee. Tim Lee, Horacio Cadiz, David Connerley, Anup Mathur, Alan Batongbacal, David Holcomb and Mike Tiffany enhanced CHITRA92 to produce CHITRA93. The development of CHITRA is supported in part by Research Initiation Award NCR-9211342 from the National Science Foundation, the National Science Foundation's SUCCEED[1] Coalition (Cooperative Agreement No. EID-9109853), and a summer 1993 grant from the U.S. Navy NSWC administered by the VPI&SU Systems Research Center.

CHITRA93 uses a histogram widget from the Free Widget Foundation, coordinated by Brian Totty, Department of Computer Science, University of Illinois, Urbana-Champaign. CHITRA93 uses code to calculate Chi square values from the Collected Algorithms of the ACM and written by Gary Pearlman of Ohio State (routines pochisq and poz). Finally, certain CHITRA93 projections uses a scanner writeen by Dr. Teodor Rus, University of Iowa, Computer Science Department.

[1]SUCCEED is a soalition of eight schools and colleges working to enhance engineering education for the twenty-first century.

Preface

CHITRA, a sanskrit word meaning a beautiful or pleasing picture, represents a new approach to understanding trace data generated by computer and communication systems. CHITRA can analyze any time series data, although its features are specifically tailored to trace data from computer and communication systems that we have studied over the years. Because a single trace file represents just a single observation of a system, and a single observation may or may not be representative of the range of behavior that the system can manifest, CHITRA93 analyzes multiple trace files representing multiple observations. A collection of trace files is called an *ensemble*. CHITRA couples methods to visualize and statistically analyze ensembles, culminating in the construction of one or more models that fit the data in an ensemble. CHITRA addresses the potential state space explosion problem in a model by providing a set of *transforms* that aggregate or delete information from a trace file.

This document is written under the assumption that the reader is familiar with the concepts and examples presented in two papers: [1], which describes the first generation of CHITRA, called CHITRA91, and [3], which describes the second generation, called CHITRA93. There is a third generation of CHITRA under development, called CHITRA94, which is hardware, operating system, and graphical user interface independent; user extensible; and includes new visualizations as discussed in [4].

Papers [1, 3] explain the following concepts. CHITRA93 is concerned with analyzing, not collecting traces. CHITRA93 takes as input an ensemble of *Timed System State Sequences* (TSSS's). Each TSSS contains a sequence of timestamped states representing the time evolution of the system that was observed. Each TSSS contains the data from one trace file representing one observation of a system under study. A TSSS represents the time evolution of a system. CHITRA93 treats time as continuous and system states as discrete. Examples of TSSS states are the control point of each process or thread in a parallel program; the virtual addresses referenced by a program; or the origin, destination, and length of the last packet injected into a communication network. To model an ensemble, a user must determine if the TSSS's, or *members*, comprising an ensemble represent the "same" system behavior. Therefore CHITRA93 provides methods for testing whether an ensemble is *homogeneous*. Using these tests, a user partitions an ensemble into mutually exclusive, exhaustive, and homogeneous subsets. Chitra93 will then create one model for each subset. Chitra93 contains two types of models. The first is a homogeneous, semi-Markov chain empirical model for each TSSS. The model is reported as a state transition matrix (given the current state, the matrix gives the probability distribution of the next state), and statistics on the occupancy time for each state. The second model, called *CHAID-based*, relaxes the Markov assumption that the next state depends only on the present state. CHAID (Chi-squared automatic interaction detection) is a method of partitioning multivariate data sets using continency tables [11]. The CHAID-based model identifies, for each state S of interest, which past or future state best predicts an occurrence of S.

A brief history introduction to the features of Chitra is given below.

Chitra91

Chitra91 is a prototype system and is described in [1, 2, 9]. The input to Chitra91 is a single textual trace file that contains a single TSSS and the result of analysis is a homogeneous semi-Markov model that fits the TSSS. The trace, once opened, can then be displayed by Chitra91 for visual inspection and visual editing by the user. Visual editing reduces the size of the semi-Markov model state space and helps the user build an accurate model of the TSSS. Chitra91 supports four *transforms* for visual editing of the TSSS, which are formally defined in [14]. *Clipping* is useful in eliminating the initial and final transient portions of the TSSS. *Aggregation* creates a hierarchical model to reduce the semi-Markov model state space size. There are two varieties of aggregation: *aggregation by pattern* and *aggregation by filter*. Aggregation by pattern is useful in capturing deterministic patterns in a TSSS. Chitra91 combines the states that form a deterministic pattern into one aggregate state. A semi-Markov model thusly formed describes the behavior of the program more accurately. *Aggregation by filter* is useful in combining states that occur infrequently into one state, thereby reducing the state space size. Finally, *projection* is a transform that uses the semantics of states and is specified textually. It is also reduces the state space size.

Chitra91 views a TSSS as a *time series*, or *signal*, as used in electrical engineering. Operations such as filtering which are often applied to signals are supported by Chitra91. Chitra91 provides three *views* of the TSSS represented as a signal: event domain, time domain, and frequency domain. These views permit the human user to look at the TSSS as time-series data.

Chitra92

Unlike CHITRA91, which allowed only one TSSS to be viewed at a time, CHITRA92 [10] allows simultaneous views of the same or different TSSSs. The views are automatically kept consistent. Transforms can be applied from multiple views. The architecture of CHITRA92 facilitates the extension of Chitra to include alternate modeling techniques, transforms, and views. To make the system portable among Unix systems, implementation was done using the widely available OSF/Motif toolkit, the MIT Athena toolkit, and the X11 window system. The format of the input TSSS files to Chitra was designed to be machine independent; for example a user can collect TSSSs on a 32 bit architecture but run Chitra on a 64 bit architecture.

Chitra93

CHITRA93 adds to CHITRA92 two main features: the ability to analyze and model ensembles, rather than single TSSS's, and the CHAID-based model.

Chitra94

CHITRA94 [4] is a complete source code rewrite of CHITRA. It is hardware, operating system, and graphical user interface independent. It consists of a collection of small programs that communicate through a standard file format, in contrast to the previous generations of CHITRA, which are single monolithic programs. This permits simple and quick user extension of the tool, and provides an application programming interface so that one can build sophisticated analysis and modeling programs using CHITRA94's modules as building blocks. CHITRA94 distinguishes numerical from categorical time series data, and is unique in its methods to visualize categorical time series. The visualization methods include a novel periodogram for categorical data, and a *mass evolution graph*, which shows the probability mass distribution evolution in time of one or more TSSS's.

Overview of this Manual

The chapters of this manual are written as a sequence of lessons. Chapter 1 starts with the simplest task that one would wish to do with CHITRA93: visualize and construct a semi-Markov model of a single trace file. Chapter 2 is a short lesson on reducing the time required to open and the disk space required to store trace files. Chapter 3 goes a step further by explaining how to visualize and model not one but a set, or ensemble, of trace files. Chapter 4 then explains how to reduce the duration or state space size of the model constructed by applying transforms to simplify an ensemble of trace files. Chapter 5 explains the CHAID-based model. Appendix A explains how to install CHITRA93. A suggested procedure for building ensemble models is given in Chapter 6. Appendix B gives the input trace file format used by CHITRA93. Some frequently asked questions are answered in Appendix C. A summary of the example trace files included with CHITRA93 is provided in Appendix D. Appendix E provides some important information to anyone that wants to modify the CHITRA93 source code, and indicates how we verified CHITRA93. Finally unimplemented items and known bugs are listed in Appendix F.

This manual is written in a tutorial style. Therefore the reader should use CHITRA93 from a workstation of X-terminal with a color monitor while following the tutorials given in each of the lessons in the manual.

Conventions Used in this Manual

This manual assumes you are using the Unix "csh" shell. The commands and file editing described in certain places, particularly §1.1 and Appendix A, must be modified if you use a different command shell.

The courier font is used to denote characters that you are asked to type on a keyboard (e.g., "Enter `chitra93_mips`".).

The colon and rightarrow ("→") are used to form the names of commands selected from cascaded menus. For example, "Root window: File → Open → .csl" denotes menu item ".csl" in the cascaded menu seen when "Open" is selected under the menu heading "File" in the CHITRA93's root window.

A "CHITRA93 session" refers to the time interval from when you start CHITRA93 until you exit from CHITRA93.

Disclaimer

Please read file $CHITRA/src/COPYRIGHT:[2] after installing the software according to Appendix A.

> This program is provided "as is" without warranty of any kind, either expressed or implied, including, but not limited to, the implied warranties of merchantability and fitness for a particular purpose. The entire risk as to the quality and performance of the program rests with the user who, should the program prove defective, also assumes the cost of all necessary servicing, repair and/or correction. In no event unless required by applicable law or agreed to in writing will the CS Department of VPI&SU, the Free Widget Foundation or its participants, or Gary Pearlman, or any other copyright holders

[2] "$CHITRA" is an environment variable defined in §1.1

be liable to the user for damages, including any general, special, incidental or consequential damages arising out of the use or inability to use the program (including but not limited to loss of data or data being rendered inaccurate, or losses sustained by the user or by third parties, or a failure of the program to operate with any other programs), even if such Department or other copyright holder has been advised of the possibility of such damages.

Chapter 1

Lesson 1: Visualizing One Trace and Creating a Semi-Markov Model

The simplest task which one can do with CHITRA93 is to open a single trace file, visualize it, and generate a model fitting the trace. This lesson guides you through these steps using a trace from a parallel program as an example. Subsequent chapters augment these steps to analyze ensembles of traces.

1.1 Before You Run Chitra93 the First Time

If CHITRA93 has not been installed on a Unix workstation on which you have an account, follow the instructions in Appendix A to obtain and install a copy of CHITRA93 via anonymous ftp.

Before you run CHITRA93 for the very first time, you must perform the steps listed below. Let C denote the hostname of the computer on which CHITRA93 is installed and on which you will execute CHITRA93 (e.g., the host running the X-client). Let S denote the hostname of the X-terminal or host whose keyboard you are typing on (e.g., the host running the X-server). If you are logged into the console of the host that executes CHITRA93, then $C = S$.

1. If $C = S$, then open a terminal window (e.g., xterm) and skip the rest of this step.

   Because CHITRA93 uses the Motif graphical user interface, you should do the things you normally do before starting a Motif application:

   (a) Type `/usr/bin/X11/xhost` C.

   (b) Open a terminal window (e.g., xterm). Log into C. Execute "`setenv DISPLAY` S`:0.0`".

2. In the terminal window you opened in step 1, edit file `~/.cshrc` and:

   (a) Add the line "`setenv CHITRA` ⟨`chitradir`⟩".

   (b) Edit the line(s) defining the `PATH` variable to add `$(CHITRA)`.

3. Type `source ~/.cshrc`.

4. Type `rehash`.

5. This setp is optional. Append the content of `$CHITRA/CHITRA-XDEFAULTS` to file `~/.Xdefaults` file.

6. Start CHITRA93 by typing `chitra93_`⟨platform⟩, where ⟨platform⟩ is the type of workstation executing CHITRA93: `alpha` (for DECstation 3000 running OSF/1 1.3 or 2.0), `mips` (for DECstation 5000 running Ultrix 4.3), `rs6k` (for IBM RS/6000), `paragon` (for Intel Paragon), `ksr` (for KSR-1), `next` (for NeXTstation running NEXTSTEP 3.x with Co-Xist), `sgi` (for Silicon Graphics Indigo), or `sparc` (for a Spartcstation running Solaris 2.3). If you see the console message "X Toolkit Error: Can't Open display", then repeat Step 1.

   After a short delay, CHITRA93 should draw a window on the screen. If you see CHITRA93's *root window*, illustrated in Fig. 1.1, then skip Step 7.

7. If you do not see the root window in Fig. 1.1, then you may need to change the list of fonts that CHITRA93 uses. The fonts used by CHITRA93 are specified in a file named `.chitra_fonts`. (Refer to Fig. A.1 in §A.2 for an illustration of a `.chitra_fonts` file.) If the `.chitra_fonts` file is not found, then typing `chitra93_`⟨platform⟩ results in Fig. 1.2. If this happens, first verify that the $CHITRA path variable is properly set (see Step 2 above). If the $CHITRA path variable is properly set, then type `ls $CHITRA/.chitra_fonts` to determine if the `.chitra_fonts` exists. If the `ls` command output does not list `.chitra_fonts`, then reinstall CHITRA93 according to the instructions in Chapter A to recreate the file.

   If you see neither the root window (Fig. 1.1) nor the warning box in Fig. 1.2, then you should see Fig. 1.3. In this case the X-terminal or workstation which you are using does not have a font that is listed in file $CHITRA/`.chitra_fonts`. Type

   ```
   cp $CHITRA/.chitra_fonts ~
   /usr/X11/bin/xlsfonts -display S:0
   ```

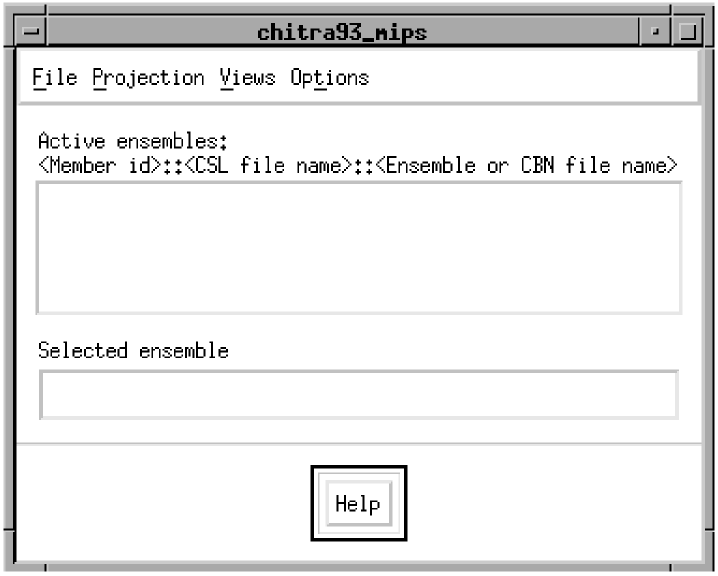


Figure 1.1: CHITRA93 root window.

The first command makes your own personal copy of file `.chitra_fonts`. Repeat the `xlsfonts` command on *every* X-terminal or host that you ever expect to use as S. Then edit `~/.chitra_fonts` to replace the font name(s) listed in the warning box that appears on your monitor (Fig. 1.3) with a name listed in the output of *all* `xlsfonts` commands that you execute.

Figure 1.2: One possible error message indicating that CHITRA93 could not properly start execution.

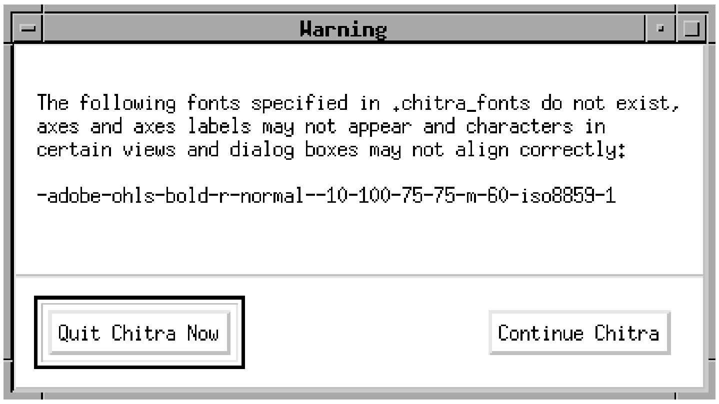


Figure 1.3: Another possible error message indicating that CHITRA93 could not properly start execution.

Advanced hints

Personalizing fonts: Each time Chitra executes, it will search for the `.chitra_fonts` file in the order listed below:

Current directory

`$CHITRA` directory

home directory (e.g., ~)

Therefore you can choose you own personal set of fonts that are different from those in $CHITRA/`.chitra_fonts` by putting a modified `.chitra_fonts` in your current directory.

`~/.Xdefaults`: Each non-comment line in the `~/.Xdefaults` file lists a Chitra widget resource name, such as

```
chitra93*parsefile_dialog*textColumns
```

and a value to use (e.g., "80"). An example is given in Figure 1.4. The first string in the resource name (e.g., "chitra93") is the name of the command you use to execute CHITRA93. It may contain an underscore, but may not contain a ".". Thus if you invoke CHITRA93 by the command `chitra93v4.122_sparc` then your `~/.Xdefaults` will be ignored (because "4.122" contains a ".").

Use the command `fgrep xTmanage $CHITRA/src/*.[h,c]` for a start on learning resource names. Note that one can only set resources in the .Xdefaults file whose values are not hardwired in the code.

1.2 Window operations

Skip this section if you are familiar with common Motif window operations.

In Motif, the *focus* refers to which window on the screen receives keystrokes and responds to mouse clicks, and this window is called the *key window*. You can usually tell which window on the screen is the key window because this window's border is highlighted in a color different from all other windows on the screen. To change focus to a window, simply click the mouse pointer in the banner at the top of the window (e.g., in Fig. 1.1, the banner contains "chitra93_mips"), and the border of this window should now be highlighted.

Try the following operations on the root window on your screen.

- You can resize any CHITRA93 window as follows: Position the mouse pointer on the window border so that the pointer shape changes shape (probably to an arrow pointing to a line). Hold down the left mouse button and moving the mouse. Release the mouse button. The window (probably

```
/* Background color */
chitra93*background: aquamarine
chitra93_alpha*background: aquamarine
chitra93_mips*background: aquamarine
chitra93_sparc*background: aquamarine

/* Width of FileSelection dialog box from ``Open .csl'' */
chitra93*parsefile_dialog*textColumns: 80
chitra93_alpha*parsefile_dialog*textColumns: 80
chitra93_mips*parsefile_dialog*textColumns: 80
chitra93_sparc*parsefile_dialog*textColumns: 80

/* Width of FileSelection dialog box to save .cbn
   after ``Open .csl'' */
chitra93*bin_dialog*textColumns: 80
chitra93_alpha*bin_dialog*textColumns: 80
chitra93_mips*bin_dialog*textColumns: 80
chitra93_sparc*bin_dialog*textColumns: 80

/* Width of FileSelection dialog box from ``Open .cbn'' */
chitra93*file_dialog*textColumns: 80
chitra93_alpha*file_dialog*textColumns: 80
chitra93_mips*file_dialog*textColumns: 80
chitra93_sparc*file_dialog*textColumns: 80
```

Figure 1.4: Sample CHITRA-XDEFAULTS file, added to your `~/.Xdefaults`.

to an arrow pointing to content will be redrawn to fit the (probably to an arrow pointing to new window dimensions.

- The upper right corner of the window may have a small square box, at the right end of the window banner. If you click the mouse on the box, the window will enlarge to cover the entire screen. This is useful to see fine detail in a window, such as a graph in a visualization. Clicking the box again restores the original window size.

- To the left of the small square box at the top of many windows may be a small circle. Clicking on the circle will replace the window by an icon. Clicking twice on the icon restores the window. This is useful when one has many windows open, to avoid cluttering the screen. If there is no box, use the "minimize" and "maximize" menu items in the menu displayed by clicking on the horizontal line in the upper left window corner.

The Sun Solaris version of CHITRA93 does not permit iconification of individual windows. Thus clicking on the circle will iconify *all* CHITRA93 windows.

1.3 How to Create an Input File for Chitra93

Start CHITRA93 by following the instructions in steps 1 and 6 of §1.1.

To use CHITRA93, one needs a trace file. The question of how one collects a trace is not addressed by CHITRA93; hence we start with the assumption that the reader has already collected traces and wishes to analyze the traces.

We begin by giving a formal statement of the definition of "trace file" used by CHITRA93. The behavior of a system can be described by enumerating the sequence of *states* it follows during its operation. A *system state* is an n-tuple, or vector of length n, enumerating the values of n system attributes at some point during its execution. An attribute might be the section of source code that each process in a parallel program is currently executing, the value of one or more program variables, the memory address last referenced, the size of the last packet injected into a communication network. For example, the system state in a communication network might be the 2-tuple *(host-name, length of packet last injected)*. The choice of state is governed by the objective of a performance study, as discussed in Chapter 6.

A system is described by the sequence of system states that it traverses during operation, termed a *system state sequence* (SSS). Let L denote the number of states in an SSS; $S_0, S_1, \ldots, S_{L-1}$ denote an SSS; and E_i denote the entry time of state S_i $(0 \leq i < L)$. A *timed SSS* (TSSS) is obtained by augmenting an SSS with the time at which each state is entered. The corresponding TSSS is a sequence of ordered pairs (E_0, S_0), (E_1, S_1), $\ldots$, (E_{L-1}, S_{L-1}), where $E_0, E_1, \ldots, E_{L-1}$ is a strictly increasing sequence. Each pair of successive states (S_k, S_{k+1}) (for $0 \leq k < L-1$) is a *transition*. Transition number k is *out* of state S_k and *into* S_{k+1}. An *event* causes a transition; the *event number* of S_i is i. A *trace file* is defined to be a TSSS, and an *ensemble* is a set of TSSS's. Henceforth we use the term "TSSS" rather than a trace file because a TSSS is initially the same as a trace file, but will differ after a transform (Chapter 4) is applied. An ensemble *member* is one SSS in an ensemble; each ensemble member is identified by a unique, small non-negative integer.

Example: The example used in this chapter is a simplified form of Example 1 from sections I. and V.B of [1]. The object of this chapter is to show the reader how to generate a semi-Markov model similar to Fig. 12 in [1]. In the example, the "system" is a shared memory parallel computer executing a multiple process program simulating the "dining philosophers," due to Dijkstra [8]. The simulation models two philosophers, numbered 0 and 1. Each philosopher alternately thinks and eats. A philosopher requires two utensils (e.g., a fork and a spoon, numbered 1 and 2) to eat. Being poor, both philosophers share the same utensils, so that one one philosopher at a time can eat. After a philosopher acquires (picks up) the utensils, he or she eats for a constant period of time, of duration x, and then releases (lays down) utensil 1 followed by utensil 2, then thinks for a constant period of time x, and then waits until utensil 1 is free, then acquires (picks up) utensil 1, then waits until utensil 2 is free, then acquires (picks up) utensil 2, and then eats for time x, and so on. The cycle is repeated R times. The dining philosophers problem represents the contention of processes (e.g., philosophers) in a computer system for resources (e.g., utensils). The trace data we will analyze is obtained from measurement of the simulation program on a Sequence Symmetry shared memory multiprocessor as described in [1].

The TSSS files are in directory $CHITRA/examples/Diners. The naming convention for files is <NumPhils>_a_x_$< R >$_<ObservationNum>.csl, where <NumPhils> is the number of philosophers (2 in this manual), and x is the duration of eating and thinking, as explained in [1]. There are at least three observations of each case traced, so that we will analyze ensembles of three TSSS's in Chapter 3. The <ObservationNum> has value 1, 2, or 3 to distinguish the three observations. Thus the three files 2_a_10_200_[1,2,3].csl each trace two philosophers that think and eat for $x = 10$ time units, in which each philosopher repeats the eat/think cycle 200 times, and represents observations one, two, and three made of the program.

The objective of our performance study is to create a model of the behavior of the two philosophers. (The model is used in [1] to see if both philosophers share the utensils fairly, if each philosopher waits to acquire utensils for a small time (compared to x), and to see if the system behavior is in a statistical sense *stationary.*)

The system state will be a 2-tuple, in which each component denotes which action (acquiring utensil 1 (denoted by A1), acquiring utensil 2 (A2), eating (Eat), releasing utensil 1 (R1), releasing utensil 2 (R2), thinking (T)) philosophers number 0 and 1 are currently performing. For example, in system state (Eat,A1) philosopher 0 is eating and philosopher 1 is acquiring the first utensil. We will also use U to denote a philosopher whose state is unknown (this is always the initial state of each philosopher in any observation period), and F to denote a philosopher that has finished.

Recall that a TSSS file consists of L 2-tuples, denoted (E_0, S_0), (E_1, S_1), $\ldots$, (E_{L-1}, S_{L-1}). Please type:

```
more $CHITRA/examples/Diners/2_a_10_200_1.csl
```

File $CHITRA/examples/Diners/2_a_10_200_1.csl contains a TSSS in the format that CHITRA93 reads as input, called a *Chitra Specification Language File* (CSL). Such a file's name ends is the ".csl" suffix. A portion of the file is shown in Fig. 1.5. The portion that follows the word "Trace:" in Fig. 1.5 is the actual TSSS. The remaining portion of the trace specifies various attributes about the TSSS.

```
Program Dining;
begin
  Setup:
    Maxcomp = 2
    comp 0-1 =  'Philosopher', 1 byte,8;
        0 = T;
        1 = A1;
        2 = A2;
        3 = Eat;
        4 = R1;
        5 = R2;
        6 = U;
        7 = F
  Trace:
    2347098440 < 6, 6 >
    2347099297 < 0, 6 >
    2347099327 < 1, 6 >
    2347099347 < 2, 6 >
    2347099366 < 3, 6 >
    2347099396 < 4, 6 >
    2347099410 < 5, 6 >
    2347099422 < 0, 6 >
    2347099447 < 1, 6 >
    2347099456 < 2, 6 >
    2347099466 < 3, 6 >
    2347099490 < 4, 6 >
    2347099499 < 5, 6 >
    ...
    2347118043 < 0, 3 >
    2347118068 < 1, 3 >
    2347118068 < 1, 4 >
    ...
    2347123599 < 7, 5 >
    2347123609 < 7, 7 >
end.
```

Figure 1.5: The content of file $CHITRA/examples/Diners/2_a_10_200_1.csl.

The first TSSS record, 2347098440 <6,6>, indicates that first observation of the system occurred at time 2347098440, when the system entered the state represented by 2-tuple (6,6), in which both philosophers were in state 6, also known as state "U" (for "undefined state"). The correspondence of 6 to "U" is given in the preamble by the statement "6=U". The time unit, which happens to be microseconds for this data, is not specified in a .csl file, and CHITRA93 has no concept of time unit. It is up to the user to choose and remember the time unit, and to insure that all input .csl files that are analyzed together have the same time unit (in this case, microseconds). The second TSSS record, 2347099297 <0,6>, indicates that at time 2347099297 the new system state is (0,6), which means that philosopher 0 changed state from 6 (e.g., U) to 0 (e.g., T), and philosopher 1 remained in state 6 (e.g., U). The remaining TSSS records have a similar interpretation. The final record shows that at time 2347123609 the system entered state (7,7), or (F,F), where "F" denotes "finished".

The remaining information in the preamble gives the value of n in the n-tuple representing system state (e.g., "Maxcomp=2"), the name of each component (e.g., "comp 0-1='Philosopher' "), the amount of space in bytes required to store each tuple component (e.g, 1 byte, which means that the value of a tuple component is limited to 0 to 255, although only 0 to 7 is used here), the number of tuple components used (e.g, "8", which is the values 0-7), and a symbolic name of up to three characters for each numeric component value (e.g., "0=T" to "7=F").

To summarize, the first step in using CHITRA93 is to create one .csl file that is similar in appearance to Fig. 1.5 for each TSSS, that you wish to analyze. Additional examples of .csl files are contained in the subdirectories in the $CHITRA/examples directory (see Appendix D). The formal grammar for a .csl file is given in Appendix B.

Miscellaneous .csl file facts

- The maximum value of n in the "Maxcomp=n" statement of a .csl file is 64 in CHITRA93 version 4.122.
- The number of tuple components used (e.g, ",8" in Fig. 1.5, and a symbolic name for each numeric component value (e.g., "0=T" to "7=F" in Fig. 1.5) are *optional*. This is useful in TSSS's containing hundreds or thousands of state vector component values, where one does not want to give every value a name.
- The maximum number of values of a state vector component is equal to the maximum value of type "unsigned int" on the machine running CHITRA93. This is usually $2^{32}-1$ on a 32-bit architecture or $2^{64}-1$ on a 64-bit architecture.

1.4 How to Open One Trace File

Now that you have a .csl file containing a TSSS, it is time to use CHITRA93 to visualize its contents. At this point you should see the root window (Fig. 1.1) on your monitor screen. First move the cursor over the word "File" in the root window and press and hold the left mouse button. A pulldown menu will

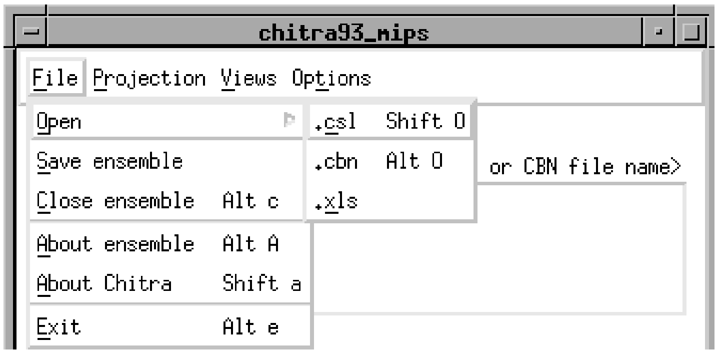


Figure 1.6: Illustration of choosing "Root window: File → Open → .csl".

appear. Position the cursor over the "Open" choice, and a second level, or cascaded, menu will appear (Fig. 1.6). Position the cursor over the ".csl" menu item in the cascaded menu and release the left mouse button. You have now invoked the command to open a .csl file.

The next box that appears on the screen is called the Motif *file selection dialog box*. Use the box to locate and click on the file name $CHITRA/examples-/Diners/2_a_10_200_1.csl, as illustrated in Fig. 1.7. As a result, the file name should be highlighted. ("Highlighted" means printed in white letters on a colored background, compared to colored letters on a white background before you clicked on it.) The top box, labeled "Filter," is set to display only file names ending in ".csl*". Normally you do not need to alter this. Below this are two side-by-side boxes labeled "Directories" and "Files". The "Directories" box initially lists the working directory when you invoked Chitra93, which is probably $CHITRA. To open a directory, such as Examples, click on the directory name. The "Files" box lists all files satisfying the pattern in the "Filter" box in the directory highlighted in the "Directory" box. After you find "$CHITRA/examples/Diners" in the "Directory" box, locate and click on "2_a_10_200_1.csl" in the "File" box. Then click on the "OK" button, and the dialog box will vanish.

Chitra93 must now convert the .csl file to a machine readable form, called a *Chitra Binary Format* (CBN) file. Chitra93 will either store the CBN file in /tmp (the temporary file system on a Unix computer), or in a directory chosen by the user. It allows selection of a directory name through a second dialog box (Fig. 1.8). For now, click on "Cancel", which will store the CBN file in /tmp. The dialog box will vanish.

Chitra93 now converts the .csl file to a CBN file. The time required is proportional to the .csl file size. If the .csl file contains any syntax errors, or if any other problems were encountered during the conversion, a box similar to

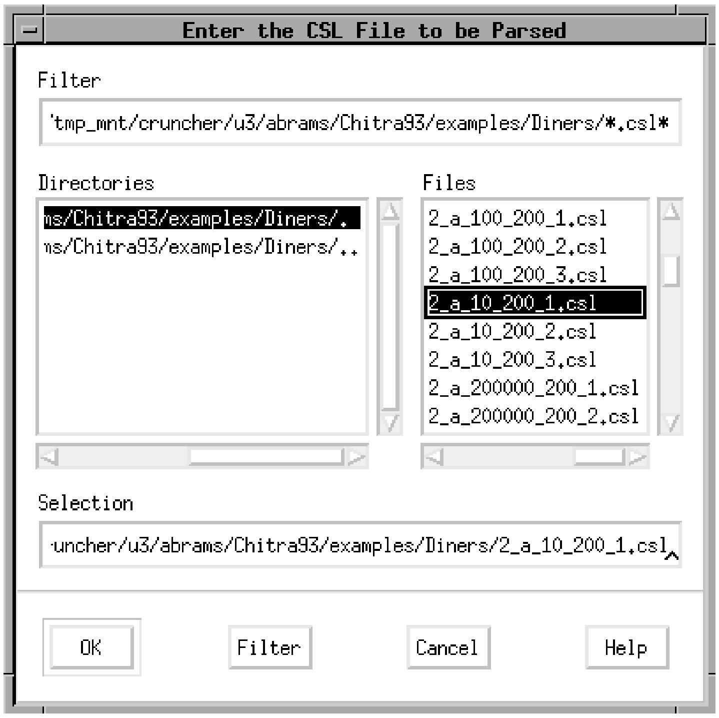


Figure 1.7: File selection box that appears upon selecting the ".csl" menu item under "Open" in root window.

Fig. 1.9 will appear. Fig. 1.9 indicates that file 2_a_10_200_1.csl contains two states with the same timestamp. Thus the system made one or more transitions in zero time. (Often the reason that a TSSS contains multiple states with the same timestamp is that the granularity of the clock used when the TSSS was recorded is not sufficiently small to guarantee that only one event occurs at each clock tick.) If you examine file 2_a_10_200_1.csl (e.g., by using `more 2_a_10_200_1.csl`), you will see that it contains:

```
2347118043 < 0, 3 >
2347118068 < 1, 3 >
2347118068 < 1, 4 >
```

The system enters state (T,Eat) (i.e., $<0,3>$) at time 2347118043, then philosopher 0 stops eating and starts acquiring the first utensil at time 2347118068 (i.e., $<1,3>$), and then (at the same time, 2347118068) philosopher 1 begins releasing utensil 1 (i.e., $<1,4>$). The message in Fig. 1.9 indicates that the sequence of three states above will be treated as though it were:

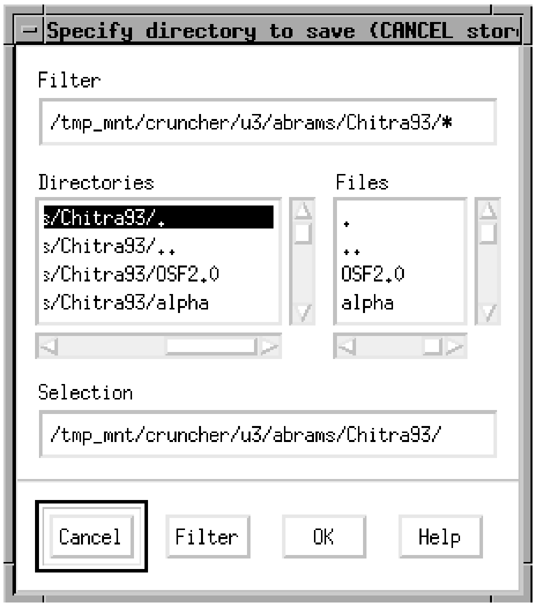


Figure 1.8: File selection box that appears upon selecting "OK" in Fig. 1.7.

```
2347118043 < 0, 3 >
2347118068 < 1, 4 >
```

This action merely ignores an intermediate state (i.e., <1,3>). Click "OK" on the warning box (Fig. 1.9).

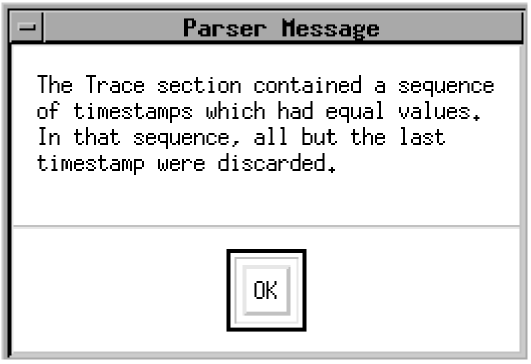


Figure 1.9: Parser message box that sometimes appears after selecting "Cancel" or "OK" in Fig. 1.8.

Advanced hints

- To display *all* files in a directory in Fig. 1.7, rather than just those ending in ".csl", highlight the "*.csl*" text in the "Filter" box and hit backspace and then click on the "Filter" button to display.
- If the content of the directory highlighted in the "Directories" box changes while the file dialog box is displayed, click on "Filter" to update the list of files in the "File" box.
- Table 1.1 describes all menu items in the root window "File" menu.
- In many dialog boxes and windows with buttons, one button is highlighted by being outlined in a square. One example is the "OK" button in Fig. 1.7 on page 26. Hitting the return key selects this button, saving you the trouble of moving the mouse pointer over the button.

1.4.1 What Happens When a TSSS is Opened

Chitra93 performs certain actions when it loads a TSSS in preparation for subsequent visualization and modeling.

1.4.1.1 System to Model State Mapping

Chitra93 represents a TSSS as the realization of a stochastic process. To do this, Chitra93 views the data in a TSSS as the observations of an experiment consisting of sampling the state of the system under study. Thus the set of all system states in a TSSS, denoted $\mathcal{S}$, are considered to be the set of all possible outcomes of the experiment. CHITRA93 defines a random variable, denoted X, which maps $\mathcal{S}$ to non-negative integers, called *model states.* Thus CHITRA93 *defines a mapping X from system to model states.* Let $\mathcal{M}$ denote the set of non-negative integers forming the range of mapping X. Letting t denote a time bounded by the smallest and largest timestamp in a TSSS, the set $\{X(t)\}$ represents a TSSS as the realization of a stochastic process.

CHITRA93 uses the following algorithm to construct mapping X. Create a list of system states $S_0, S_1, \ldots, S_{L-1}$. Scan the list starting at S_0, and delete every element that duplicates an earlier element. Let m denote the length of the resultant list. Let $s_0, s_1, \ldots, s_{m-1}$ denote the resultant list. (Note that $\mathcal{S} = \{s_0, s_1, \ldots, s_{m-1}\}$.) Then X is defined as $X(s_i) = i$. Therefore the model state space, $\mathcal{M}$, is the integers $0, 1, \ldots, m-1$. (However, we will see in Chapter 4 that the mapping X and set $\mathcal{M}$ change each time a transform is applied.)

As an example, in Fig. 1.5, $X(<6,6>) = 0$, $X(<0,6>) = 1$, $X(<1,6>) = 2$, and so on. As a second example, suppose the TSSS for another system contains state sequence <100>, <104>, <100>, <121>. Then $X(<100>) = 0$, $X(<104>) = 1$, and $X(<121>) = 2$.

CHITRA93's views and models are displayed in terms of model states, rather than system states. This is because the model state space is guaranteed to be one-dimensional, while the system state space is n-dimensional (where n is specified by the "Maxcomp=n" statement in a .csl file (e.g., Fig. 1.5)). One dimension is easier to visualize and model than an arbitrary number of dimensions.

1.4.1.2 Display File: Internal Representation of a TSSS

When a TSSS is opened, CHITRA93 defines the initial mapping X, and then represents the TSSS by two sequences: an *occupancy time sequence* and a *model state sequence.* Trace file (E_0, S_0), (E_1, S_1), ..., (E_{L-1}, S_{L-1}) is represented as occupancy time sequence $E_1 - E_0, E_2 - E_1, \ldots, E_{L-1} - E_{L-2}$ and model state sequence $M_0, M_1, \ldots, M_{L-2}$, where $M_i = X(S_i)$. Therefore:

> *the last system state, S_{L-1} (e.g., <7,7> at time 2347123609 in Fig. 1.5), is excluded from all visualizations of the TSSS and from all statistics reported in all models of the TSSS.*

As part of the "Open" operation invoked from the root window, CHITRA93 will construct one *display file* for each TSSS opened, containing the occupancy time and model state sequences. Thus CHITRA93 converts a CBN file to a display file. Display files are always stored in the /tmp file system. A display file name is prefixed by "cHi" and constructed automatically by CHITRA93. When CHITRA93 starts, it automatically removes all files in /tmp starting with the "cHi" prefix that you own so that old display files from a CHITRA93 session that did not terminate normally will not accumulate in /tmp. In addition, display files are automatically deleted when a user *closes* an ensemble or when CHITRA93 exits normally (§1.7).

If the user selects "Cancel" in the dialog box of Fig. 1.8, then the CBN file is automatically deleted when the display file is created.

Note that because CHITRA93 converts a .csl file to a CBN file, and then converts the CBN file to a display file, one must have disk space equal to roughly three times the .csl file size to open the .csl file.

1.4.1.3 Remapping to TSSS's to Start at Time Zero

When CHITRA93 maps the TSSS to a model state and occupancy time sequence, it does not record the initial timestamp E_0 in the original TSSS. Therefore all visualizations of the TSSS have zero as their initial times, rather than E_0.

1.4.2 Root Window Appearance After Opening .csl File

We have so far opened one ensemble containing a single .csl file. At this point, the only CHITRA93 window remaining on the screen is the root window. The top pane lists all TSSS's which are currently open in the list box labeled, "Active ensembles". The top pane now contains one entry, for file "2_a_10_200_1.csl" (Fig. 1.10). Each line in the top pane has the form <Ensemble id>::<CSL file name>::<Ensemble, CBN, or display file name>. We now consider the meaning of the line

> 0::/tmp_mnt/cruncher/u3/abrams/Chitra93/examples/Diners/-
> 2_a_10_200_1.csl::/tmp/cHiBAAa04910

in Fig. 1.10. Each ensemble is assigned an identifier (or id). The "0" means that we opened ensemble number zero. The "/tmp_mnt/cruncher/u3/abrams/-Chitra93/examples/Diners/2_a_10_200_1.csl" names the .csl file comprising ensemble zero. The final field, "<Ensemble, CBN, or display file name>", contains (1) a name of the form "Ensemble*i*" where i is a small integer, if the ensemble contains two or more TSSS's (not discussed until Chapter 3), (2) the CBN file name, if the CBN file is not stored in /tmp, or (3) the display file name, if neither (1) nor (2) hold. In our present case, the ensemble contains one TSSS (hence (1) does not hold) and we selected "Cancel" in the dialog box of Fig. 1.8 so that CHITRA93 stored the CBN file automatically in /tmp (hence (2) does not hold). Therefore the display file name, in this case "/tmp/cHiBAAa04964", is present in the root window. (The display file name will change each time CHITRA93 is run.)

1.4.3 Displaying Information About an Ensemble

Next we will open a window that contains summary information about the TSSS that was just opened.

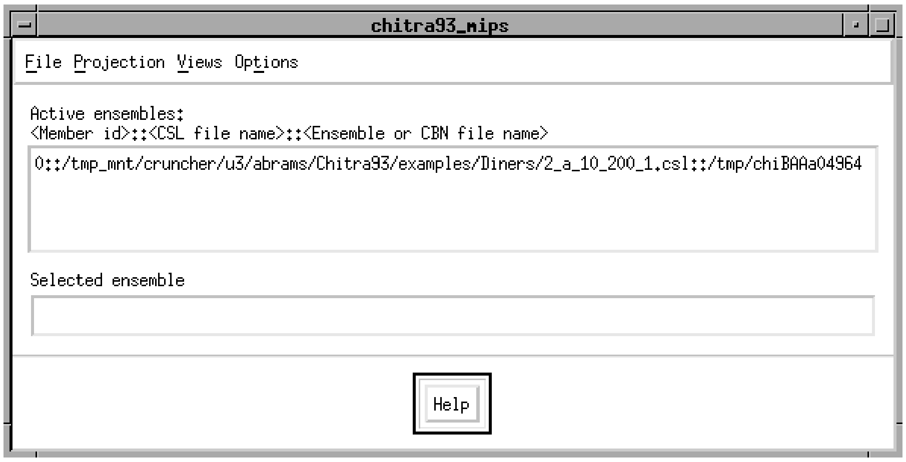


Figure 1.10: Root window after one .csl file is opened; compare to Fig. 1.1.

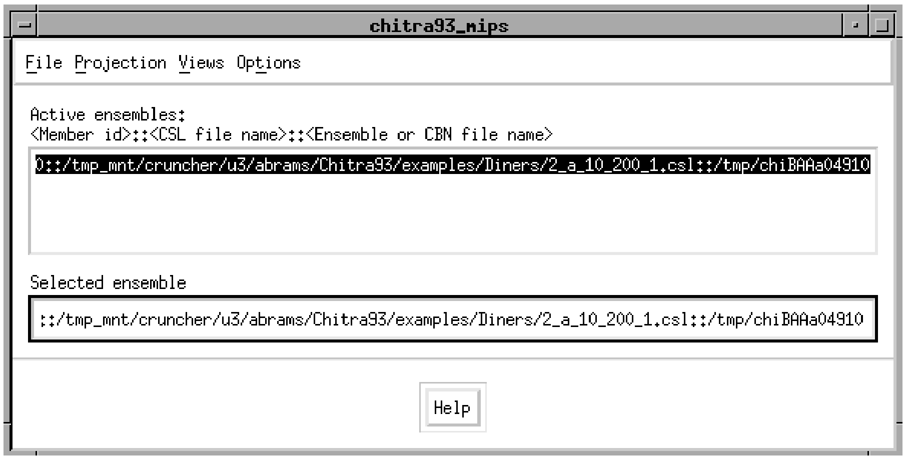


Figure 1.11: Appearance of root window after selecting a TSSS (compare to Fig. 1.1).

1. Move the mouse into the root window and click on ensemble 0. As a result, the first line in the top pane should be highlighted, and the line should be duplicated in the bottom pane, marked "Selected ensemble", as illustrated in Fig. 1.11).

2. Select "About ensemble" under the "File" menu in the root window. The window in Fig. 1.12 should be drawn on the screen. The first line says

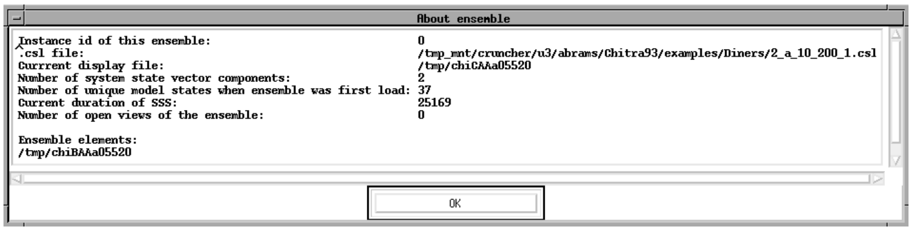


Figure 1.12: Result of selecting “File → About ensemble” in root window after opening a .csl file but not saving the .cbn file.

that this is ensemble zero, which agrees with the first field of the line in the list box in the root window (Fig. 1.10). The second and third lines give the .csl file name and the display file name; these two lines agree again with the top line in the root window list box. The fourth line in Fig. 1.12 says that the original .csl file contains system state vectors with 2 components (namely the states of the two philosophers). The fifth line says that the number elements in the set of model states, $\mathcal{M}$, is 37. Thus there are 37 unique system states in the original .csl file. The sixth line gives the duration of the TSSS, which is $E_{L-1} - E_0$. From Fig. 1.5, this is $2347123609 - 2347098440 = 25169$. The seventh line says that there are zero views of the ensemble open; we will open views of the ensemble in §1.5. The bottom of the window displays a list of .cbn files (if they are not stored in /tmp) or display file names representing the TSSS's in the ensemble. Because the ensemble contains one TSSS, and because we hit Cancel in Fig. 1.8, a single display file name (matching the name on the first line in the list box of the root window (Fig. 1.10)) is listed.

Advanced hints

- Certain menu items can be invoked without the mouse using the keyboard equivalents listed to the right of the menu choices. For example, Fig. 1.6 shows that holding the "Shift" key down while pressing the letter "O" key invokes the ".csl" menu item under the "open" menu. Note that the "File" menu need not be visible for this keyboard equivalent to work. As another example, holding the "Alternate" key while pressing "e" exits CHITRA93.
- The underlined characters in each menu item indicate another way that one may invoke menu choices without the mouse. When no menus are visible, holding the "Alternate" key and hitting one of the F, P, V, or t keys opens one of the four menus (for File, Projection, Views, or Options, respectively). With the menu visible, holding the "Shift" key and hitting the key for the underlined character (e.g., hit the letter "e" if the "File" menu is visible) invokes a menu choice (e.g., the "Exit" menu item).

1.5 How to Visualize One TSSS

A TSSS is visualized by first selecting the ensemble name containing the TSSS (i.e., Step 1 in §1.4.3), and then selecting one of the menu items under the "Views" menu in the root window. The menu choices under "Views" are listed in Fig. 1.2. Of the "Views" menu choices, the "2D", "FFT", and "Text" choices represent the possible visualizations. We start by describing "Text", because it reinforces the concepts introduced in §1.4.1. You should remember the following rule: Whenever you select a menu item under "Views", the menu item applies to whatever ensemble is listed in the "Selected ensemble" pane of the root window. If the "Selected ensemble" pane is empty, then selecting a menu item under "Views" does nothing.

A common error
If you select a menu item under "Views" in the root window, and nothing happens, it is probably because an ensemble name was not selected in the root window. An ensemble is "selected" if its name is highlighted in the top pane and duplicated in the bottom pane of the root window.

1.5.1 Text View

Selecting any view from "Views" in the root window results in opening of a new window containing the output of the view. Select the "Text" menu item

Menu item	*Description*
Open	Opens one or more files, each containing a TSSS.
Open→.csl	Read a single human readable, .csl file, from disk into CHITRA93. Also create a machine readable .cbn file corresponding to .csl file.
Open→.cbn	Read one or more machine readable .cbn file corresponding to a .csl file, from disk into CHITRA93. The set of files selected defines an ensemble.
Open→.xls	Not implemented.
Save ensemble	Operation applies to the ensemble selected in the root window. Writes a file whose name ends in .xls and contains the absolute file name of each .csl or .cbn file comprising the ensemble, along with a list of transforms applied so far to the ensemble.
Close ensemble	If there are views (windows) open for the selected ensemble, a dialog box will come up to warn the user that there are views not closed. Proceeding with the close operation will close all open views for the selected ensemble, releases storage in CHITRA93 associated with a previously opened .csl or .cbn file, including the temporary .cbn file (if any) used to store a .csl file. This does *not* affect the copy of the .csl or .cbn file on disk.
About ensemble	Pops up a box listing facts about the selected ensemble.
About Chitra	Pops up a box listing facts listing authors, version number, and additional information about about the version of CHITRA93 that is executing.
Exit	If there are views open, a dialog similar to that of choosing Close ensemble will appear to warn the user of opened views. If the user proceeds with the Exit operation, another dialog box will warn users that there are still open ensembles. Further proceeding with the Exit operation will command CHITRA93 to release storage associated with previously opened .csl or .cbn file, including the temporary .cbn file (if any) used to store a .csl file. If all views have been closed and all .csl and .cbn files opened into Chitra have been deleted, CHITRA93 will be terminated without prompting the user.

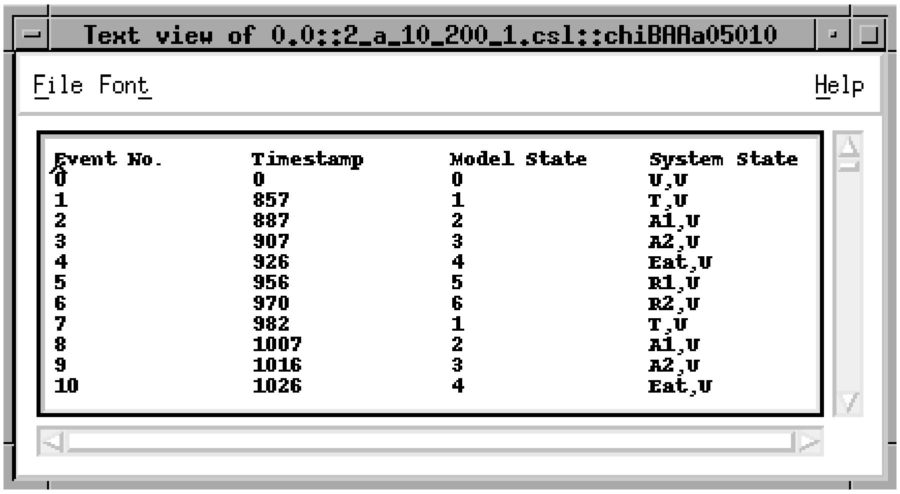
Table 1.1: Menu choices under “File” in the root window.

Menu item		*Description*
2D		Open a 2D view of the selected ensemble that maps model states to the y-axis. (See §1.5.3.)
	2D→Time	Open a 2D view with time as the x-axis.
2D	2D→Event	Open a 2D view with event number as the x-axis.
FFT		Open a fast Fourier transform view of the selected ensemble. (See §1.5.4.)
	FFT→Time	Open an FFT view based on the graph in 2D → Time view.
	FFT→Event	Open an FFT view based on the graph in 2D → Event view.
Text		Open a textual view of the ensemble. Displays each element of the TSSS as a four-tuple: event number, timestamp, model state, and system state. (See §1.5.1.)
Homogeneity		Open a view that tests the homogeniety of the selected ensemble. (See Chapter 3.)
	Homogeneity→ Adjacency matrix	Display a homogeneity matrix for the ensemble.
	Homogeneity→ Kruskal-Wallis rank sum	Test the hypothesis that all occupancy time samples of one selected state in each ensemble member are drawn from the same distribution.
Model		Open a model view of the selected ensemble.
	Model→ Continuous time semi-Markov	Show a continuous time semi-Markov model of the selected ensemble. The model predicts the next state as a function of the current state. (See §1.6.)
	Model→ CHAID-based	Show a modelb of the selected ensemble based on the CHAID method. The model identifies the best predicting state for each selected state, and then shows the transition probabilities in terms of the best predicting states. (See Chapter 5.)

Table 1.2: Menu choices under "Views" in the root window.

under "Views" in the root window. You should see a new window appear on the screen similar to Fig. 1.13.

The window banner in Fig. 1.13 contains "Text view of 0.0". Hence the view shows one of CHITRA93's views – the text view – and the data being viewed is a single TSSS – trace 0 in ensemble number 0. Text view lists on successive



Event No.	Timestamp	Model State	System State
0	0	0	U,U
1	857	1	T,U
2	887	2	A1,U
3	907	3	A2,U
4	926	4	Eat,U
5	956	5	R1,U
6	970	6	R2,U
7	982	1	T,U
8	1007	2	A1,U
9	1016	3	A2,U
10	1026	4	Eat,U

Figure 1.13: Text view of Fig. 1.5.

lines, for each state S_i in a portion of the viewed TSSS, the four quantities i, E_i, $X(S_i)$, and S_i. Thus the right hand column contains the TSSS. Compare this view with Fig. 1.5.

The preceding paragraph uses the term "a portion of" because the vertical scrollbar (i.e., the region with the up and down arrors) in Fig. 1.13 shows that only a small portion of the start of the file is visible in the window. In contrast, the horizontal scrollbar (i.e., the region with the left and right arrows) shows that almost the entire width of the text view is visible.

Try clicking on the down arrow to scroll the text down to look at more of the TSSS in the window. A faster way to scroll is to drag the small white box in the slider while holding the left mouse button down. Move the box all the way to the bottom of the slider, and observe that the final state is <7,5> with entry time 2347123599. This state is the second to last entry in the .csl file of Fig. 1.5, and shows that the final state in Fig. 1.5 (<7,7> at time 2347123609) is omitted, as was explained in §1.4.1.2.

The text view has several uses. First, it provides a way to see the "raw" TSSS without leaving CHITRA93 to use a Unix utility, such as `more` or a text editor. Second, it enumerates the system to model state mapping (e.g., columns 3 and 4 in Fig. 1.13). Finally, we will see in Chapter 4 that the textual listing is helpful in showing how the TSSS is modified by transforms.

1.5.2 The File and Fonts Menus in Each View

Every view has at least two menus: "File" and "Font", explained in Tables 1.3 and 1.4, respectively. First try using the "Font" menu to change the font size of the text view window. They try closing the window by selecting "File → Close this view". The text view window should vanish.

If any of small, medium, or large fonts defined in file `.chitra_fonts` (i.e., the second, third, or fourth font name in file `.chitra_fonts`) do not exist (and hence are listed in a box similar to Fig. 1.3 on 18 when CHITRA93 starts execution) then the default font (i.e., the first font listed in file `.chitra_fonts`) will be substituted. In such a case the Font menu will disallow the selection of the missing font.

Menu item	*Description*
Update this view	Used after you apply a transform (discussed in Chapter 4). Rewrites the view to represent the latest ensemble version.
About this view	In certain views, allows user to display the values of and/or adjust parameters associated with view
Close this view	Removes the window containing the view from the screen.

Table 1.3: Menu choices under "File" in all view windows. Some windows have additional choices.

Menu item	*Description*
Small	Redisplay the contents of the window using the "small font", which is the second font name appearing in file `.chitra_fonts` (see Fig. A.1).
Medium	Redisplay the contents of the window using the "medium font", which is the third font name appearing in file `.chitra_fonts` (see Fig. A.1).
Large	Redisplay the contents of the window using the "large font", which is the fourth font name appearing in file `.chitra_fonts` (see Fig. A.1).

Table 1.4: Menu choices under "Font" in all view windows.

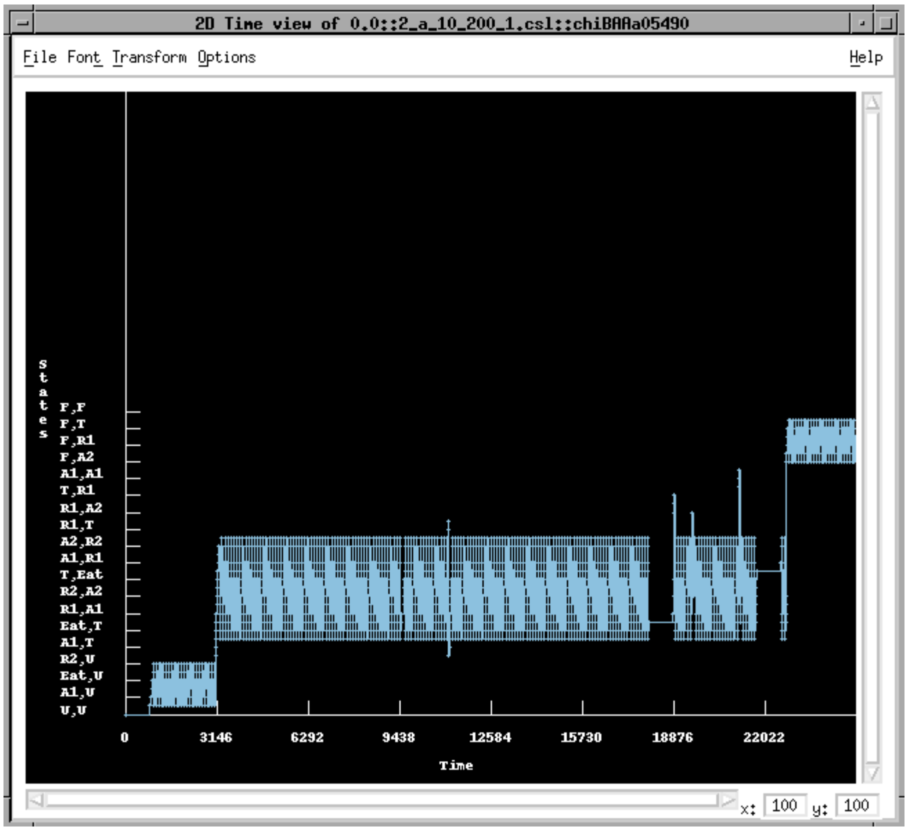


Figure 1.14: 2D time view of TSSS in Fig. 1.5.

1.5.3 2D Views

When you closed the Text View in §1.5.1, the focus should have returned to the root window (Fig. 1.11). If not, change focus to the root window by clicking on the root window banner. Be sure that ensemble 0 is still selected, as illustrated in Fig. 1.11 on page 31.

Next, select "Views → 2D". This will expose a cascaded menu with two choices: "Time" and "Event". Choose "Time". As a result, a new window should open on the screen similar in appearance to Fig. 1.14.

1.5.3.1 2D Time View

The 2D time view displays the data in a TSSS by plotting time on the x-axis and model state on the y-axis. The x-axis is continuous and represents the time interval represented by the trace displayed. The y-axis is discrete and represents integers ranging from 0 to the maximum element of model state set $\mathcal{M}$. (Recall that model states are nonnegative integers; hence their values are used as y-axis coordinates.) A y-axis coordinate of i represents model state i. Thus, referring to the text view (Fig. 1.13), y coordinates 0, 1, and 2 represent model states (U,U), (T,U), and (A1,U).

By default, the y-axis labels show the system states to which certain model states correspond. Usually not all system states are labeled due to the height of the font used to print labels.

The horizontally plotted lines represent a period during which the system remains in the corresponding y-axis state. The horizontal line length equals the state occupancy time. Element M_i of the model state sequence is represented by line segment $(E_i, M_i), (E_{i+1}, M_i)$.

To see this, we will *zoom* into the graph and examine the coordinates of the line segment endpoints in detail. Notice that the lower right corner of the window contains two numeric fields that you can change, labeled "x:" and "y:". The two numbers represent the percentage of the x and y axis that are currently visible in the window. When a 2D view is first opened, the values are 100%. Move the mouse pointer to the box labeled "x:", then double click on the number "100". The number "100" should now be highlighted. Now type "10" followed by hitting the return key. The window content will redraw to show only 10% of the x-axis. Next move the cursor to the left arrowhead in the horizontal scrollbar. Click on the left arrowhead until the scrollbar moves all the way to the left end of the horizontal slider. Now move the cursor to the number "100" to the right of "y:". Double click on the "100". Type "20" and hit return. Move the mouse pointer to the down arrowhead in the vertical scrollbar. Continue clicking on the down arrowhead until the vertical scrollbar is at the bottom end of the vertical slider. The window should now be similar in appearance to Fig. 1.15.

In Fig. 1.15, the first horizontal line has endpoints (0,0) and (857,0). This is because the system was in model state 0 (U,U) between times 0 and 857 (due to the first two lines in Fig. 1.13). Vertical plotted lines, such as the line with end points (857,0) and (857,1), represent state transitions. A cross is plotted at each point representing state entry, such as (0,0) and (857,1). Some vertical lines have multiple crosses meaning that a succession of rapid state transitions occurred that are represented by the same column of pixels in the window.

1.5.3.2 Options Menu in 2D Views

In addition to the standard "File" and "Font" menus described in §1.5.2, there are two additional menus in the 2D views: "Transform" and "Options". The "Transform" menu is explained in Chapter 4. The "Options" menu is summarized in Table 1.5.

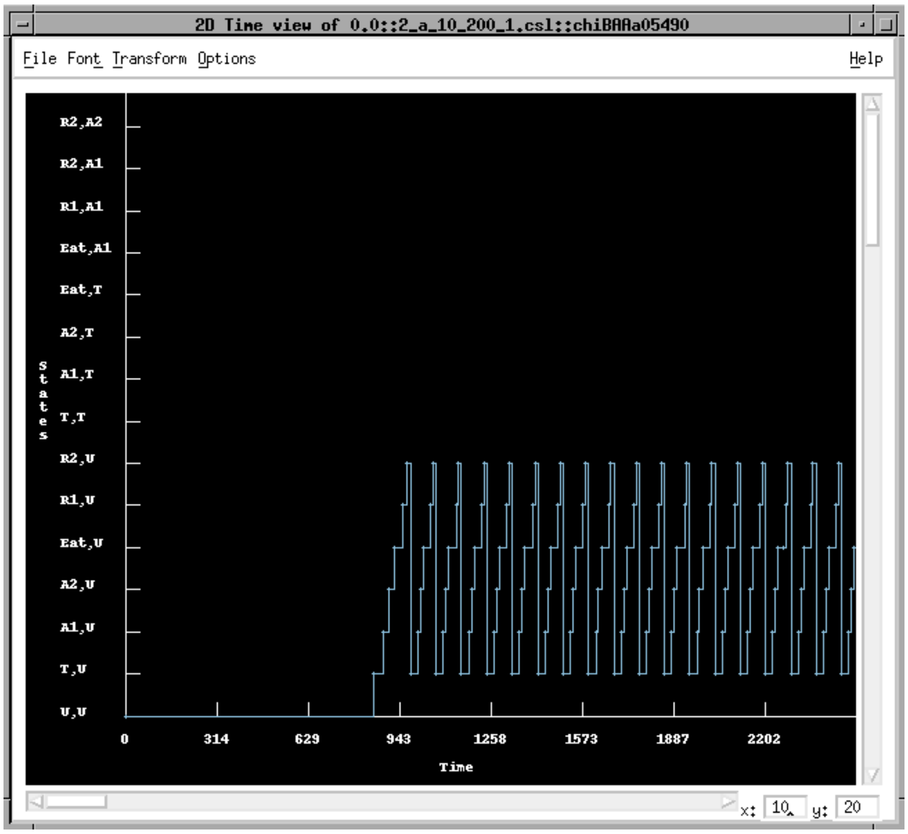


Figure 1.15: 2D time view after zooming x axis to 10% and y axis to 20% and scrolling window to lower left corner of graph.

1.5.3.3 2D Event View

In TSSS's where there is a large variance in the occupancy times of states, the widths of the horizontal lines in time view will have a large variance. This makes it hard to see states with short duration. Therefore CHITRA93 provides the event view, in which all horizontal lines have equal length (unity).

To be precise, event view differs from time view in that the x-axis represents event numbers, rather than time. Recall from §1.3 that the event number of S_i is i in SSS $S_0, S_1, \ldots, S_{L-1}$. Thus element M_i of the model state sequence is represented by line segment $(i, M_i), (i+1, M_i)$.

As an example, follow these steps:

- Change focus to the 2D time window by clicking the mouse pointer in its banner.

Menu item	*Description*
Axes	Set axes style.
Axes → Visibility → On Axes → Visibility → Off	Turns on and off the display of axes labels.
Axes → Labels → System states Axes → Labels → Model states	Selects whether the y-axis should be labeled by system or model states.
Line style	Set line style.
Line style → Connected-Horizontal/Vertical Line style → Connected- Diagonal Line style → Unconnected	Selects whether to use horizontal and vertical lines in the plot (see Fig. 1.15), diagonal lines between the crosses in the plot, or just to display the crosses.

Table 1.5: Menu choices under "Options" in 2D views.

- Double click on the number in the "x:" box in the lower right. Enter "30" and hit return. Double click on the number in the "y:" box in the lower right. Enter "50" and hit return.
- Click on the right arrowhead in the horizontal scrollbar at the bottom of the window until the slider is at the extreme right of the scrollbar. Click on the down arrowhead in the vertical scrollbar to the right of the window until the slider is at the extreme bottom of the scrollbar.
- Change focus to the root window. Select "Views → 2D → Event". Repeat Steps 1.5.3.3 and 1.5.3.3.
- Reduce both the time and event view windows to half their original height.

The result should appear similar to Fig. 1.16. Observe that the horizontal lines in the time view vary in width, to represent state occupancy times. In contrast the horizontal line widths in the event view are all equal. Therefore the event view shows a larger portion of the file than the time view at the same x-axis zoom value.

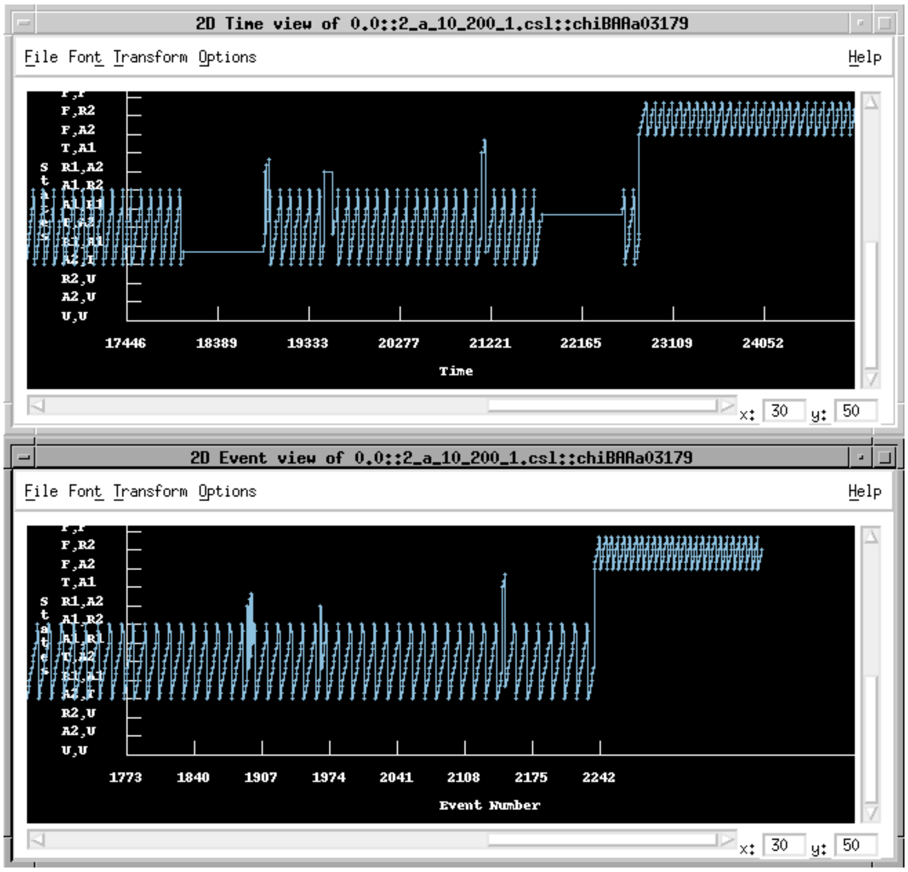


Figure 1.16: 2D event view

Advanced hints Do not drag the scrollbar in the 2D and FFT views, because CHITRA93 does not implement scrollbar dragging in these views. If you drag the scrollbar then the scrollbar will not correctly represent what portion of the graph is visible.

Given that you cannot drag a scrollbar in 2D and FFT views, you can quickly scroll by clicking not on the arrowheads bordering a scrollbar, but in the extreme ends of the scrollbar slider area just inside the arrowheads.

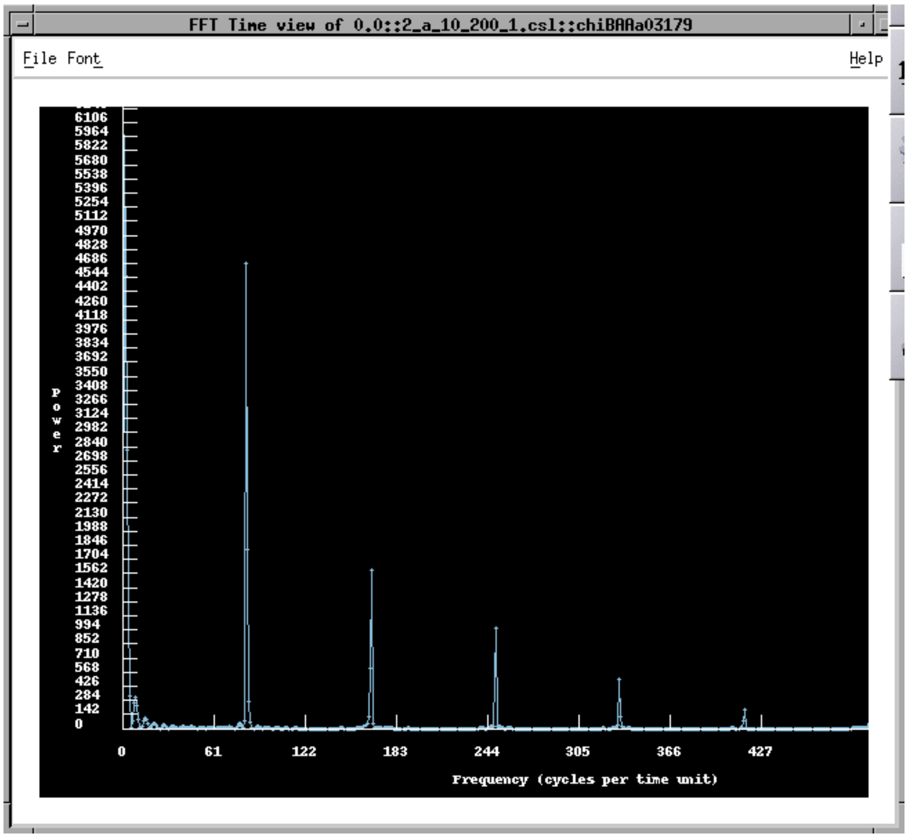


Figure 1.17: FFT time view

1.5.4 FFT Views

Just as there are 2D X-Y plots in the time domain and the event domain, there are also FFT views in the time and event domains. The FFT views plot frequency along the x-axis and power on the y-axis (Fig. 1.17). The FFT time view displays a periodogram of the discrete Fourier transform series of a TSSS instance.

Change focus to the root window and select "Time" under "FFT" under the "Views" menu. The x axis in the FFT view always ranges from zero to 500. The window shown in Fig. 1.17 should appear. There are six peaks in the curve. The declining pattern of the peaks suggests that there is a pattern of state transitions in the original TSSS.

Next select "About this view" under "File". The box that appears (Fig. 1.18) allows one to change *either* the sampling rate or the number of samples. The

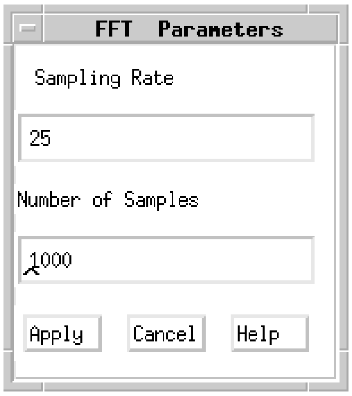


Figure 1.18: "About this view" for FFT view

sampling rate has the same time units as the TSSS over with the FFT will be computed. The default, when an FFT view is opened, is to make 1000 samples of the function in the 2D time view. The default is shown in Fig. 1.18: 1000 samples at intervals of 25 time units, for a total duration of 25,000, which equals the TSSS duration of 25,169 (from Fig. 1.12) rounded down to the nearest multiple of 1000. The time required to compute the graph in the FFT view grows as you reduce the sampling rate. For example, a sampling rate of 1 usually requires minutes or hours to compute the FFT.

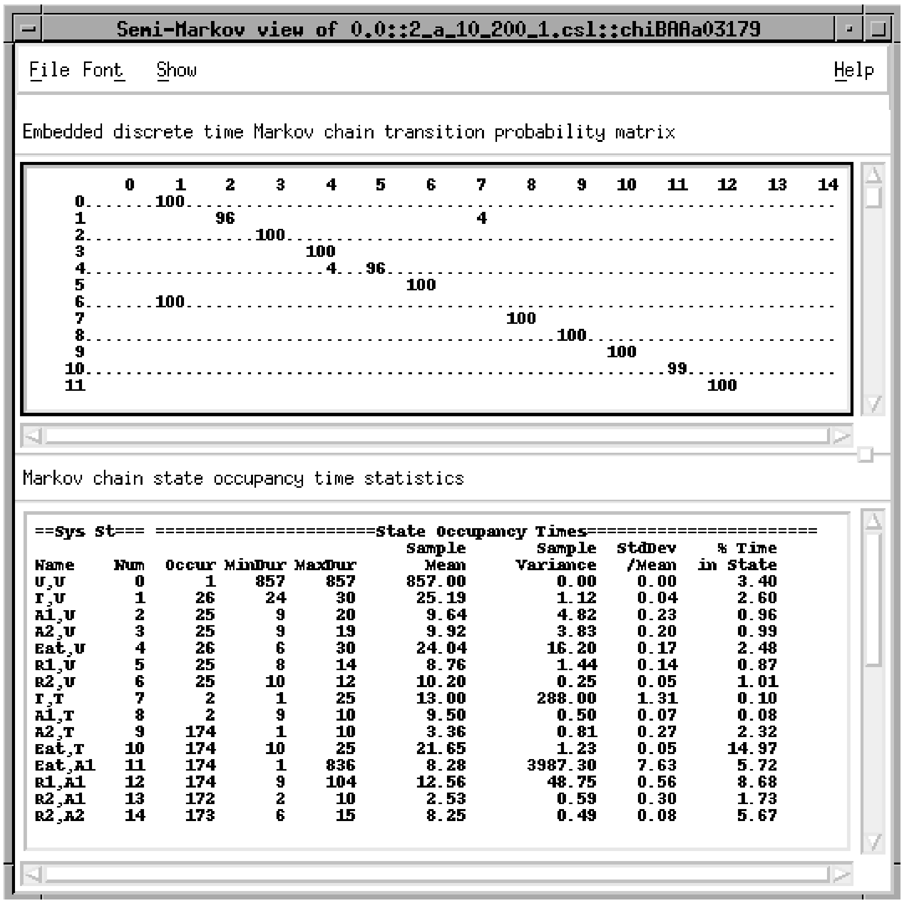
The FFT event view displays an event domain periodogram for the discrete Fourier transform series. The only difference between this view and the FFT time view is that samples for computing the discrete Fourier transform are taken on an event basis. Successive samples are taken using the values of successive system states. Therefore, the FFT event is a fast but less accurate estimate of a TSSSs periodogram behavior. The FFT event view display is similar in appearance to the FFT time view shown in Fig. 1.17.

1.6 How to Create a Semi-Markov Model of One Trace

A model is the end result of CHITRA93. This chapter discusses the most commonly used model of CHITRA93: the continuous time semi-Markov model. (An alternate model, called CHAIDS-based, is discussed in Chapter 5.)

Change focus to the root window. Be sure that ensemble 0 is selected. Then choose "Views → Continuous time semi-Markov". The result should match Fig. 1.19.

The Model view in CHITRA93 consists of two independently scrollable panes.



	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
0		100													
1			96					4							
2				100											
3					100										
4					4	96									
5							100								
6		100													
7									100						
8										100					
9											100				
10												99			
11													100		

==Sys St=== =====================State Occupancy Times======================

Name	Num	Occur	MinDur	MaxDur	Sample Mean	Sample Variance	StdDev /Mean	% Time in State
U,U	0	1	857	857	857.00	0.00	0.00	3.40
I,U	1	26	24	30	25.19	1.12	0.04	2.60
A1,U	2	25	9	20	9.64	4.82	0.23	0.96
A2,U	3	25	9	19	9.92	3.83	0.20	0.99
Eat,U	4	26	6	30	24.04	16.20	0.17	2.48
R1,U	5	25	8	14	8.76	1.44	0.14	0.87
R2,U	6	25	10	12	10.20	0.25	0.05	1.01
I,T	7	2	1	25	13.00	288.00	1.31	0.10
A1,T	8	2	9	10	9.50	0.50	0.07	0.08
A2,T	9	174	1	10	3.36	0.81	0.27	2.32
Eat,T	10	174	10	25	21.65	1.23	0.05	14.97
Eat,A1	11	174	1	836	8.28	3987.30	7.63	5.72
R1,A1	12	174	9	104	12.56	48.75	0.56	8.68
R2,A1	13	172	2	10	2.53	0.59	0.30	1.73
R2,A2	14	173	6	15	8.25	0.49	0.08	5.67

Figure 1.19: Model view.

The top pane contains the embedded discrete time Markov chain's one step transition probability matrix. The definition of a semi-Markov chain requires enumeration of the occupancy time distribution for each state in $\mathcal{M}$ and enumeration of a transition rate matrix. Instead of these, CHITRA93 displays in model view statistics of occupancy time and a one step probability transition matrix, P. The transition rate matrix can be calculated from the occupancy time statistics and P. The semi-Markov view displays P rather than the rate matrix because we feel that a user gets more intuition about the sub-ensemble behavior from the former than from the later. Matrix P contains one row and one column for each element of $\mathcal{M}$. Let $n = ||\mathcal{M}||$, and let $m_0, m_1, \ldots, m_{n-1}$ denote a sorted list of the elements of $\mathcal{M}$. (In this Chapter, $\forall i, 0 \le i < n, m_i = i$

will hold. This will not in general be true when a transform from Chapter 4) is applied. The state transition matrix is an $m \times m$ matrix where m is the number of elements in the set of model states, $\mathcal{M}$. Move the scroll bar slider in the bottom pane all the way to the bottom to see the last line to see that the largest model state is $n = 35$. The (i, j) entry in the top pane represents the number of transitions in the model state sequence corresponding to the TSSS from state m_i to m_j divided by the total number of transitions in the sequence. The fraction is an estimate of the transition probability from state m_i to m_j. The entries in the table must be divided by 100 (e.g., they are printed without a decimal point to save space). Also the remaining table entries, which are not printed, are zero.

The bottom pane contains state occupancy time statistics for each state, formatted with information for one state per line. The statistical section contains the following information about each state:

Name: Contains a system state name

Num: Contains the corresponding model state name

Occur: How many times the state to which the line corresponds occurs in the TSSS.

MinDur: The minimum occupancy time of the state in the TSSS.

MaxDur: The maximum occupancy time of the state in the TSSS.

Sample Mean: The sample mean of the occupancy times of the state in the TSSS.

Sample Variance: The sample variance of the occupancy times of the state in the TSSS.

StdDev/Mean: The ratio of sample standard deviation to sample mean occupancy time. Also known as the coefficient of variation.

% Time in State: the ratio of the sample mean to the sum of all numbers in the sample mean column.

Increase the semi-Markov window width as described in §1.2. Note that the matrix in the top pane is formatted by default to be no more than 80 columns wide. Thus a matrix is displayed as a sequence of 80 column wide "slices". To display the matrix with an unlimited number of columns in width, select "Show → All". Now observe that the horizontal scrollbar shows that you are viewing only a portion of the matrix's width. Move the scrollbar left and right to view the entire matrix.

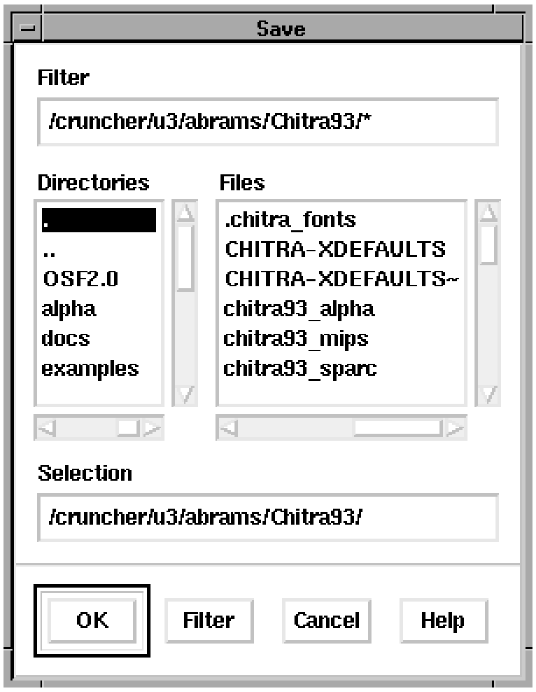


Figure 1.20: Dialog box that appears if you choose "File → Save" in a semi-Markov model window.

Writing a Model to a File
To save the model to a file, select "Save" under the "File" menu. A dialog box similar to Fig. 1.20 should appear. You can either type an entire path and file name in the "Selection" field, or use the list boxes under "Directories" and "Files" to find an existing file name. Clicking on "OK" will write the model as plain text to the named file.

1.7 Closing Ensembles and Exiting Chitra93

One can open multiple TSSS's simultaneously in CHITRA93. Change focus to the root window, select ".csl" under "Open" under the "File" menu, and select $CHITRA/examples/Diners/2_a_10_200_2.csl. This trace represents a second observation of the same program as ensemble 0. Select "Cancel" in the dialog box to name the CBN directory (Fig. 1.8). Note that the new .csl file appears as ensemble number 1 in the second line of the list box in the root window. Open a time view and compare the result to the time view of ensemble 0. Note

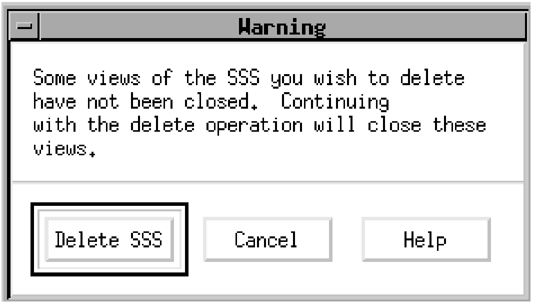


Figure 1.21: Dialog boxes that appear if you ask to close an ensemble with open views.

that you can tell which time view window corresponds to which ensemble by the window banners: one says "2D Time View of 0.0...", where 0.0 means TSSS number 0 in ensemble 0, while the other says "2D Time View of "1.0", which means TSSS number 0 in ensemble 1. If you forget what TSSS is in ensemble 0 versus 1, just look at the list box in the root window or use the "File → About ensemble" menu item. The 2D time view of the two TSSS's are very similar, but not identical. (We will explore their similarities further in Chapter 3.)

1.7.1 Closing Ensembles

The total amount of disk space required by CHITRA93 when it runs is proportional to the number of open ensembles. Therefore you may wish to *close* an ensemble which you no longer plan to analyze to reduce the disk space required for display files, and the virtual address space size of the CHITRA93 application.

Change focus to the time view of ensemble 1. Select "File → Close". The window will disappear (This frees the virtual address space required to store a bitmap image of the 2D event graph.). Change focus to the root window. Select "File → Close ensemble". This will delete the display files associated with the ensemble, and remove the ensemble name from the list box in the root window.

With the focus set to the root window, select ensemble 0. Then choose "File → Close ensemble". A box now pops up (Fig. 1.21) asking if you want CHITRA93 to automatically close the windows containing open views of the ensemble first. Select "Cancel" so that the views and the ensemble are not removed.

1.7.2 Exiting Chitra

When you complete a CHITRA93 session, select "File → Exit" in the root window. If there are any views open, Exit will display the top dialog box in Fig. 1.22 asking whether they should automatically be closed. Then, if Exit finds any

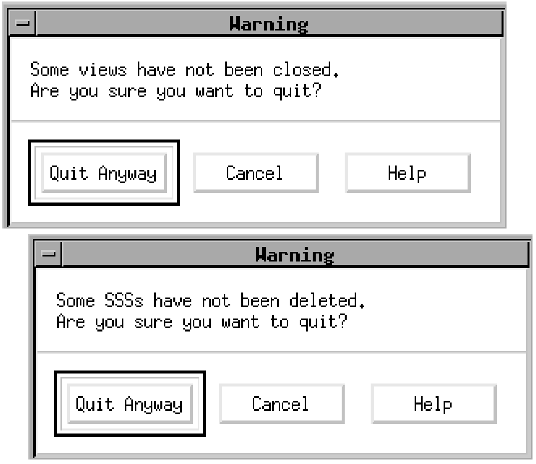


Figure 1.22: Dialog boxes that appear if you ask to close open views before allowing CHITRA93to exit.

open ensembles (e.g., ensembles listed in the root window list box), then it will display the bottom dialog box in Fig. 1.22 asking whether they should automatically be closed. When there are no open views or ensembles, Exit will remove the root window and terminate CHITRA93.

Chapter 2

Lesson 2: Saving Time and Disk Space

Trace files can be large. Large files require long times to parse when CHITRA93 converts them from .csl to CBN format. Large files also require lots of disk space. This short lesson explains a way to cope with both problems.

Part of the information in this lesson is also a prerequisite to our discussion of how to analyze ensembles of traces in Chapter 3.

2.1 Saving Time by Loading Traces with .cbn Files

Recall from §1.4 that when a user selects "Open → .csl" in the root window, CHITRA93 must now convert the .csl file to a machine readable form, called a *Chitra Binary Format* (CBN) file. Most of the time required to open a .csl file is spent performing this conversion. Thus CHITRA93 allows one to save the CBN file using a filename that ends with the suffix ".cbn". Then, in a future CHITRA93 session, you can open the .cbn file rather than the .csl file, avoiding the .csl to .cbn conversion time.

There is one caveat in saving .cbn files: because a .cbn file is machine readable, it is *not* portable across platforms. If you save a .cbn file on one platform (e.g., a SPARCstation), and then try opening it on another (e.g., a DECstation), the two warning boxes in Fig. 2.1 will appear. (Usually the smaller and less informative warning box appears on top of the larger one; be sure to move the warning box on the top so that you can see the more meaningful message in the larger box.)

To identify which platform a .cbn file was created on, use the following convention: Store .cbn files in a sub-directory whose name is a platform name. For

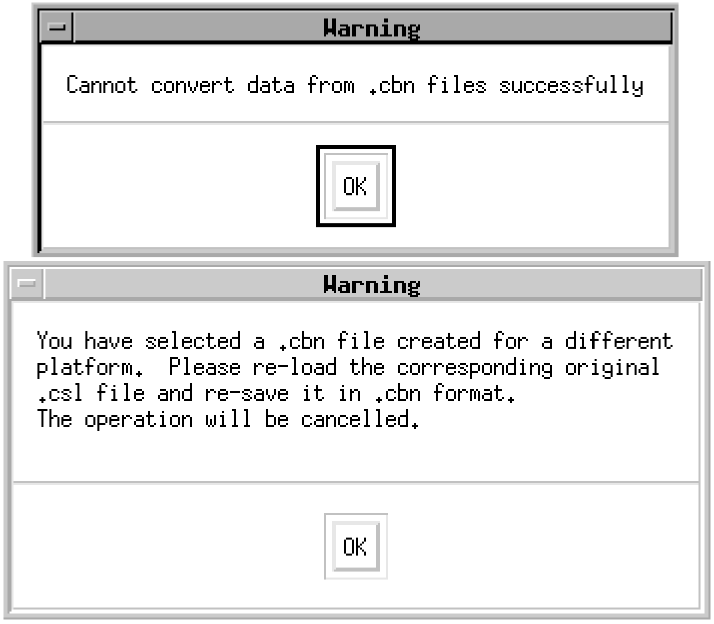


Figure 2.1: Dialog box that appears if you attempt to open a .cbn file created on another platform.

example, the $Chitra/examples directory contains sub-directories `alpha`, `mips`, and `sparc`, and are used to store .cbn files created when running CHITRA93 on each of these platforms.

Start CHITRA93 (steps 1 and 6 of §1.1). Select "Open → .csl". Select $CHITRA/examples/Diners/2_a_10_200_1.csl and hit "OK" to display the dialog box in Fig. 1.8 on page 27. Do not select "Cancel". Rather, select $CHITRA/examples in the left list box and your platform name (e.g., "mips") in the right list box. Select "OK" to create file 2_a_10_200_1.cbn in directory $CHITRA/examples/⟨platform⟩. Remember,

- Select a *directory* name in the right list box, not a *file* name.
- Be sure that you have write permission for the directory name you choose for the location of the .cbn file.
- Do not hit "return" to dismiss the box, because "return" is bound to "Cancel". Click on "OK" with the mouse.

Next we will verify in three ways that you have in fact created the .cbn file.

Figure 2.2: Root window appearance after opening a .csl file but saving the .cbn file to a directory of your choice.

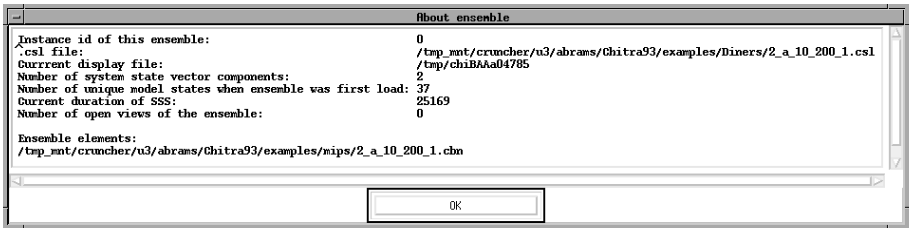


Figure 2.3: Result of selecting "File → About ensemble" in root window after opening a .csl file but saving the .cbn file to a directory of your choice.

1. Type `ls -l $CHITRA/examples/⟨platform⟩/2_a_10_200_1.cbn` in a terminal window and verify that file 2_a_10_200_1 exists and that its creation time and date are the current time and date.

2. Check that "$CHITRA/examples/⟨platform⟩/2_a_10_200_1.cbn" appears in the root window at the right end of the first line in the top list box, as shown in Fig. 2.2. Compare this to Fig. 1.10 on page 31, which contains a display file name in /tmp.

3. Select ensemble 0 in the root window, and then choose menu item "File → About ensemble". The .cbn file is now listed at the bottom of the window under "Ensemble elements". Also, the window shows the three files with three representations of the trace: the .csl file (on the second line), the display file (on the third line: /tmp/cHiBAAa047), and the .cbn file.

Next, select "OK" in the "About ensemble" box. Ensemble 0 should still be selected in the root window. Close ensemble 0 by selecting menu item "File → Close ensemble" in the root window.

Now we will reopen the TSSS, but this time as a .cbn file, to see how the procedure differs from .csl files. Select "Open → .cbn" from the root window. The file selection box appears identical to that for "Open → .csl" (Fig. 1.7 on page 26) except that the filter string at the top of the box ends in "*.cbn*" rather than "*.csl*". Select $CHITRA/examples/⟨platform⟩/2_a_10_200_1.cbn and select "OK". You should notice that the window disappears and, almost immediately, the root window displays the .cbn file in its list box. (Compare this to the delay when you used "Open → .csl".) Note that the ensemble is numbered 1, rather than 0. CHITRA93 does not reuse ensemble numbers during a session; thus ensemble i resulted from the i-th execution of an Open menu command. Note that the root window shows "NO .csl" in the field for "< CSL file name >" in the list box (Fig. 2.4). Select ensemble 1 in the root window, and then select menu item "File → About ensemble". Note that the second line in the "About ensemble" box also lists "NO .csl" as the .csl file name and that the last line in the box lists the .cbn file rather than a display file. Click on "OK" in the "About ensemble" box. Change focus to the root window and select "Views → 2D Time". Note that the banner of the resultant 2D window also lists "NO .csl" as the .csl file name (not illustrated).

Change focus to the root window and select "File → Exit" to exit CHITRA93. The two dialog boxes in Fig. 1.22 will appear because we left the 2D time view open. Select "Quit Anyway" in each box and CHITRA93 will terminate.

2.2 Saving Disk Space by Using Compressed Trace Files

To save disk space, one can use either the "compress" command or gnu zip ("gzip") to compress a .csl or .cbn file. All compressed .cbn files must be located within a directory for which you have write permission. The file selection dialog boxes used for the "Open → .csl" (Fig. 1.7 on page 26) and "Open → .cbn" menu choices in the root window display files with the suffix ".csl*" and ".cbn*", respectively. Therefore compressed files (ending in ".csl.Z" or ".cbn.Z") and gzip'd files (ending in ".csl.gz" or ".cbn.gz") will be displayed.

The penalty for compressing files is extra time required to uncompress files during each "Open" operation.

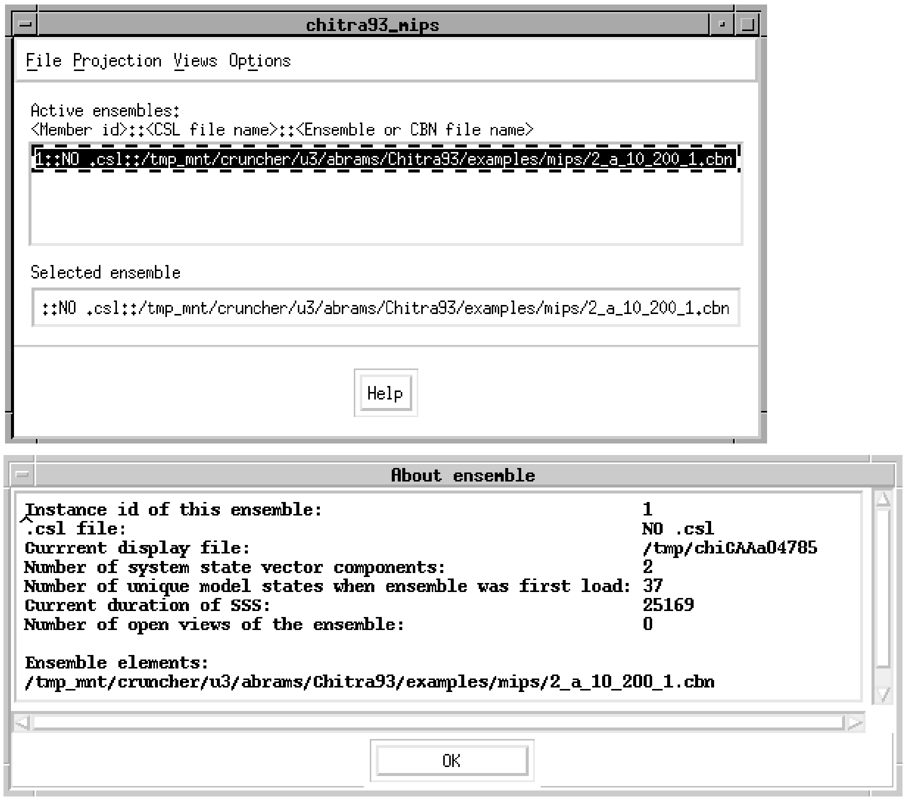


Figure 2.4: Result of selecting "File → Open .cbn" and later "File → About ensemble" in root window.

Chapter 3

Lesson 3: Analyzing an Ensemble of Trace Files

A single TSSS represents a single period of observation of a system. It is dangerous to draw conclusions on the system based on one TSSS alone. For example, a TSSS might be recorded from a parallel computer during the middle of the night when there was only one user running a program. Such a TSSS is obviously not representative of what happens during the daytime when multiple users are running programs. Looked at from the perspective of a statistician, one would never draw a conclusion or test a hypothesis based on a sample size of one! Therefore one should collect and analyze multiple TSSS's, called an *ensemble* in CHITRA93. Recall from §1.3 that an ensemble is a set of TSSS's.

Compared to our analysis of a single TSSS in Chapter 1, three new issues arise in analyzing ensembles of TSSS's. First, how is an ensemble visualized? Second, is the ensemble *homogeneous*, or should it be partitioned into homogeneous sub-ensembles? Informally, an ensemble is "homogeneous" if it represents the "same" behavior. For example, if we collect two TSSS's at night and three during the day, and we opened all five TSSS's in CHITRA93, we might expect that CHITRA93's homogeneity tests would suggest partitioning the ensemble into two sub-ensembles, consisting of the day versus nighttime TSSS's. Third, how do we generate a model fitting each homogeneous sub-ensemble?

3.1 Opening Ensembles

CHITRA93 provides a natural method of specifying which TSSS's comprise an ensemble: *multiple selection* in the "Open → .cbn" dialog box (Fig. 1.8 on page 27). CHITRA93 places two constraints on the files that may form an ensemble:

1. Only .cbn files (not .csl files) can form an ensemble.
2. Only files in the same directory can form an ensemble.

To illustrate, we will form an ensemble from three observations of the Dining Philosophers program used in earlier chapters. In Chapter 2 we created one .cbn file, from $CHITRA/examples/Diners/2_a_10_200_1.csl. Recall that from the naming convention given in §1.3, the files 2_a_10_200_2.csl and 2_a_10_200_3.csl represent two other observations of the same two philosopher program.

Start CHITRA93. First convert 2_a_10_200_2.csl and 2_a_10_200_3.csl to .cbn files by the procedure given in §2.1. Directory $CHITRA/examples/⟨platform⟩ contains three files: 2_a_10_200_1.cbn, 2_a_10_200_2.cbn, and 2_a_10_200_3.cbn. Change focus to the root window and close all ensembles that are open. The root window should appear as Fig. 1.1 on page 17.

Now we will open the ensemble. Select "File → Open → .cbn". Select directory $CHITRA/examples/⟨platform⟩ in the left list box. If necessary, scroll the right list box until the three .cbn files listed above are visible. To select all three TSSS's:

1. Position the mouse pointer over 2_a_10_200_1.cbn.
2. Press and release the left mouse button to highlight "2_a_10_200_1.cbn".
3. Move the mouse pointer over 2_a_10_200_3.cbn.
4. While holding down a shift key, press and release the left mouse button.

If you make an error, and the wrong set of .cbn file names is highlighted, just repeat the procedure. (A mouse click in the list box un-highlights the previous selection and highlights a new selection.) If the procedure is properly done, the three .cbn file names above should now be highlighted, as shown in Fig. 3.1. Click the mouse pointer on "OK".

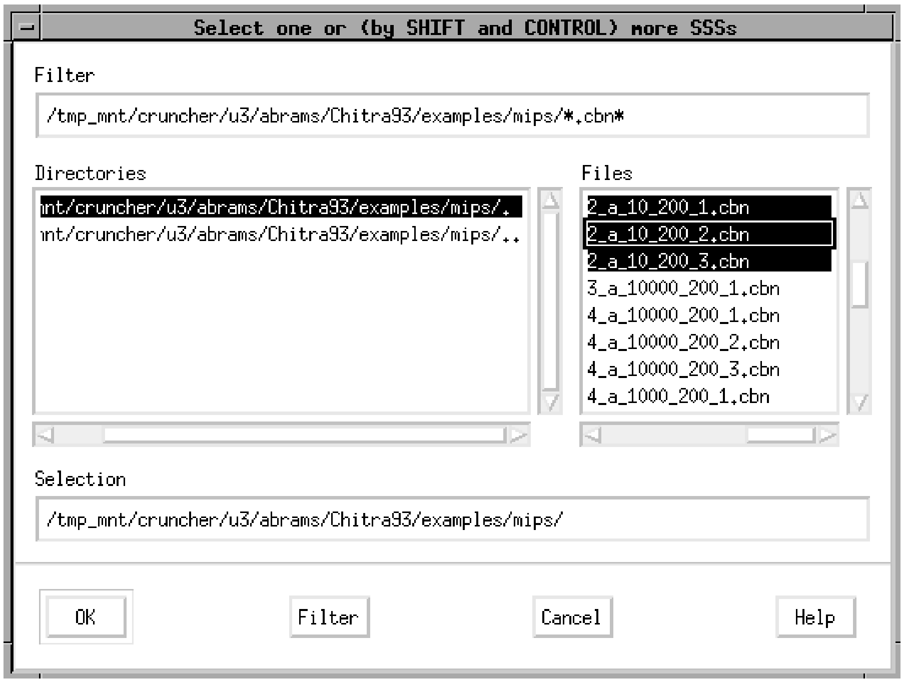


Figure 3.1: Appearance of "Open → .cbn" file dialog box to define an ensemble after selecting three .cbn files using the shift key.

Advanced hints

- Another way to select the three .cbn files in the list box is to position the pointer over the first name, then press and *hold* the left mouse button, and drag the mouse pointer down until all three names are highlighted. Then release the mouse button.

- The list box in Fig. 3.1 contains eight file names. If you wish to select more than eight file names, then resize the window height (as described in §1.2) until there is room for all the names.

- If you wish to select more file names than will fit on the screen, even after increasing the dialog box window height, or if you wish to select file names that are not contiguous, then replace the steps 3 and 4 above by holding the control key down and clicking on a sequence of file names. You are free to reposition the scrollbar during this operation, so that the file names need not simultaneously fit in the list box.

3.1.1 What Happens When an Ensemble Is Opened

§1.4.1 explains what happens when an ensemble containing a *single* TSSS is opened. When an ensemble with two or more TSSS's is opened, the procedure described in §1.4.1 varies.

System to Model State Mapping

§1.4.1.1 explained that when a single TSSS is opened, CHITRA93 defines a random variable, X, mapping the system states in the original .csl file to a subset $\mathcal{M}$ of non-negative integers called model states. When an ensemble is originally opened, CHITRA93 generates an mapping X as follows.

Informally, CHITRA93 assigns as model states successive non-negative integers the first time it encounters a system state when scanning each of TSSS's 0, 1, ... from start to end. Thus model state 0 is the first system state in TSSS number 0, and model state 1 is the next system state in TSSS 0 that does not duplicate the first system state, and so on. Formally, suppose that an ensemble contains n TSSS's, and that there are m unique system states in the n files. Create a list by concatenating the n TSSS's in increasing TSSS number order (0 followed by 1, and so on). Next, scan the resulting file from start to end, and delete any system state that duplicates a system state occurring earlier in the list. The list must have length m. Let $s_0, s_1, \ldots, s_{m-1}$ denote the resultant list. (Note that $\mathcal{S} = \{s_0, s_1, \ldots, s_{m-1}\}$.) Then X is defined as $X(s_i) = i$. Therefore the model state space, $\mathcal{M}$, is the integers $0, 1, \ldots, m-1$. (As mentioned earlier, we will see in Chapter 4 that the mapping X and set $\mathcal{M}$ change each time a transform is applied.)

3.1.2 Appearance of Root Window After Opening an Ensemble

After you clicked "OK" in the file selection dialog box of Fig. 3.1, the box should have vanished and the focus should have returned to the root window. The root window list box should contain one item, which is ensemble 2 (see Fig. 3.2). (The ensemble number is 2 because the two .csl files that you converted to .cbn earlier were ensemble numbers 0 and 1.) Note that the third field of the line contains "Ensemble 2" rather than the file name of a .cbn or display file because the ensemble is composed of three .cbn files and is represented by three display files.

3.1.3 Displaying Information About an Ensemble

Suppose you forget *which* .cbn files you selected to define the ensemble. To recall the names, select "File → About ensemble". The bottom of the window lists the three .cbn files you selected (see Fig. 3.3).

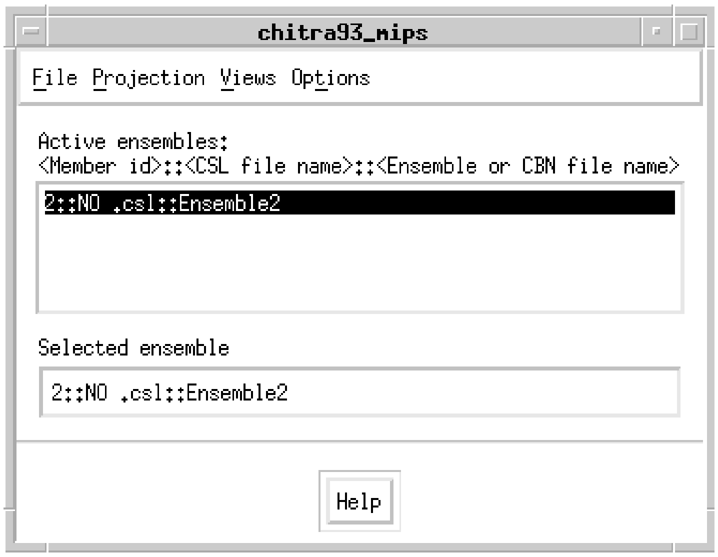


Figure 3.2: Root window appearance after opening an ensemble of two or more .cbn files.

Figure 3.3: Result of selecting "File → About ensemble" after opening an ensemble of three .cbn files.

There are two differences of note in the "About Ensemble" window for an ensemble (Fig. 3.3) with more than two members compared to the "About Ensemble window for an ensemble with a single member (Figs. 1.12 on page 32 and 2.4 on page 54). First, the number of model states listed in Fig. 3.3 is now 43, compared to 37 when we opened only member 0 in Fig. 1.12 and 2.4.

The additional 43-37=6 states are therefore present in members 1 and 2 but not in member 0. Second, field "Current duration of TSSS" shown in Figs. 1.12 and 2.4 is not displayed in Fig. 3.3. That is because the TSSS's comprising an ensemble will not have equal durations in general.

Select "OK" in the "About ensemble" window, so that the window vanishes and focus returns to the root window.

3.2 Visualizing Ensembles

We will now visualize two of the three TSSS's in the ensemble. Select ensemble 2 in the root window. Choose menu item "Views → 2D → Time". The dialog box in Fig. 3.4 appears to allow specification of which ensemble members to view, and whether to view them together or separately. Position the mouse pointer on 2_a_10_200_1.cbn and click the left mouse button, and then position the pointer on 2_a_10_200_3.cbn and hold the control key down and click the left mouse button. This will select two of the three members, as shown in Fig. 3.4. Then select "separate", which will open a separate window containing a 2D view for each ensemble member as shown in Fig. 3.5. (The "Combined" choice cannot be used with for the 2D and FFT views.) Then hit apply. As a result, two 2D time windows should open on the screen, one labeled "2.0" in the banner and the second labeled "2.2". The banners identify which window is 2_a_10_200_1.cbn versus 2_a_10_200_3.cbn.

3.3 Testing Ensembles for Homogeneity

Recall at the start of this section that one should determine if an ensemble under analysis is homogeneous, and if it is not, then the ensemble should be partitioned into mutually exclusive, exhaustive, and homogeneous subsets called sub-ensembles. This section examines ways to test the *hypothesis of homogeneity*, which is "the TSSS's in an ensemble are all realizations of the same stochastic process".

CHITRA93 provides four methods of collecting evidence as to whether the hypothesis of homogeneity is true or false: visualization, adjacency matrix, Kruskal-Wallis rank sum test, and coverage of a pattern. Each is explained below.

3.3.1 Method 1: Visualization

The most obvious thing that one can do is to simply compare the 2D or FFT views of all members in an ensemble. For example, ensemble members 0 and 2 in Fig. 3.5 does not reveal any gross difference between the two members. For example, both graphs have the same y-axis labels, and both graphs contain two intervals when the system stays in the same state for an unusually long duration

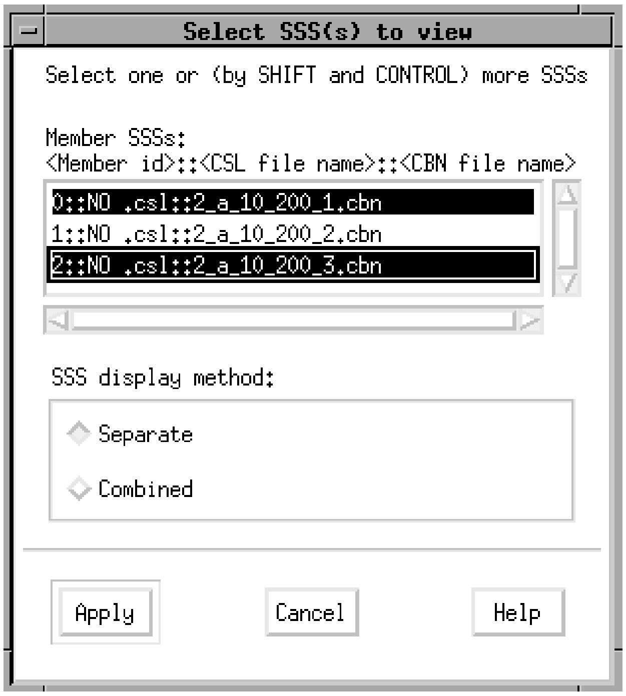


Figure 3.4: Dialog box that appears when you select any view of an ensemble containing two or more TSSS's.

(e.g., just before time 18876 and around time 22022 in member 0 of Fig. 3.5). Member 2 appears to have a few more perturbations or "spikes" in the graph for brief periods.

This provides some evidence that the hypothesis of homogeneity is true for these two members.

Comparison of visualizations has several pitfalls. First, one must visualize *all* ensemble members. This is not practical for ensembles with dozens of members. Second, many TSSS's, such as the dining philosophers TSSS's used here, contain system states that are *categorical*, not numerical. (States are numeric if a partial ordering exists among the state values.) For example, the state "eat" is not "less than" or "greater than" "think". Thus there is not a unique way to map system states to the y axis in the 2D views.[1] CHITRA93 uses model states as y-axis coordinates, and model states in turn are assigned using the algorithm described in §3.1.1 to define X. Thus two TSSS's that are not homogeneous

[1] See [4] for a further discussion of visualizing categorical versus numerical data. CHITRA94 contains visualization methods for categorical states that avoid this problem.

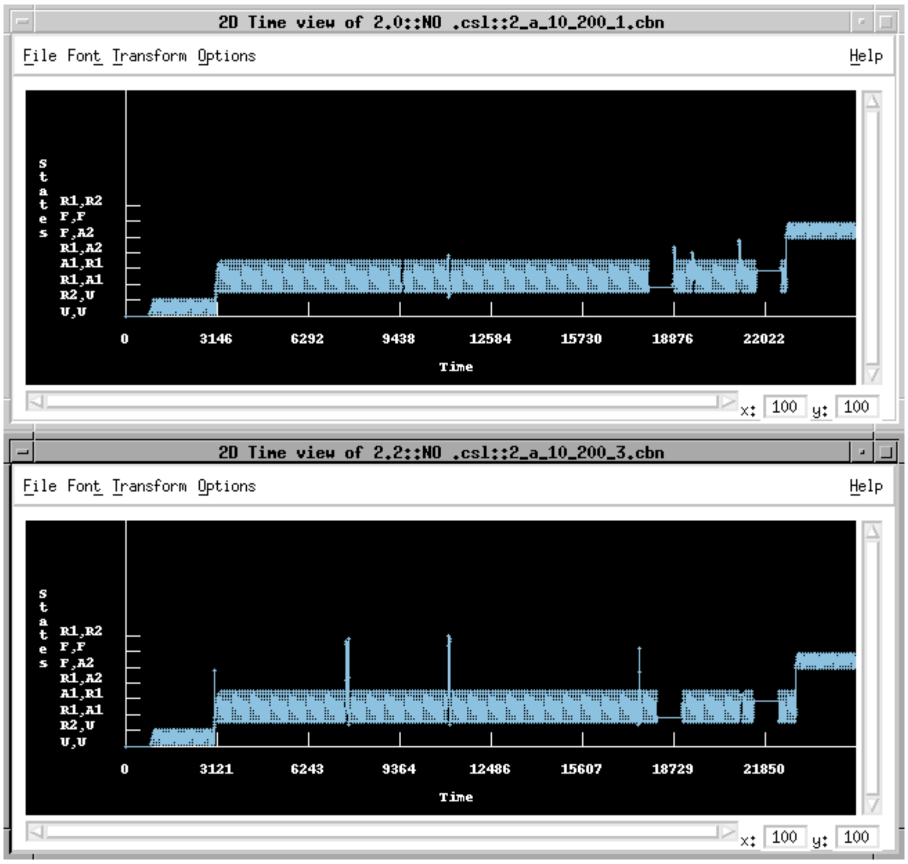


Figure 3.5: 2D time views of two of the three TSSS's in the ensemble resulting from selection in Fig. 3.4.

might by chance result in 2D views that appear similar in CHITRA93. Similarly, two TSSS's that are in fact homogeneous might by chance result in radically different 2D views.

3.3.2 Method 2: Adjacency Matrix

If an ensemble is homogeneous, then all its TSSS's should contain the same set of transitions. The adjacency matrix conveniently displays this information for an unlimited number of TSSS's in an ensemble in a single matrix.

The adjacency matrix for an ensemble is formed as follows. Let $n = ||\mathcal{M}||$, and let $m_0, m_1, \ldots, m_{n-1}$ denote a sorted list of the elements of $\mathcal{M}$. The adjacency matrix is an $n \times n$ matrix. Element i, j is equal to the percentage of the TSSS's that contain at least one transition from state m_i to m_j. Thus, if element i, j is 0, no TSSS's have a transition from model state m_i to m_j; while an entry of 100% indicates that all TSSS's contain at least one instance of the transition. Nearly all non-zero entries in the adjacency matrix of a homogeneous ensemble should be close to 100%.

Let's examine the adjacency matrix for our ensemble of three dining philosopher program TSSS's. Change focus to the root window. Be sure that ensemble 2 is selected. Select menu item "Views → Homogeneity → Adjacency matrix". In the "Select TSSS(s) to view" dialog box (Fig. 3.4), select all three ensemble members in the list box and "Combined" under "TSSS display method". Then select "Apply". The window shown in Fig. 3.6 should appear on your monitor.

Fig. 3.6 shows a portion of the adjacency matrix, for transitions into states 0 to 14 of the 43 states. Move the scrollbars to see the rest of the matrix. The only possible values for each matrix cell are 0, 0.33, 0.67, and 1, corresponding to no, one of three, two of three, or three of three TSSS's having the transition corresponding to the cell. A number of entries are 100%, supporting the hypothesis of homogeniety. However many entries are 33% or 67%, suggesting that the hypothesis is false. (We will return in §4.5 to this test after applying transforms and see that eliminating states that do not occur very often eliminates almost all 33% and 66% entries, so that in the end the adjacency matrix supports the homogeneity hypothesis.)

3.3.3 Method 3: Kruskal-Wallis Rank Sum Test

CHITRA93 contains a second homogeneity test that is a classical statistical test of the hypothesis that all samples of the occupancy time of one state in a set of TSSS's are drawn from the same distribution: the Kruskal-Wallis (KW) rank sum test [15, pp. 422–426], which is an analysis of ranks test. The result of the test is a simple "true/false" answer based on a statistical test. However, please keep the following caveat in mind:

> All statistical tests – including CHITRA93's KW test – are sometimes *wrong*. Thus when the test may say the hypothesis is true (false), even though it is really false (respectively, true).

Thus there are two types of errors. A *Type I* error in the statistics literature is for CHITRA93 to report "reject" as the test outcome, even though the hypothesis is really true. The probability of a Type I error is usually denoted by α in the statistics literature, and can be set by the CHITRA93 user. A *Type II* error is for CHITRA93 to report "accept" as the test outcome, even though the hypothesis is really false. The probability of a Type II error is denoted by β. The CHITRA93 user has no way to estimate the value of β; however keep in mind that reducing α *increases* β. This explains why the default for α in CHITRA93 is 5% rather than a smaller value, such as 1%.

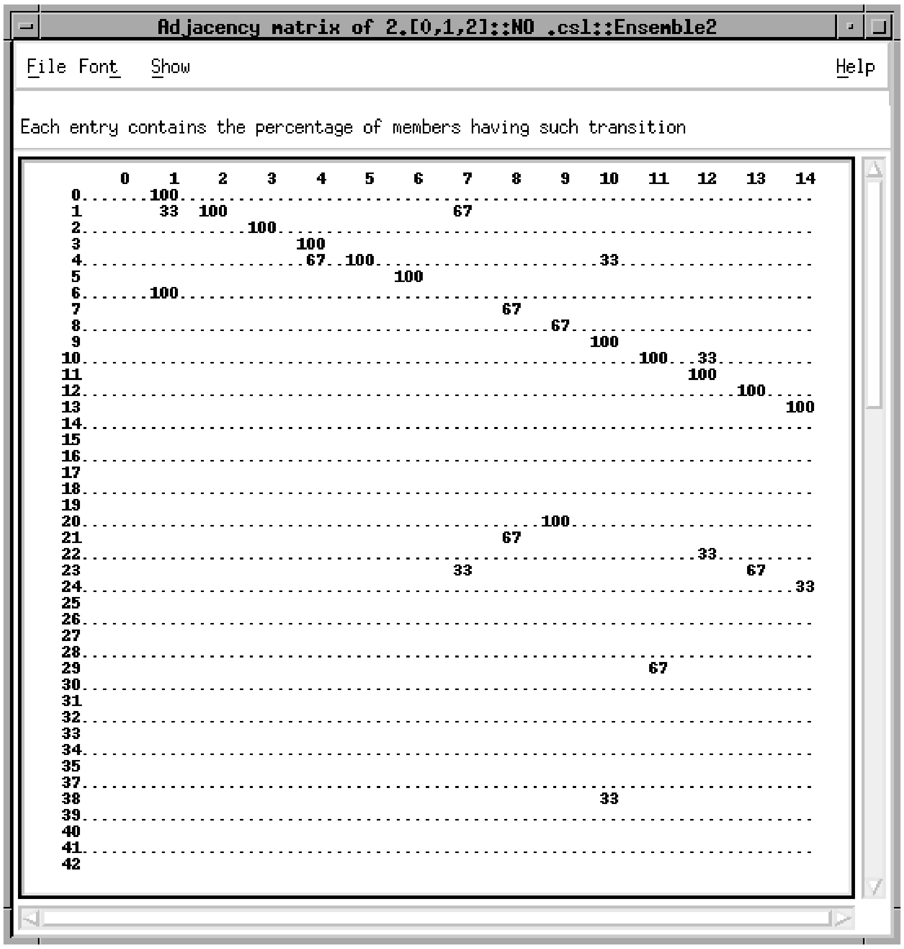


Figure 3.6: Adjacency matrix for three Dining Philosopher program TSSS's.

There are many statistical tests of homogeneity. We chose the KW test for CHITRA93 because it does not require any assumption about the distribution underlying the data. However the price for this ease of use is that more samples are required to obtain an answer with the same probability of truth than another statistical test that assumes a certain distribution.

To use the test, the user selects a single state that occurs in all members, along with a significance level (or probability of Type I error) α for the χ^2 test that underlies the KW method. The KW test is applied to a set of samples, each consisting of the set of occupancy times of each occurrence of the selected state in one member. The null hypothesis tested is that the samples in all member's

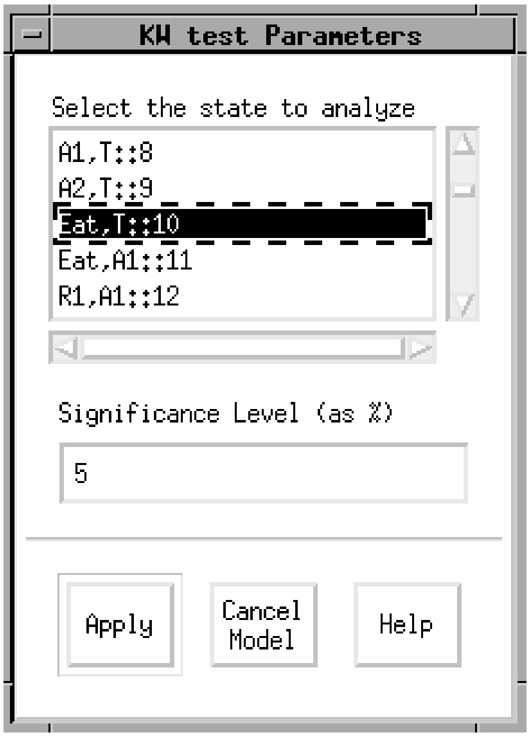


Figure 3.7: Dialog box specifying on which state's occupancy times the KW test will be performed.

are drawn from identical distributions.

Let's test the three Dining Philosopher program TSSS's for homogeneity using the KW test. Change focus to the root window. Be sure that ensemble 2 is selected. Select menu item "Views → Homogeneity → Kruskal-Wallis rank sum". In the "Select TSSS(s) to view" dialog box (Fig. 3.4), select all three ensemble members in the list box and "Combined" under "TSSS display method". Then select "Apply". The "KW test parameters" dialog box will now appear (Fig. 3.7). Now we must decide to *which* state's occupancy times the KW test will be applied. We will use the most time consuming state shown in the semi-Markov model output of Fig. 1.19 on page 45: system state Eat,T (also known as model state 10), which accounts for almost 15% of the execution time of member 0. Scroll the list box until this state appears. Select the state by clicking the mouse pointer on the state name. You must also specify a significance level, which defaults to 5%.

Now select "Apply" in Fig. 3.7. If the state selected (e.g., "Eat,T") appears

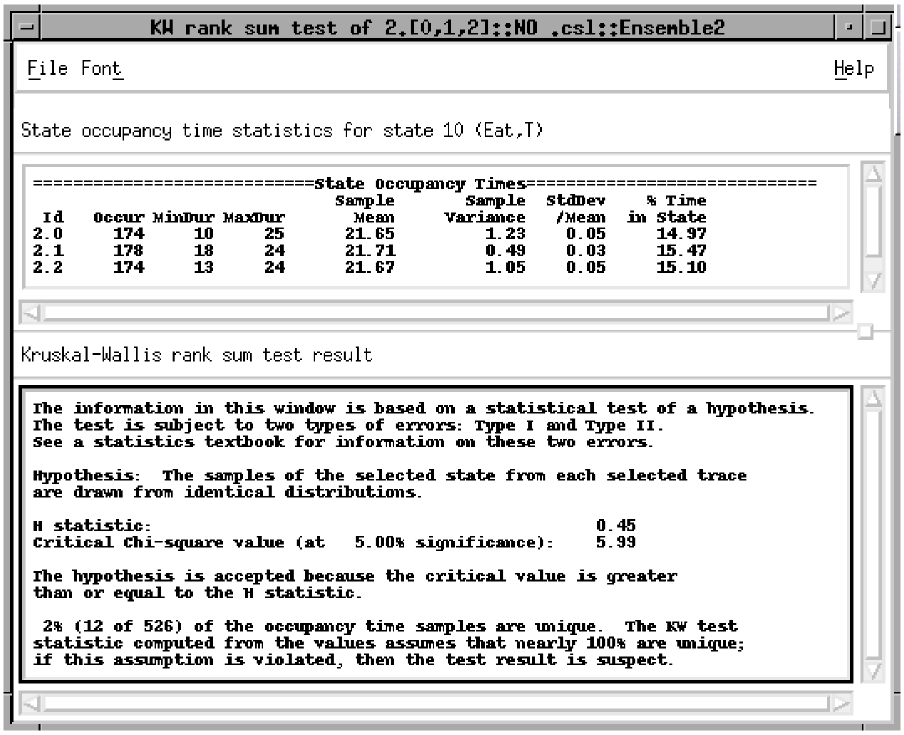


Figure 3.8: Kruskal-Wallis (KW) rank sum test for the three Dining Philosopher program TSSS's applied to state Eat,T.

in *all* members in the ensemble, then the test is evaluated and the results are presented in the window shown in Fig. 3.8. Otherwise you will see the warning box in Fig. 3.9.

The resulting window (Fig. 3.8) contains two panes. The top pane lists statistics about the occupancy time samples of state (Eat,T) in each of the three TSSS's. For example, the state occurs 178 times in member 1 and accounts for 15.47% of the execution time. In fact, all three members have similar mean occupancy times in the state. The bottom pane contains the test result. The the critical χ^2 value (5.99) exceeds the KW statistic H (0.45), and thus the hypothesis is accepted. It is a good idea to repeat the test on the occupancy time of other states.

The bottom three lines in the bottom pane indicate that the KW test implemented in version 4.122 of CHITRA93 assumes that few occupancy time samples are unique. This is violated by the Dining Philosopher program TSSS's, because only 2% of the values are unique. One may therefore want to apply other sta-

Figure 3.9: Warning box that appears if the state selected for KW test does not appear in all ensemble members.

tistical tests to the occupancy times.[2] Therefore, under the "File" menu is the choice "Save occupancy times". Selecting this item will display the file dialog box to choose a file in which to save the samples which the KW test is testing. One can then import this file to a statistical software package to perform further tests.

3.3.4 Method 4: Pattern Coverage

TSSS's from computer and communication systems sometimes contain patterns of state transitions. §4.4.1 describes how CHITRA93 can search for all occurrences of a pattern. A pattern that occurs with similar frequency in each member contributes evidence that the ensemble is homogeneous.

3.4 Homogeneous Partitioning of an Ensemble

If you conclude that the entire ensemble is not homogeneous by the tests described above, then you must identify a partition and then test each partition for homogeneity. This process is illustrated in [4].

Some criteria you can use for deciding how to partition follows. With method 1 (visualization), you could visually see which members appear similar to choose a partition. With method 2 (adjacency matrix), you could see which member(s) are responsible for a near-zero percentage in the transition matrix.[3] With method 3 (KW test), you could use the statistics in the top pane to look for members with similar means and variances to choose a partition. With method 4 (pattern occurrences) you could partition an ensemble into two subsets based on which members do and do not contain a certain pattern with similar frequency.

[2] There is an alternative statistic for the KW test that works for data in which few sample values are unique; this is used in CHITRA94.

[3] A partitioning algorithm using adjacency matrices is given in [4], and demonstrates how CHITRA94 can be extended by the user to automate partitioning.

When you decide on a partitioning, you must reapply the homogeneity tests to each partition to confirm that the partitioning does in fact produce homogeneous sub-ensembles. Do this through the dialog box in Fig. 3.4 on page 61, by selecting only those members that belong to the sub-ensemble that you wish to test.

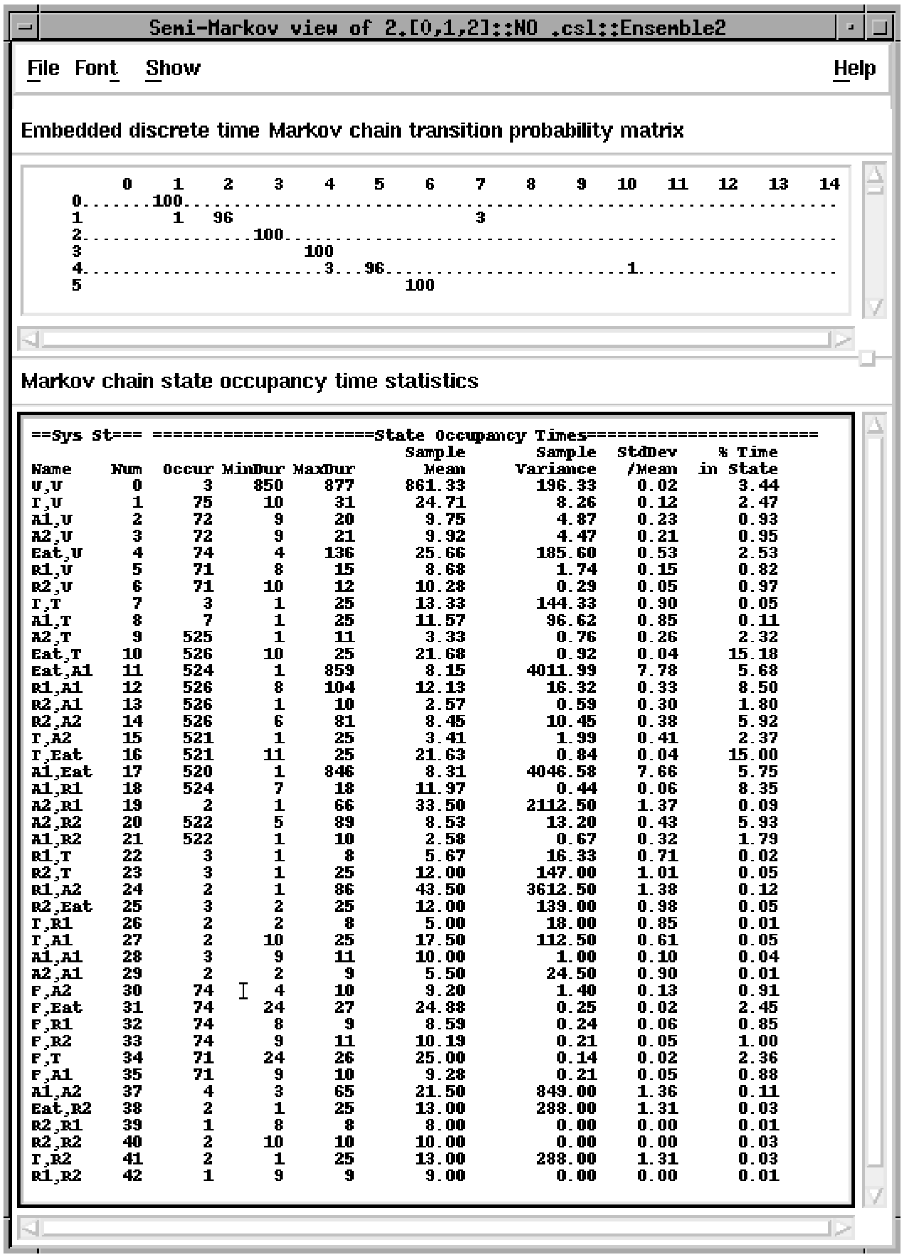
3.5 Modeling Ensembles

The information in §1.6 on creating a semi-Markov process model from one TSSS also applies to creating a model from an ensemble of two or more TSSS's. The following differences in how the model is obtained from the ensemble exist.

1. When one selects "Views → Model → Continuous time semi-Markov", as with any view, the dialog box in Fig. 3.4 on page 61 appears. The user can use this box to see one of several models:
   (a) Create a semi-Markov model of *just one* member (i.e., "Separate" versus "Combined") by selecting only one name from the top list box (e.g., the *exact same* model as shown in Fig. 1.19 on page 45). The radio button setting does not matter.
   (b) Create a single semi-Markov model of the *entire ensemble* by selecting all names in the top list box and the "Combined" radio button.
   (c) Create a single semi-Markov model of the *some but not all members* in the ensemble by selecting all names in the top list box and the "Combined" radio button.
   (d) View on the screen simultaneously a semi-Markov model of each member separately by selecting some or all members in the top list box and the "Separate" radio button.
2. Let Σ denote the set of members selected in the box of Fig. 3.4. The system and model state spaces (respectively, $\mathcal{S}$ and $\mathcal{M}$) of the semi-Markov model chain are the set of system and model states that occur in all TSSS's in the ensemble members in set Σ.
3. In the top pane, element i, j of the transition matrix is the percentage of transitions in the ensemble members in set Σ matching (m_i, m_j).
4. In the bottom pane, the statistics are calculated from all model states in the model and occupancy time state sequences in all ensemble members in set Σ.

We will now generate alternative 1b above. Change focus to the root window and select "Views: Model → Continuous time semi-Markov". Select all three members and "Combined" in the list box of Fig. 3.4. A model view window similar to Fig. 3.10 should appear. Move the square box on the right end of the horizontal line between the top and bottom panes up so to enlarge the bottom pane (and shrink the top pane) to expose all model states. Compare the ensemble model (Fig. 3.10) to the model of 2_a_10_200_1.cbn alone (Fig. 1.19 on page 45). For example, the entries in the "Occur" column in Fig. 3.10 are much larger numbers (e.g., 526 versus 174 for state "Eat,T"), because samples from three TSSS's are represented in Fig. 3.10.

Try some of the other alternatives listed above with CHITRA93, and compare the results. When you are done, exit CHITRA93.



Semi-Markov view of 2.[0,1,2]::NO .csl::Ensemble2

File Font Show Help

Embedded discrete time Markov chain transition probability matrix

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
0		100													
1		1	96					3							
2				100											
3					100										
4					3	96					1				
5							100								

Markov chain state occupancy time statistics

==Sys St=== ======================State Occupancy Times=======================

Name	Num	Occur	MinDur	MaxDur	Sample Mean	Sample Variance	StdDev /Mean	% Time in State
U,U	0	3	850	877	861.33	196.33	0.02	3.44
I,U	1	75	10	31	24.71	8.26	0.12	2.47
A1,U	2	72	9	20	9.75	4.87	0.23	0.93
A2,U	3	72	9	21	9.92	4.47	0.21	0.95
Eat,U	4	74	4	136	25.66	185.60	0.53	2.53
R1,U	5	71	8	15	8.68	1.74	0.15	0.82
R2,U	6	71	10	12	10.28	0.29	0.05	0.97
I,T	7	3	1	25	13.33	144.33	0.90	0.05
A1,T	8	7	1	25	11.57	96.62	0.85	0.11
A2,T	9	525	1	11	3.33	0.76	0.26	2.32
Eat,T	10	526	10	25	21.68	0.92	0.04	15.18
Eat,A1	11	524	1	859	8.15	4011.99	7.78	5.68
R1,A1	12	526	8	104	12.13	16.32	0.33	8.50
R2,A1	13	526	1	10	2.57	0.59	0.30	1.80
R2,A2	14	526	6	81	8.45	10.45	0.38	5.92
I,A2	15	521	1	25	3.41	1.99	0.41	2.37
I,Eat	16	521	11	25	21.63	0.84	0.04	15.00
A1,Eat	17	520	1	846	8.31	4046.58	7.66	5.75
A1,R1	18	524	7	18	11.97	0.44	0.06	8.35
A2,R1	19	2	1	66	33.50	2112.50	1.37	0.09
A2,R2	20	522	5	89	8.53	13.20	0.43	5.93
A1,R2	21	522	1	10	2.58	0.67	0.32	1.79
R1,T	22	3	1	8	5.67	16.33	0.71	0.02
R2,T	23	3	1	25	12.00	147.00	1.01	0.05
R1,A2	24	2	1	86	43.50	3612.50	1.38	0.12
R2,Eat	25	3	2	25	12.00	139.00	0.98	0.05
I,R1	26	2	2	8	5.00	18.00	0.85	0.01
I,A1	27	2	10	25	17.50	112.50	0.61	0.05
A1,A1	28	3	9	11	10.00	1.00	0.10	0.04
A2,A1	29	2	2	9	5.50	24.50	0.90	0.01
F,A2	30	74	4	10	9.20	1.40	0.13	0.91
F,Eat	31	74	24	27	24.88	0.25	0.02	2.45
F,R1	32	74	8	9	8.59	0.24	0.06	0.85
F,R2	33	74	9	11	10.19	0.21	0.05	1.00
F,T	34	71	24	26	25.00	0.14	0.02	2.36
F,A1	35	71	9	10	9.28	0.21	0.05	0.88
A1,A2	37	4	3	65	21.50	849.00	1.36	0.11
Eat,R2	38	2	1	25	13.00	288.00	1.31	0.03
R2,R1	39	1	8	8	8.00	0.00	0.00	0.01
R2,R2	40	2	10	10	10.00	0.00	0.00	0.03
I,R2	41	2	1	25	13.00	288.00	1.31	0.03
R1,R2	42	1	9	9	9.00	0.00	0.00	0.01

Figure 3.10: Semi-Markov model constructed from all three ensemble members.

Chapter 4

Lesson 4: Transforming an Ensemble

A *transform* maps each TSSS in an ensemble to a simpler TSSS, either containing fewer model states or representing a shorter duration. Recall from §1.4.1.1 that an ensemble is treated as a set of realizations of a stochastic process $X(t)$. Transforms either reduce the range of X or reduce the interval over which t is defined. CHITRA93 uses the classical method of controlling state space explosion: a hierarchical model, in which all levels but the bottom contain aggregate model states.

CHITRA93 has three classes of transforms:

Clipping: Reduces the interval over which t is defined by extracting a portion of a TSSS.

Aggregation: Reduces the range of X by replacing model states by aggregate model states and hence defining a hierarchical model.

Projection: Reduces the range of X by applying user defined system state transformation rules.

Transforms are illustrated on the Dining Philosophers program TSSS's. We will reduce the 43 state ensemble model from Chapter 3 (Fig. 3.10) into a hierarchical model with only three states in its top level.

4.1 Ensemble Versions

When a user opens an ensemble, CHITRA93 opens an *instance* of the TSSS's in the ensemble. Let $\mathcal{E}$ denote an ensemble instance. Each ensemble instance $\mathcal{E}$ will have multiple *versions*, denoted $\mathcal{E}_0, \mathcal{E}_1, \ldots, \mathcal{E}, \ldots$. Version $\mathcal{E}_0$ is created when the ensemble is first opened; successive versions are created each time an ensemble is transformed. Version $\mathcal{E}_0$ is identical to the original disk copy; other versions modify the loaded instance, not the original disk copy. A user can open multiple instances of the same ensemble, permitting each instance to be modified independently. This is useful for comparing the model obtained by different sequences of transforms.

Henceforth, system to model state mapping $X_{\mathcal{E}_v}$ is subscripted by $\mathcal{E}$ and model state space $\mathcal{M}(\mathcal{E}_v)$ is a function of version v of ensemble $\mathcal{E}$ because CHITRA93 modifies the mapping whenever a new ensemble version is created through transformation (§4). Also, the length of version v of a TSSS is denoted L_v. Subscripts are dropped when the ensemble and version are clear from context. The algorithm to map system to model states described in §3.1.1 thus defines $X(\mathcal{E}_0)$ and $\mathcal{M}(\mathcal{E}_0)$.

4.2 2D Views and the Transform Menu

Two of the three transform classes – clipping and aggregation – are invoked by the *tranforms* menu in the 2D views (Table 4.1). The third class – projection – is invoked by a family of separate commands used outside CHITRA93 (see §4.6).

4.3 Clipping Transform

The clipping transform creates a new ensemble version by removing the initial and/or final portion of each member in an ensemble. We will use clipping to model only the portion of the Dining Philosopher program TSSS's in which *both* philosophers are executing.

Start CHITRA93. Open an ensemble consisting of the three TSSS's used earlier: 2_a_10_200_1.cbn, 2_a_10_200_2.cbn, and 2_a_10_200_3.cbn. (Follow the instructions in §3.1.) CHITRA93 will call the ensemble "Ensemble0" in the root window. Open a 2D time view of all three members in separate 2D windows. (Follow the instructions in §3.2, but select all three member names in the list box, rather than just 2_a_10_200_1.cbn and 2_a_10_200_3.cbn.)

Each graph (two of which are shown in Fig. 3.5 on page 3.5) consists of three portions. In the first, just one philosopher 0 is executing (hence the system state names are (x,U), where x is one of T, A1, A2, Eat, R1, or R2). The undefined ("U") state for philosopher 1 indicates that philosopher 1 has not yet made any state transitions and hence is not yet executing. In the second portion, both philosophers are running (thus the system states contain no "U" or "F"). In the third, philosopher 1 alone is running (thus the system states are (U,x)).

Menu item			*Description*
Clip			Create new ensemble version by removing the initial and final portions of all ensemble members
Aggregate			Create new ensemble version by combining two or more model states in all ensemble members into one or more new model states (called aggregate states)
Aggregate	→ By Pattern		Replace model state subsequences matching a pattern by one aggregate state
Aggregate	→ By Pattern	→ Select	Select a model state subsequence to use as a pattern; pop up a box reporting on frequence of pattern occurrence in each ensemble member; plot in a different color in 2D view all occurrences of the pattern
Aggregate	→ By Pattern	→ Deselect	Remove highlighting of a pattern produced by "Aggregate → By Pattern → Select".
Aggregate	→ By Pattern	→ Apply	Replace all occurrences of a pattern by one new model state; update all views of ensemble to reflect new state
Aggregate	→ By Filter		Replace any model state with number of occurrences or occupancy time below a user-specified threshold by one or more aggregate states
Undo last transform			Given that the current ensemble version is v, make $\mathcal{E}_{v-1}$ the current ensemble.

Table 4.1: Menu choices under "Transform" in 2D time and event windows.

4.3.1 Selecting a Region to Clip by a Bounding Box

To restrict our analysis to the TSSS portions in which both philosophers are executing, we first note that both philosophers are running by about time 3146 in 2_a_10_200_1.cbn and 3121 in 2_a_10_200_2.cbn and 2_a_10_200_3.cbn. Also philosopher 0 stops running a little after time 22022 in 2_a_10_200_1.cbn, and time 21850 in 2_a_10_200_2.cbn and 2_a_10_200_3.cbn. Thus we wish to retain the TSSS portion in all three members that starts after time 3146 and stops by about time 21850. To do this, change focus to the 2D time window displaying member 0 (denoted "2D Time view of 0.0…"). Chose menu item "Transform: Clip". CHITRA93 will now draw a bounding box, similar to ones used in drawing tools, around the portion of the graph that you wish to *retain*. Move the mouse pointer below the x axis to the right of x-axis label 3146. Press and hold the left mouse button. Move the mouse pointer diagonally until it is somewhat to the left of the x-axis label 22022 (so that it is to the left of time 21850) and above the curve on the graph. You should now see a box in dashed lines that encloses part of the graph, similar to Fig. 4.1 (however the box is in solid lines in Fig. 4.1).

The vertical sides of the box alone are significant. These determine the two times at which to clip. The horizontal sides of the box actually are ignored; therefore it is not crucial that you enclose all states desired within the horizontal sides.

4.3.2 Definition of Clipping Transform

The clipping transform maps ensemble version $\mathcal{E}_v$ to version $\mathcal{E}_{v+1}$ as follows. A 2D view visualizes one model state sequence $\hat{M}_0$, $\hat{M}_1$, …, $\hat{M}_{L_v}$ in ensemble version $\mathcal{E}_v$. Let n_L ($n_L \geq 0$) and n_H ($n_H \leq L_v$) denote integers such that the largest model state subsequence that lies on or in the bounded box is $\hat{M}_{n_L}$, $\hat{M}_{n_L+1}$, …, $\hat{M}_{n_H-1}$, $\hat{M}_{n_H}$. (Graphically, model states $\hat{M}_{n_L}$, $\hat{M}_{n_L+1}$, …, $\hat{M}_{n_H-1}$, $\hat{M}_{n_H}$ correspond to the set of crosses in a 2D view that lie on or inside the box.) For each model state sequence $O_0, O_1, \ldots, O_{L_v}$ and occupancy time sequence M_0, M_1, …, M_{L_v} in $\mathcal{E}_v$, ensemble $\mathcal{E}_{v+1}$ will contain occupancy sequence O_{n_L}, O_{n_L+1}, …, O_{n_H-1}, O_{n_H} and model sequence M_{n_L}, M_{n_L+1}, …, M_{n_H-1}, M_{n_H}.

4.3.3 How Views Change After Clipping

Now release the mouse button. The 2D view that is the key window should change. The other two windows may change; if not, click in their banners or select "File → Update this view" in each window. The three windows should now appear similar to those shown in Fig. 4.2. Note that the graph segments in which one philosopher only is running have now been removed.

Change focus to the root window and select "Views: Model → Continuous

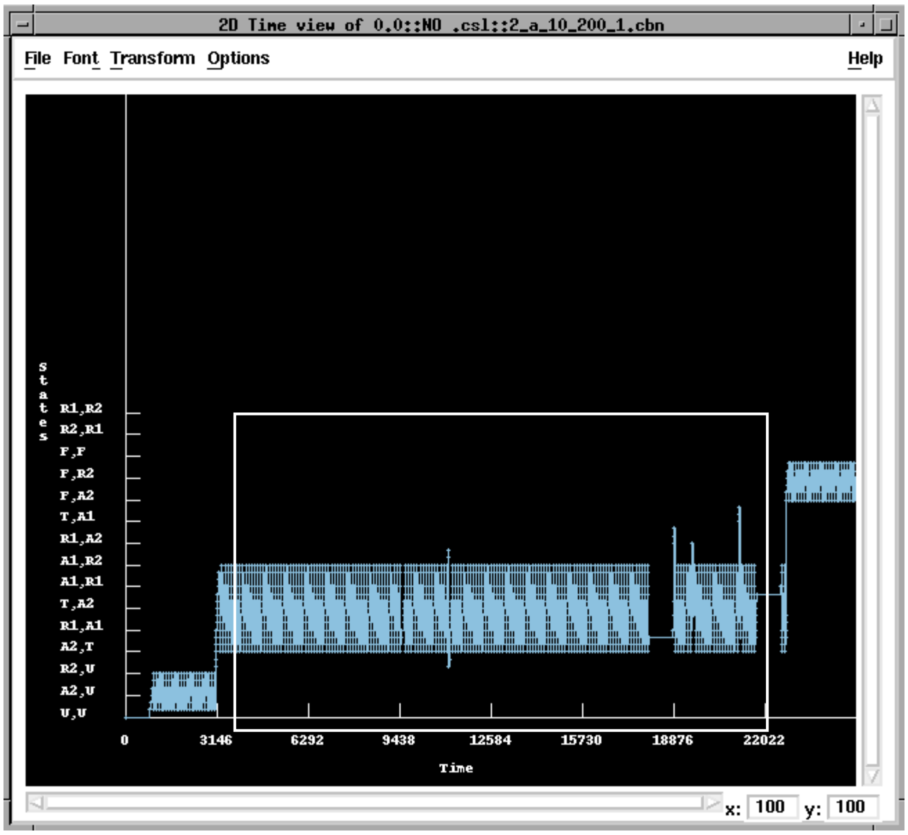


Figure 4.1: Illustration of bounding box drawn in response to "Transform: Clip" menu item to select time period when both philosophers are executing.

time semi-Markov". Select all three members and "Combined" in the list box of Fig. 3.4 on page 61. When the model view window appears (Fig. 4.3), move the square box on the right end of the horizontal line between the top and bottom panes up so to enlarge the bottom pane (and shrink the top pane) to expose all model states. Compared to our last model view (Fig. 3.10 on page 70), there are now only 26 states in the model, which is fewer than the model before clipping (e.g., 43 in Fig. 3.10). The reduction from 43 to 26 occurred because all system states in which a philosopher was in state "U" or "F" have been deleted.

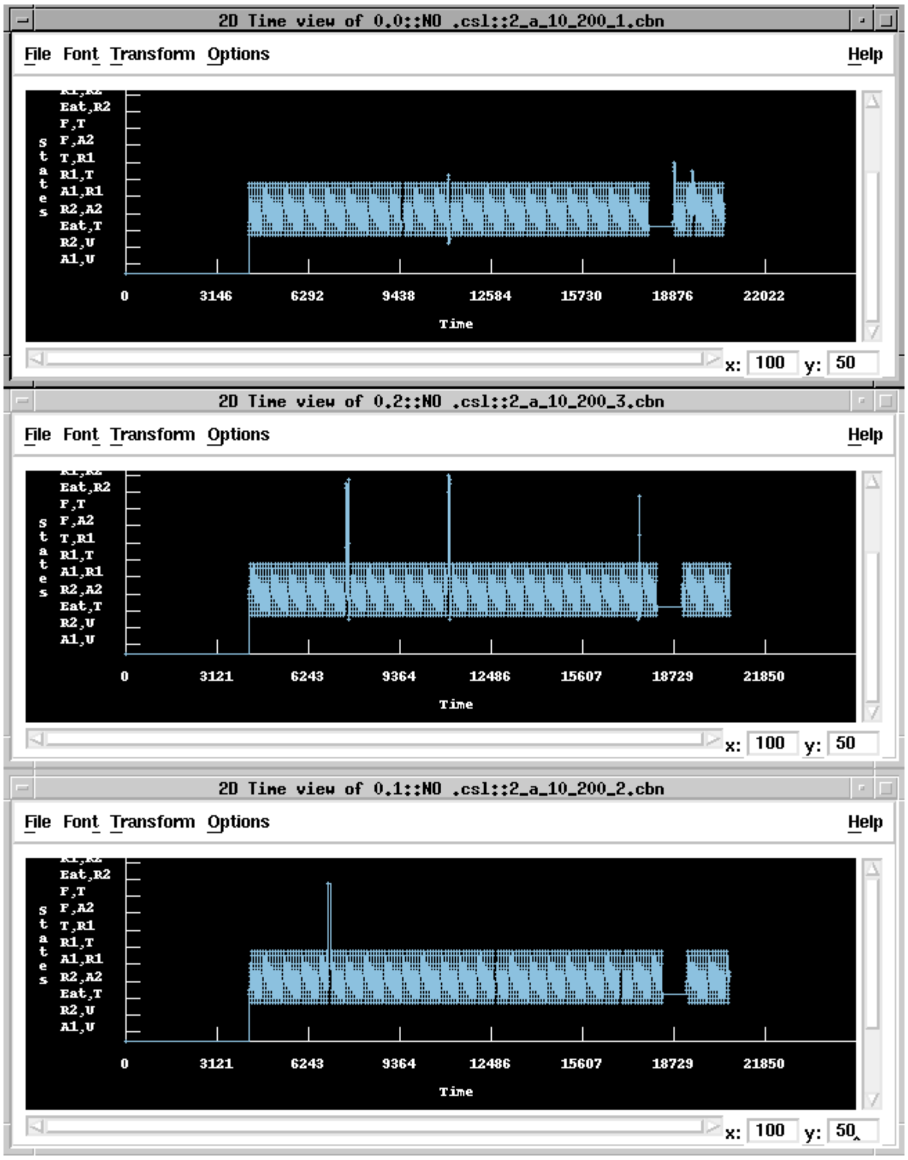
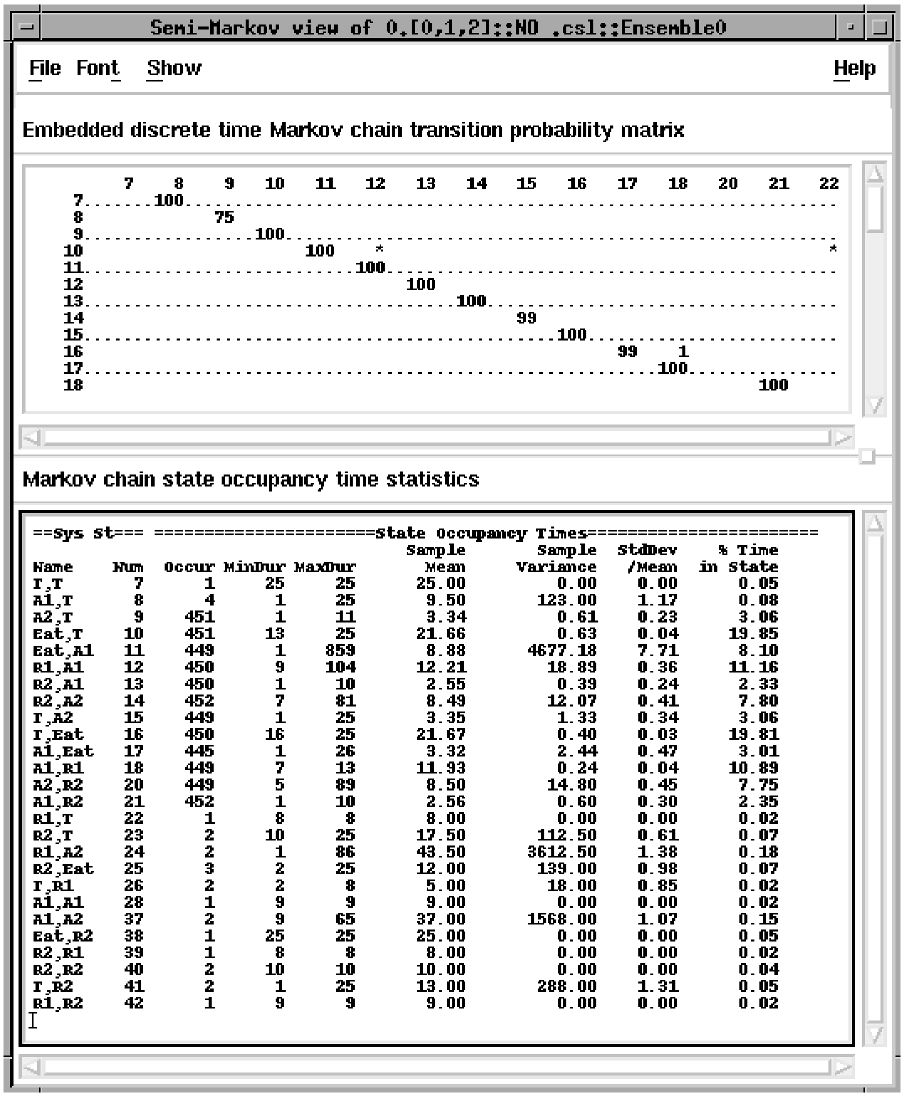


Figure 4.2: Result of applying clip transform



Embedded discrete time Markov chain transition probability matrix

	7	8	9	10	11	12	13	14	15	16	17	18	20	21	22
7		100													
8			75												
9				100											
10					100	*									*
11						100									
12							100								
13								100							
14									99						
15										100					
16											99	1			
17												100			
18														100	

Markov chain state occupancy time statistics

==Sys St=== =====================State Occupancy Times======================

Name	Num	Occur	MinDur	MaxDur	Sample Mean	Sample Variance	StdDev /Mean	% Time in State
I,T	7	1	25	25	25.00	0.00	0.00	0.05
A1,T	8	4	1	25	9.50	123.00	1.17	0.08
A2,T	9	451	1	11	3.34	0.61	0.23	3.06
Eat,T	10	451	13	25	21.66	0.63	0.04	19.85
Eat,A1	11	449	1	859	8.88	4677.18	7.71	8.10
R1,A1	12	450	9	104	12.21	18.89	0.36	11.16
R2,A1	13	450	1	10	2.55	0.39	0.24	2.33
R2,A2	14	452	7	81	8.49	12.07	0.41	7.80
I,A2	15	449	1	25	3.35	1.33	0.34	3.06
I,Eat	16	450	16	25	21.67	0.40	0.03	19.81
A1,Eat	17	445	1	26	3.32	2.44	0.47	3.01
A1,R1	18	449	7	13	11.93	0.24	0.04	10.89
A2,R2	20	449	5	89	8.50	14.80	0.45	7.75
A1,R2	21	452	1	10	2.56	0.60	0.30	2.35
R1,T	22	1	8	8	8.00	0.00	0.00	0.02
R2,T	23	2	10	25	17.50	112.50	0.61	0.07
R1,A2	24	2	1	86	43.50	3612.50	1.38	0.18
R2,Eat	25	3	2	25	12.00	139.00	0.98	0.07
I,R1	26	2	2	8	5.00	18.00	0.85	0.02
A1,A1	28	1	9	9	9.00	0.00	0.00	0.02
A1,A2	37	2	9	65	37.00	1568.00	1.07	0.15
Eat,R2	38	1	25	25	25.00	0.00	0.00	0.05
R2,R1	39	1	8	8	8.00	0.00	0.00	0.02
R2,R2	40	2	10	10	10.00	0.00	0.00	0.04
I,R2	41	2	1	25	13.00	288.00	1.31	0.05
R1,R2	42	1	9	9	9.00	0.00	0.00	0.02

Figure 4.3: Semi-Markov model constructed from all three ensemble members.

4.4 Aggregation Transform

Aggregation transforms replace certain model states by new model states, called *aggregate* states. There are two ways to aggregate in CHITRA93: by pattern and by filter. The two will be explained by describing how each transform maps ensemble version $\mathcal{E}_v$ to version $\mathcal{E}_{v+1}$. Let m denote the maximum value in $\mathcal{M}(\mathcal{E}_v)$.

4.4.1 Aggregation by Pattern

We have thus far reduced the model state space from 43 to 26 states. We will now reduce the size much further by introducing aggregate states. This will make our model hierarchical, because the semi-Markov view will show the top level model in terms aggregate states; one can then query the definition of an aggregate state (§4.4.1.4 and §4.4.2.4) to find its definition in terms of other states.

Because a user will not normally visualize all TSSS's in an ensemble, feedback on whether the pattern arises in non-visualized TSSS's is required. Thus CHITRA93 reports the number of times the pattern was found in each TSSS in the ensemble, and the fraction of each TSSS's duration that was replaced by the pattern.

4.4.1.1 Definition of Aggregation by Pattern

Let μ be a sequence (i.e., a pattern) of model states. The aggregation transform maps μ in version $\mathcal{E}_v$ to version $\mathcal{E}_{v+1}$ as follows. Initially set $\mathcal{E}_{v+1}$ to $\mathcal{E}_v$. For each TSSS in $\mathcal{E}_{v+1}$, do the following. Let M and O denote the model and occupancy time sequences corresponding to the TSSS. The pattern μ may occur multiple times in M; repeat the following for each instance x of μ in M. Let y denote the occupancy time subsequence in O corresponding to M. Replace x by $m+1$ and y by the sum of the elements in y. Note that after the transform, if pattern μ was found then the model state space size grew by one: $\mathcal{M}(\mathcal{E}_{v+1}) = \mathcal{M}(\mathcal{E}_v) \cup \{m+1\}$.

4.4.1.2 Selecting the Pattern by a Bounding Box

From the time view zoomed to 10% in the x-axis shown earlier (Fig. 1.15 on page 1.15) as well as the FFT time view (Fig. 1.17 on page 1.17) of ensemble member 0, it appears that there are patterns of state transitions present in at least this member. Let's replace any patterns we find by an aggregate state. Change focus to the 2D time view of member 0. Zoom to 3%. You should see something similar to Fig. 4.4 (but without the bounding box). This figure appears to contain around a half dozen repetitions of the same state transition pattern that looks like a staircase; one instance of the pattern is enclosed in the box in Fig. 4.4.

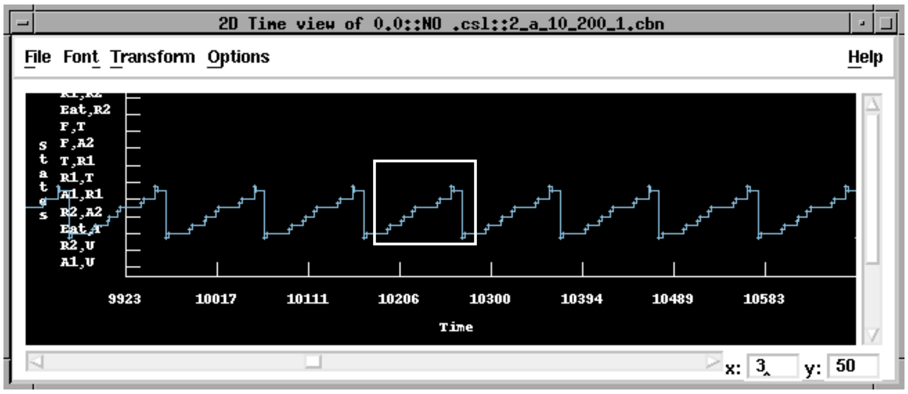
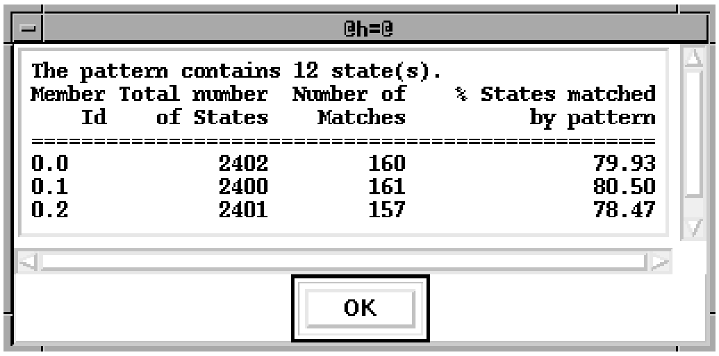


Figure 4.4: Result of zooming top window in Fig. 4.2 (2D time view of member 0) to 3%.



The pattern contains 12 state(s).

Member Id	Total number of States	Number of Matches	% States matched by pattern
0.0	2402	160	79.93
0.1	2400	161	80.50
0.2	2401	157	78.47

Figure 4.5: Window that appears after "2D: Transforms → Aggregate → By Pattern → Apply" indicating frequency of occurrence of a pattern in each ensemble member.

To see if this is indeed a pattern, select menu item "Transforms → Aggregate → By Pattern → Select". Draw a bounded box as described in §4.3 around one instance of the pattern, as illustrated in Fig. 4.4. Now select "Transform → Aggregate → By Pattern → Apply". You will see a window open that reports the frequency of occurrence of the pattern in each ensemble member (Fig. 4.5). The pattern occurs between 157 and 161 times in each ensemble member. The pattern accounts for between 78.47 and 80.50 percent of the states in each ensemble member. Thus the pattern will simplify most of all three TSSS's. Click "OK" in the box to dismiss it.

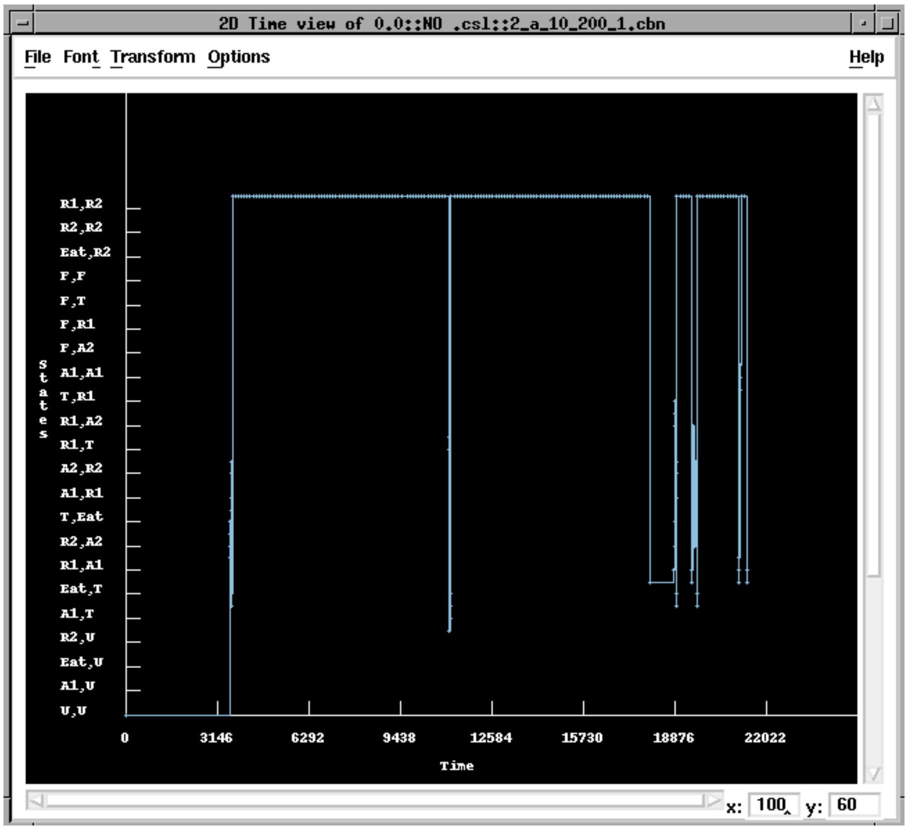


Figure 4.6: Result of applying pattern match to Fig. 4.4 and resetting zoom factors.

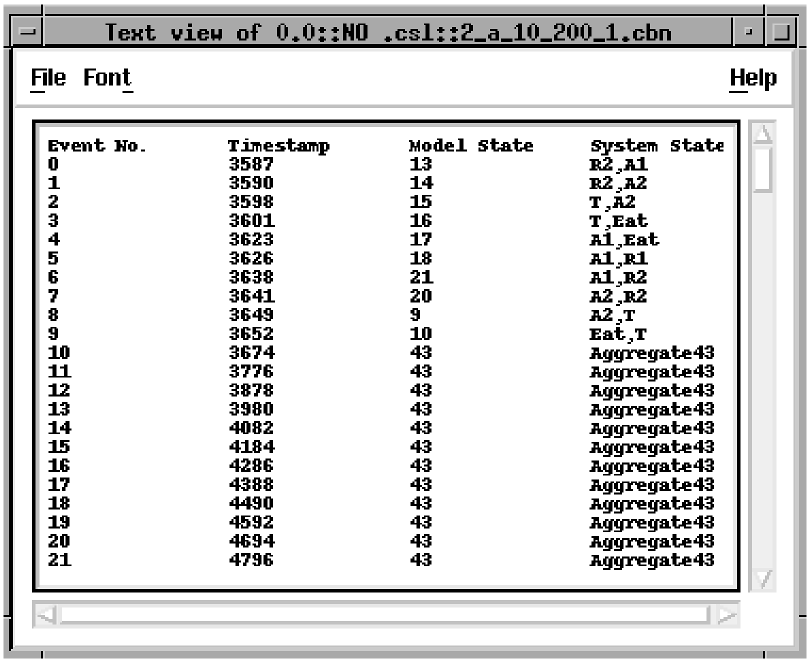
4.4.1.3 How the Views Change After Pattern Aggregation

2D View

Change focus to the 2D time view of 2_a_10_200_1.cbn, and change the x and y zoom to 100 and 60, respectively. The result should be similar to Fig. 4.6. Before the aggregation transform was applied, the largest model state was $m = 42$. Most of the graph is now replaced by a new aggregate state (43) (visible in Figs. 3.10 and 4.3). Most of the graph is now a horizontal line with y coordinate 43. (The y-axis labels do not label each pont on the y-axis; therefore the new state is not labeled.)

Text View

Let's see how the text view is affected by the clipping and aggregation by



Event No.	Timestamp	Model State	System State
0	3587	13	R2,A1
1	3590	14	R2,A2
2	3598	15	T,A2
3	3601	16	T,Eat
4	3623	17	A1,Eat
5	3626	18	A1,R1
6	3638	21	A1,R2
7	3641	20	A2,R2
8	3649	9	A2,T
9	3652	10	Eat,T
10	3674	43	Aggregate43
11	3776	43	Aggregate43
12	3878	43	Aggregate43
13	3980	43	Aggregate43
14	4082	43	Aggregate43
15	4184	43	Aggregate43
16	4286	43	Aggregate43
17	4388	43	Aggregate43
18	4490	43	Aggregate43
19	4592	43	Aggregate43
20	4694	43	Aggregate43
21	4796	43	Aggregate43

Figure 4.7: Text view after aggregation by pattern transform is applied.

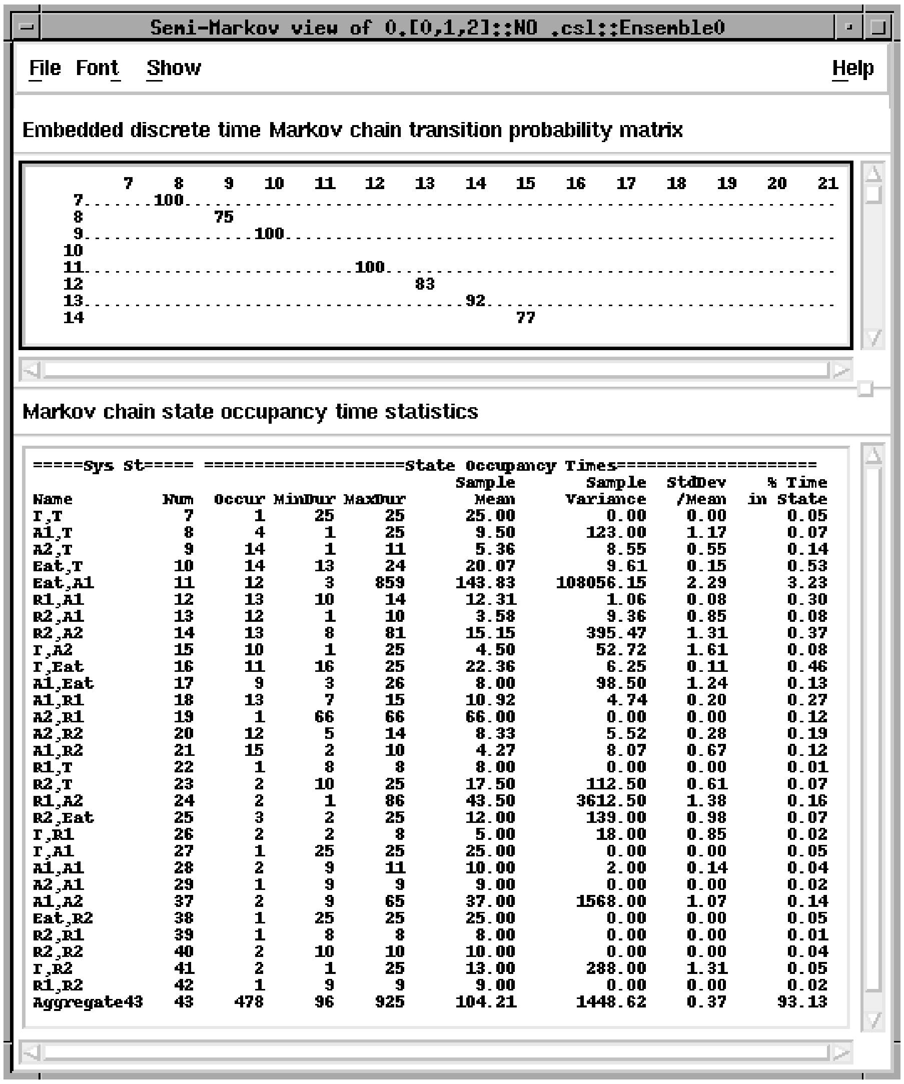
pattern transforms. Next change focus to the root window and select "Views → Text". A window similar to Fig. 4.7 should appear. Compared to Fig. 1.13 on page 36, model state 43 is now added, and CHITRA93 has generated the name "Aggregate43" to denote the system state corresponding to model state 43.

Model View

Now let's see how the semi-Markov model is affected by the aggregation by pattern. A window similar to Fig. 4.8 should appear. Move the square box at the right end of the horizontal line separating the panes up until the last state ("Aggregate43") is exposed. Compared to Fig. 3.10 on page 70, the model state 43 is added at the bottom of the lower pane. Also notice that only model state 43 occurs hundreds of times, and all other model states occur 1 to 15 times. Thus most of the TSSS is represented by aggregate state 43.

4.4.1.4 Querying the Definition of an Pattern Aggregate State

To query the definition of an aggregate state, such as model state 43, change focus to the root window. Select "Options → Define aggregate state → Pattern". The dialog box shown in Fig. 4.9 pops up. Enter the ensemble's id ("0" from



Embedded discrete time Markov chain transition probability matrix

	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21
7		100													
8			75												
9				100											
10															
11						100									
12							83								
13								92							
14									77						

Markov chain state occupancy time statistics

=====Sys St=====		===================State Occupancy Times===================						
					Sample	Sample	StdDev	% Time
Name	Num	Occur	MinDur	MaxDur	Mean	Variance	/Mean	in State
T,T	7	1	25	25	25.00	0.00	0.00	0.05
A1,T	8	4	1	25	9.50	123.00	1.17	0.07
A2,T	9	14	1	11	5.36	8.55	0.55	0.14
Eat,T	10	14	13	24	20.07	9.61	0.15	0.53
Eat,A1	11	12	3	859	143.83	108056.15	2.29	3.23
R1,A1	12	13	10	14	12.31	1.06	0.08	0.30
R2,A1	13	12	1	10	3.58	9.36	0.85	0.08
R2,A2	14	13	8	81	15.15	395.47	1.31	0.37
T,A2	15	10	1	25	4.50	52.72	1.61	0.08
T,Eat	16	11	16	25	22.36	6.25	0.11	0.46
A1,Eat	17	9	3	26	8.00	98.50	1.24	0.13
A1,R1	18	13	7	15	10.92	4.74	0.20	0.27
A2,R1	19	1	66	66	66.00	0.00	0.00	0.12
A2,R2	20	12	5	14	8.33	5.52	0.28	0.19
A1,R2	21	15	2	10	4.27	8.07	0.67	0.12
R1,T	22	1	8	8	8.00	0.00	0.00	0.01
R2,T	23	2	10	25	17.50	112.50	0.61	0.07
R1,A2	24	2	1	86	43.50	3612.50	1.38	0.16
R2,Eat	25	3	2	25	12.00	139.00	0.98	0.07
T,R1	26	2	2	8	5.00	18.00	0.85	0.02
T,A1	27	1	25	25	25.00	0.00	0.00	0.05
A1,A1	28	2	9	11	10.00	2.00	0.14	0.04
A2,A1	29	1	9	9	9.00	0.00	0.00	0.02
A1,A2	37	2	9	65	37.00	1568.00	1.07	0.14
Eat,R2	38	1	25	25	25.00	0.00	0.00	0.05
R2,R1	39	1	8	8	8.00	0.00	0.00	0.01
R2,R2	40	2	10	10	10.00	0.00	0.00	0.04
T,R2	41	2	1	25	13.00	288.00	1.31	0.05
R1,R2	42	1	9	9	9.00	0.00	0.00	0.02
Aggregate43	43	478	96	925	104.21	1448.62	0.37	93.13

Figure 4.8: Model view of ensemble after aggregation by pattern transform is applied. Compare to Fig. 3.10.

the list box in the root window) in the top field, "43" in the second field, and hit apply. The window shown in Fig. 4.10 should appear. It shows that model state 43 consists of a sequence of 12 system states: Eat,A1; R1,A1; ..., Eat,T.

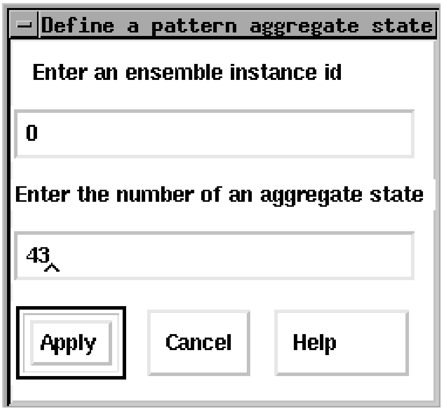


Figure 4.9: Dialog box invoked by "Options → Define aggregate state → Pattern".

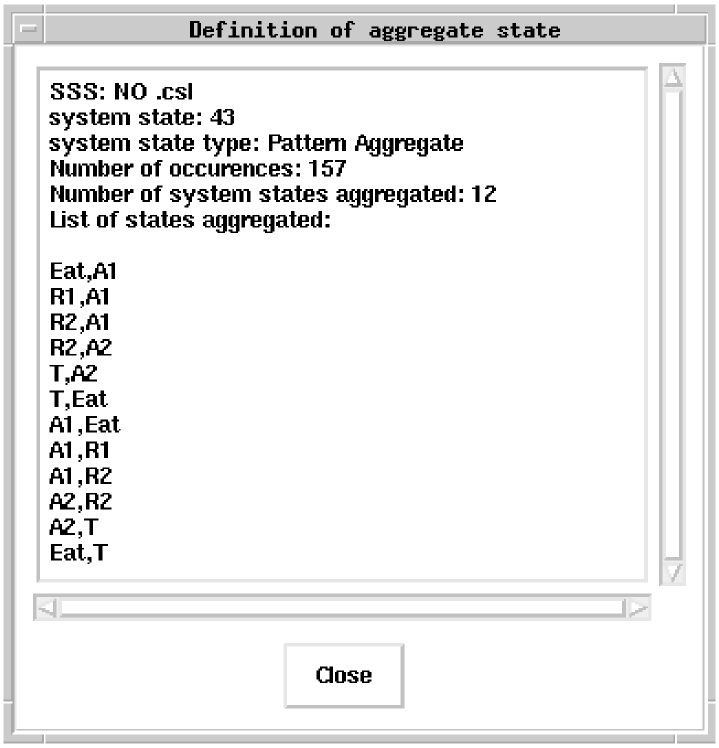


Figure 4.10: Definition of a pattern aggregate state resulting from "Options → Define aggregate state → Pattern"

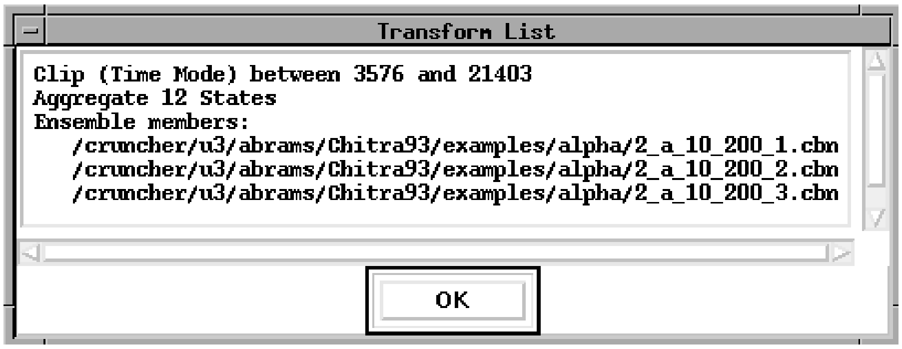


Figure 4.11: List of transforms applied so far, invoked from "Root Window: Options → List transforms for ensemble".

4.4.1.5 Generating a Transform List

Suppose we forget what transforms we have applied to an ensemble so far. The options menu in the root window (Table 4.2) provides auxiliary information about transforms and aggregate model states. Change focus to the root window and select "Options → List transforms for ensemble". A transform list similar to Fig. 4.11 should appear. The top of the box says that we first applied a clip transform from a 2D time window using 3576 and 21403 as bounding box sides (Fig. 4.4), followed by a pattern aggregation of 12 states.

Menu item	*Description*
List transforms for ensemble	Displays a list of the sequence of transforms that have been applied to the selected ensemble so far.
Define aggregate state	Defines the states comprising an aggregate state along with a few statistics about the state.
Define aggregate state→ Filter	Displays the acyclic chain corresponding the to selected aggregate filter state (see section 4.4.2.4).
Define aggregate state→ Pattern	Displays the pattern of states that the selected aggregate pattern state represents (see section 4.4.1.4).

Table 4.2: Menu choices under "Options" in the root window.

4.4.2 Aggregation by Filter

Filter transforms replace states either that occur infrequently or that account for a small fraction of the total by one or more filter aggregate states.

4.4.2.1 Definition of Aggregation by Filter

The filter transform replaces each contiguous subsequence of states that meets a *filtering criterion* in each model state sequence in an ensemble instance $\mathcal{E}_v$ by a model state whose value exceeds all model states in $\mathcal{M}(\mathcal{E}_v)$. The filter criterion is that either the occupancy time or the number of instances in $\mathcal{E}_v$ of a state is less than a certain value.

Consider a TSSS in the ensemble. Define the predecessor of model state M_i to be M_{i-1} for $i > 1$; model state M_0 has no predecessor. Define the successor of model state M_i to be M_{i+1} for $i < L_v - 1$; model state M_{L_v} has no successor.

Let $f(\mathcal{E}_v)$ denote the set of model states in $\mathcal{E}_v$ that meet the filter criterion. A *run* is a subsequence of a model state sequence in a member of $\mathcal{E}_v$ such that (1) all states in the subsequence are in $f(\mathcal{E}_v)$, (2) the predecessor state of the first state in the subsequence is either undefined or is not in $f(\mathcal{E}_v)$; and (3) the successor state of the last state in the subsequence is either undefined or is not in $f(\mathcal{E}_v)$. The filter transform maps $\mathcal{E}_v$ to $\mathcal{E}_{v+1}$ as follows. Initially set $\mathcal{E}_{v+1}$ to $\mathcal{E}_v$. Let the number of distinct runs in $\mathcal{E}_{v+1}$ be n_r. Replace all instances of runs in $\mathcal{E}_{v+1}$ by model states $\{m+1, m+2, \ldots, m+n_r\}$ such that all runs sharing the same predecessor and successor state are replaced by the same model state. (The model state sequence of all TSSS's in the ensemble are viewed as having a common state prepended and and a common state appended, so that every state in every run has a predecessor and successor.) For each subsequence in an occupancy time sequence in $\mathcal{E}_{v+1}$ that corresponds to a run, replace the subsequence by the sum of its elements.

4.4.2.2 Selecting the Filter by a Histogram

The rightmost column of the semi-Markov view of one member (e.g., Fig. 1.19 on page 45) is labeled "% Time in State". The value listed for each state is the sum of the occupancy time of each occurrence of the state in the TSSS for the member in the current ensemble version divided by the time interval that the TSSS represents in the current version. For example, state Eat,T in Fig. 1.19 represents 14.97% of the duration of the TSSS for member 0 in the initial ensemble version. We define the *mean state occupancy time* of a state to be the average of the value under "% Time in State" in a model view of each ensemble member separately.

Change focus two the 2D time view of ensemble member 0. Select menu item "Transforms → Aggregate → By filter". A window containing a histogram will appear (Fig. 4.12). The histogram shows the distribution of mean state occupancy times of all states. The x-axis is normalized to a scale of 0 to 100%.

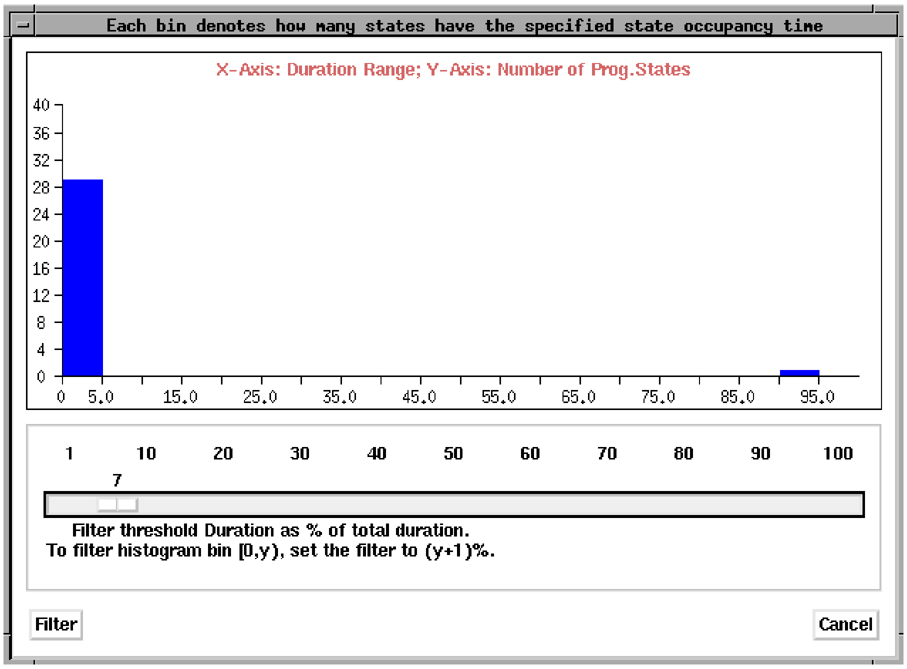


Figure 4.12: Histogram used to select filter transform.

Thus the bin of height 29 in Fig. 4.12 indicates that 29 states had a mean state occupancy time (averaged over *all* ensemble members) in the interval [0,5)%. The number "29" agrees with the fact that 29 of the 30 states listed in the bottom pane of the last semi-Markov view (Fig. 4.8) have a value under 5%. Meanwhile one state (model state 43) has a mean occupancy time of 93.13%; thus there is a bar that is one unit high in the bin [90,95)% in the histogram (Fig. 4.12).

To select the filter transform so that the 29 states with small state occupancy time are replaced by aggregate states, one would move the slider to a value between the two bars. To be precise, to filter all states with an occupancy time less than $p\%$, one would move the slider to a value of $p + 1\%$. Set the slider to 7% and click on the "Filter" button. CHITRA93 now applies the filter, with $f(\mathcal{E}_v)$ containing the set of states that have mean occupancy time less than 7% (i.e., all states except Aggregate43).

(If "Transforms → Aggregate → By filter" is selected from a 2D *event* window, rather than a *time* window, then the histogram displays the distribution of the mean percentage of *occurrences* of all states, rather than the percentage of times (not illustrated). The mean percentage of occurrence of a state equals the value in the "Occur" column in model view divided by the sum of the values

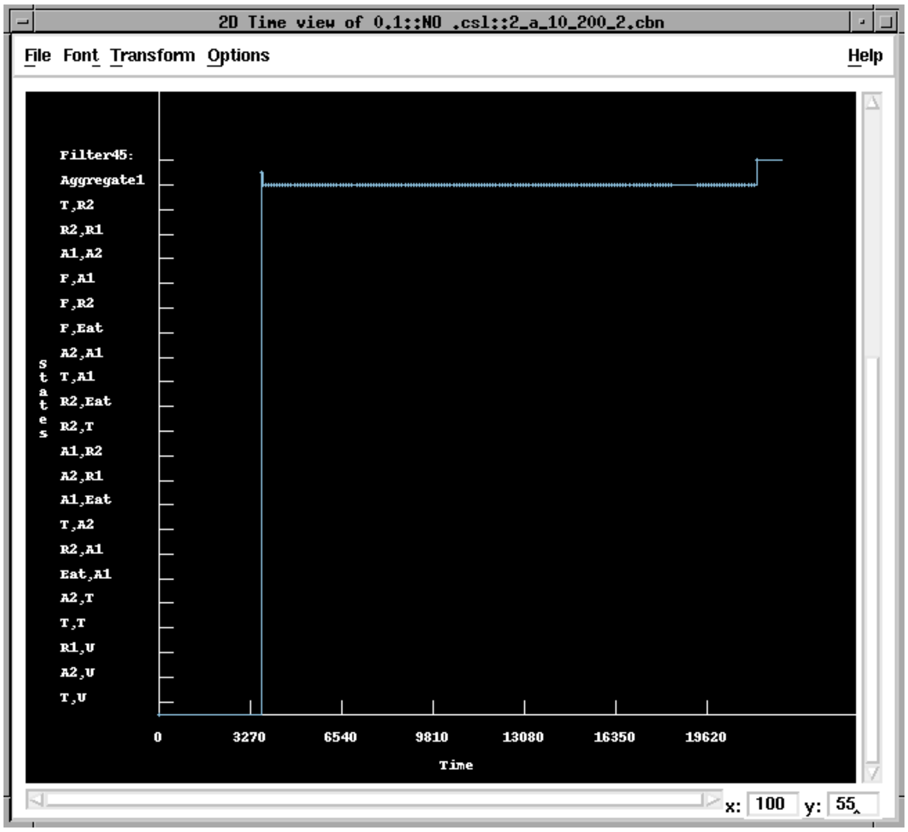


Figure 4.13: 2D time view after both pattern and filter aggregate transforms are applied.

in the "Occur" column.)

4.4.2.3 How the Views Change After Filter Aggregation

2D View

After you click on "Filter" in the histogram, you should see a 2D time view similar to Fig. 4.13. Two new states are present: Filter44 and Filter45. (Only Filter45 is labeled by the y-axis labeling algorithm in the font size used.) The entire graph now consists of either the pattern aggregate state ("Aggregate43") or one of these two filter states.

Text View

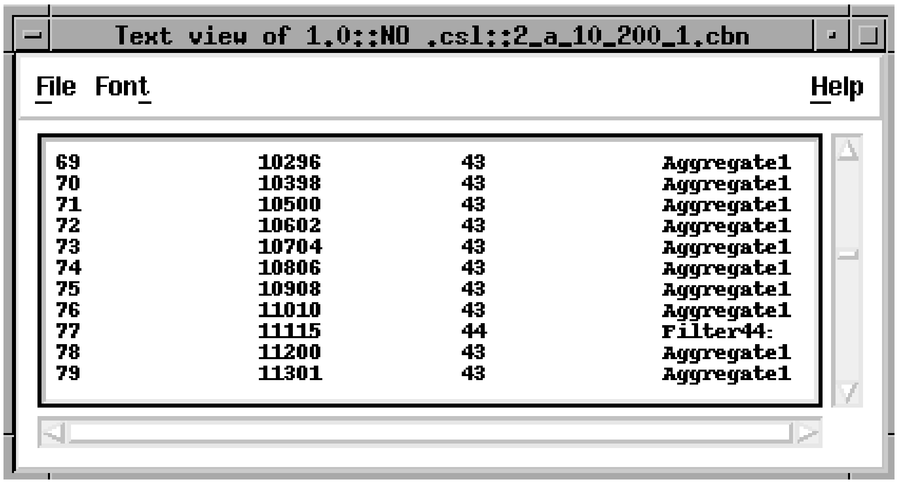


Figure 4.14: Text view after both pattern and filter aggregate transforms

Now change focus to the text view window and select "File → Update". The result is a window similar to Fig. 4.14. Note that states labeled "Filter44" and "Filter45" are now present in the view. None of the original system states (e.g., those in Fig. 1.13 on page 36) remain.

Model View

Now change focus to the model view window and select "File → Update". Observe in the resultant window (shown in Fig. 4.15) that there are now only 3 states (model states 43, 44, and 45), compared to the 43 states we started out with before applying any transforms.

4.4.2.4 Querying the Definition of a Filter Aggregate State

Recall that in §4.4.1.4 that one could querry the definition of a pattern aggregate state comprising the hierarchical semi-Markov model. Similarly, one can querry the definition of a filter aggregate state.

Each filter state aggregates one or more runs. (Recall from §4.4.2.1 that a run is a subsequence of a model states in a member of $\mathcal{E}_v$ such that all states in the subsequence are to be removed by the filter threshold – meaning the state's occupancy time is less than 7% – and the predecessor and successor states of the subsequence in the model state sequence have occupancy time greater than 7%.) Thus the definition of an aggregate state specifies the set of runs and the relative frequency of each run. A filter aggregate state is thus equivalent to an acyclic Markov process.

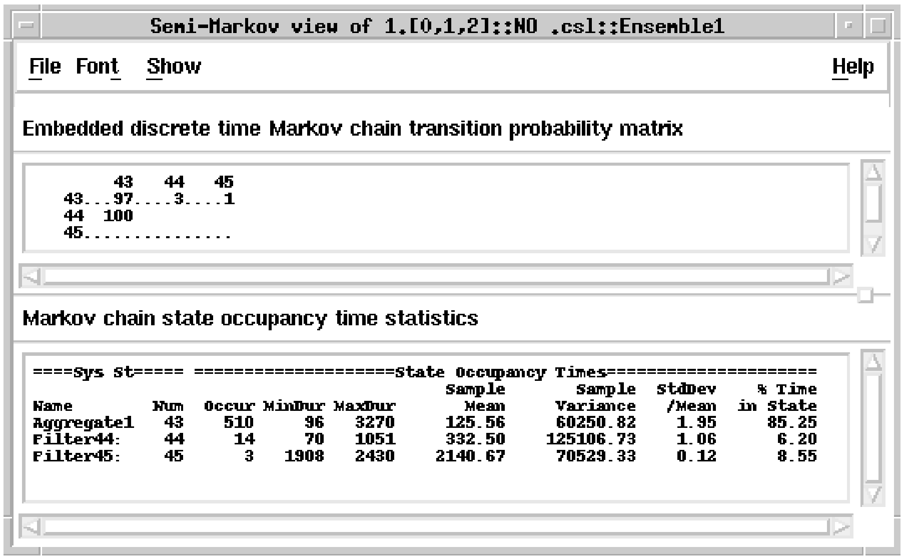


Figure 4.15: Model view of entire ensemble after both pattern and filter aggregate transforms

Let's look at the definition of state filter44. Select "Options → Define aggregate state → Filter". A dialog box similar to Fig. 4.9 pops up. Enter the ensemble's id ("0" from the list box in the root window) in the top field, "44" in the second field, and hit apply. The window shown in Fig. 4.16 should appear. The window lists a sequence of runs. Scroll through the window using the vertical scroll bar. For example, the first run occurs one time, consists of R1,T; R2,T; A1,T; A2,T; Eat,T, and has predecessor and successor state 43.

4.5 Revisiting the Question of Homogeneity

Now let us return to the question of whether the ensemble of three Dining Philosopher program TSSS's is homogeneous. Recall that §3.3 proposed four methods: visualization, adjacency matrix, KW test, and pattern coverage. Before we applied transforms, the visualization and KW test supported the hypothesis of homogeneity. (The reader might try visualization and the KW test for the transformed ensemble as an exercise.) However, the adjacency matrix did not. Let's look at the adjacency matrix now that we've transformed the ensemble into just three states. Change focus to the root window. Be sure that ensemble 0 is selected. Select "Views → Homogeneity → Adjacency matrix". In the "Select TSSS(s) to view" dialog box (Fig. 3.4), select all three ensemble members in the list box and "Combined" under "TSSS display method". Then

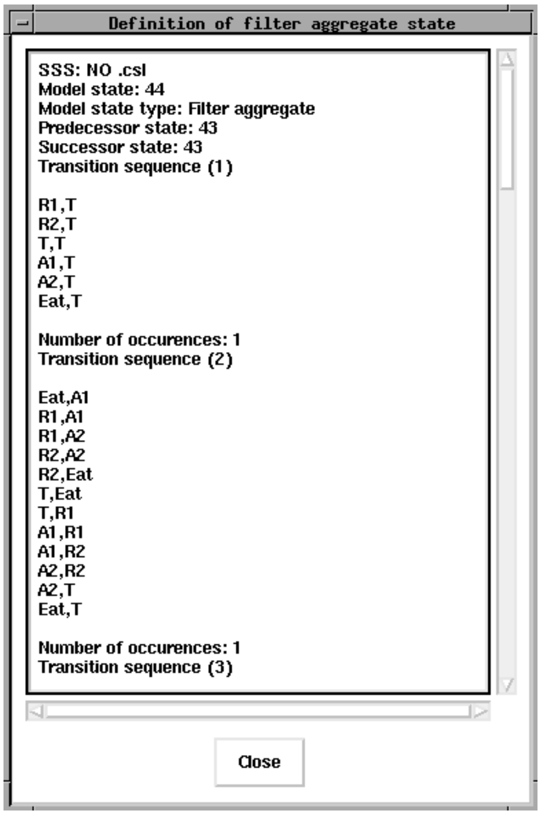


Figure 4.16: Definition of a filter aggregate state resulting from "Options → Define aggregate state → Filter".

select "Apply". The window shown in Fig. 4.17 should appear on your monitor. This entry has all 100% entries, which now *supports* the homogeneity hypothesis. Note that it is much easier to examine this small three state matrix than the larger adjacency matrix of the untransformed ensemble.

Next consider pattern coverage. When we selected a pattern earlier (Fig. 4.4 on page 79), we found that the pattern selected represented between 78.47% and 81.50% of the states in each ensemble member (Fig. 4.5). The fact that the same pattern represents a similar percentage of all members also supports the homogeniety hypothesis.

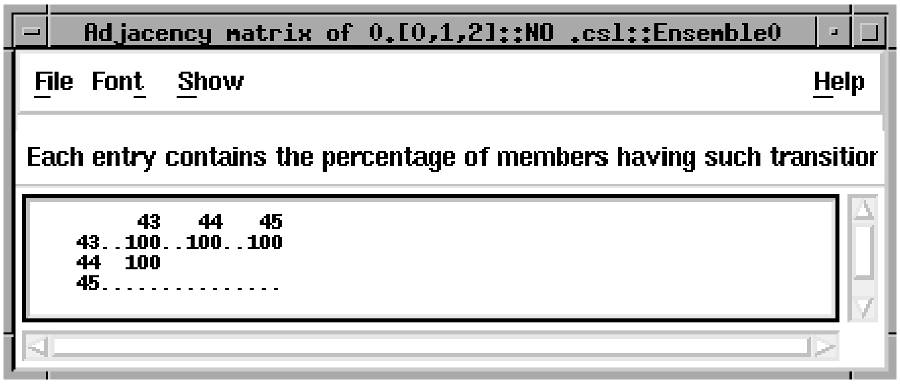


Figure 4.17: Adjacency matrix for three Dining Philosopher program TSSS's after clipping, pattern aggregation, and filter aggregation.

4.6 Projection Tranforms

Projections are transforms that modify the original .csl file. Projections are stand-alone commands that one executes before opening a .csl file into CHITRA93. They take a .csl file as input and produce one or more .csl files as output. The commands, which all start with the letter "p", are listed in Table 4.3. Documentation of each command is available in a text file in with the same name as the command in directory $CHITRA/man.

4.7 Consistency of Views

Whenever you apply a transform in a time or event window, all other time and event windows of the same instance of the TSSS are automatically updated to show the result of the transform. However, you must explicitly select the "Update this view" menu item under the "File" menu of text and model view to make these views consistent with the time or event window in which the transform was selected.

4.8 Writing a Transformed Ensemble to a File

Select "File → Save ensemble" (Table 1.1 on page 1.1) in the root window to write a file that contains a list of transforms applied to the ensemble selected in the root window, along with a list of the ensemble members, in a file whose name ends with .xls. A dialog box similar to Fig. 1.20 on page 47 will appear to allow you to name the file.

Command	*Function*
ped	Eliminate duplicate state vectors in successive records of a .csl file
pic	Isolate one component of a state vector (Example use: If vector represents state of each process in a parallel program, look at one process's behavior and ignore the rest.)
pmu	Map the values of each state vector component to new values (Example use: Given a TSSS from a LANalyzer with 48-bit physical addresses from a few hosts, map each 48-bit address to a one byte number to reduce storage and improve human readability.)
prc	Replace a component value in all state vectors by a new value (Example use: In a TSSS representing an address reference trace, replace addresses of all elements of array A by a single address.)
psc	Split the ensemble into multiple ensembles, one per state vector component (Example use: If system state vector represents state of each process in a parallel program, apply psc to look at each process's behavior separately)
psr	Scale the range of numeric state vector components (Example use: Reduce state space by representing thousands of bytes as packet length, rather than ones of bytes, in network traffic TSSS's)
psv	Split the ensemble into multiple ensembles, such that all TSSS's in an ensemble have a common value for one state vector component (Example use: Filter out all traffic to a particular host in a network traffic TSSS.)

Table 4.3: CHITRA93 projection commands.

Chapter 5

Lesson 5: Creating a CHAID-based Ensemble Model

A fundamental question that CHITRA93's models of ensembles addresses is, "When will a particular state, say σ, occur?" The CHAID (Chi-squared automatic interaction detection) [11] based model directly addresses this question. The model can uncover more complex dependencies about when a state σ occurs than can the semi-Markov model, but requires more computation to do so. Thus the CHAID-based model is used typically to analyze a handful of states, rather than analyzing all states as the semi-Markov model does.

The CHAID-based model examines all occurrences of σ in the ensemble, as well as W_l states preceding each occurrence, and W_u states following each occurrence. Thus the model is built by examining a window of $W = W_u + W_l + 1$ states. Suppose that the value of state S_k in a TSSS is σ. CHAID asks, "which state S_{k+l} $(-W_l \leq l \leq W_u \wedge l \neq 0)$ best predicts the fact that the value of S_k is σ?". Discussion of how CHAID answers the question is postponed until §5.3. For each state σ analyzed, CHAID displays a table listing its estimate of the probability that each position in $[-W_l, W_u]$ is the "worst" predictor. Generally, the position with the lowest probability is the "best" predictor, denoted $\hat{l}$. The CHAID-based model also displays a transition matrix. The entries for the column labeled by state σ give the probability distribution of transitions into σ conditioned on the state l positions away.

There is important caveat on using the CHAID-based model: the model is based on a statistical test of a hypothesis, as explained in §5.3. As explained in §3.3.3, statistical tests are sometimes wrong. Therefore the conclusion as to which value of k "best" predicts occurrences of a state are sometimes wrong.

5.1 Comparing the CHAID-based and Semi-Markov Models

The *Markov assumption* underlying the semi-Markov process is that the probability distribution of the current model state conditioned on knowing only the previous state is equal to the distribution conditioned on knowing all past states. Thus we can ignore all but the previous states when modeling the process. The CHAID-based model provides a way to test the validity of the Markov assumption.

The semi-Markov model predicts occurrences of a state σ by computing an estimate of the one step, *forward* transition probability matrix. The cell in the row for model state σ' and column for model state σ contains an estimate of the probability that whenever the system is in state σ', it will next enter state σ.

The CHAID-based model identifies what position $\hat{l}$ best predicts occurrences of a state σ. It displays a *backward* or *forward* transition matrix: the cells for column σ display the probability distribution of the state that the system was in $\hat{l}$ steps earlier (if $\hat{l} < 0$) or later (if $\hat{l} > 0$). A value of $\hat{l} = -1$ corresponds to the semi-Markov model.

5.2 Example

The CHAID-based model does not provide additional information about the Dining Philosophers program traces used in earlier models. Thus we will use a different example for this chapter: the central server computer system model shown in Fig. 5.1. Let's examine what happens to a single job in this system. The job leaves the terminal and, with equal probability, visits one of the two CPU's. Next the job visits one disk. Then the job returns to the terminal. There is a service time associated with each device. Let the time spent at either CPU or disk be a constant of two time units. When a job visits the terminal, the think time distribution is discrete with possible values of 1 or 2 time units, each with equal probability. Thus there is a short think time (1 unit) or a long think time (2 units). Finally, and most importantly, anytime a job leaves either CPU, it goes to Disk1 if its last think time was short, but to Disk2 if its last think time was long.

Ideally, CHITRA93 should be able to identify the correlation of which disk is visited to the think time value. We will see that this is detected by the CHAID-based model, but not the semi-Markov model.

Start CHITRA93. Open $CHITRA/examples/CHAID/CentralServer.csl. Open a text view (not shown). Note that there are six states: two for the two. cpus, two for the two disks, and two for the two think time durations. Therefore the .csl file uses system state "ThinkS" to represent a short think time, of 1 time unit, and state "ThinkL" to represent a long think time.

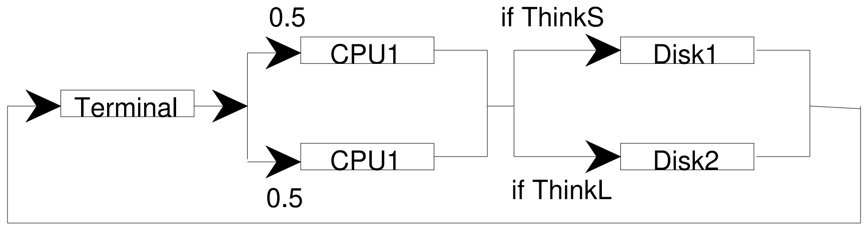


Figure 5.1: Central server computer system model.

Open a semi-Markov view, which should match Fig. 5.2. What information does the semi-Markov view give us on the conditions under which a job visits each disk? By looking in the transition matrix in the top pane at the columns for Disk1 and Disk2 (model states 2 and 4, respectively), we see that *a job moves from either cpu to either disk*. Thus the semi-Markov model allows a job that last experienced a short think time to visit a CPU and then visit Disk 2, but the real system would *never* allow allows this! The semi-Markov model does not identify the correlation of think time and which disk is visited.

Next open a CHAID-based model. Two windows will be drawn on your screen. The window in the foreground, which is the key window, is entitled "Model Parameters", and is shown in Fig. 5.3. The larger, background window is actually *blank* when the view is first selected. Unlike the semi-Markov model, there are a variety of parameters which you may optionally set. These are important because the default settings may take a very long time to work.

5.2.1 Setting the CHAID-based Model Parameters

Merge method

The box under "Merge method" in Fig. 5.3 contains two diamonds, called *radio buttons*, labeled "Free" and "Monotonic". The term "radio" denotes that only one of the diamond buttons can be depressed at a time, similar to a button to select a station on a car radio. The shaded diamond (next to "Free") denotes which button has been pressed. The top field, labeled "Merge method", is normally set to its default value, "Free". The field's use requires understanding of how CHAID works, and is thus postponed until §5.3.4. Do not modify the merge method setting.

Type of analysis

The next radio buttons specify whether the analysis is independent or joint. In independent analysis, the CHAID procedure is applied independently to each state that the user selects (via the "List of states to analyze" buttons, described later), or once for all user selected states. "Independent" results in a potentially

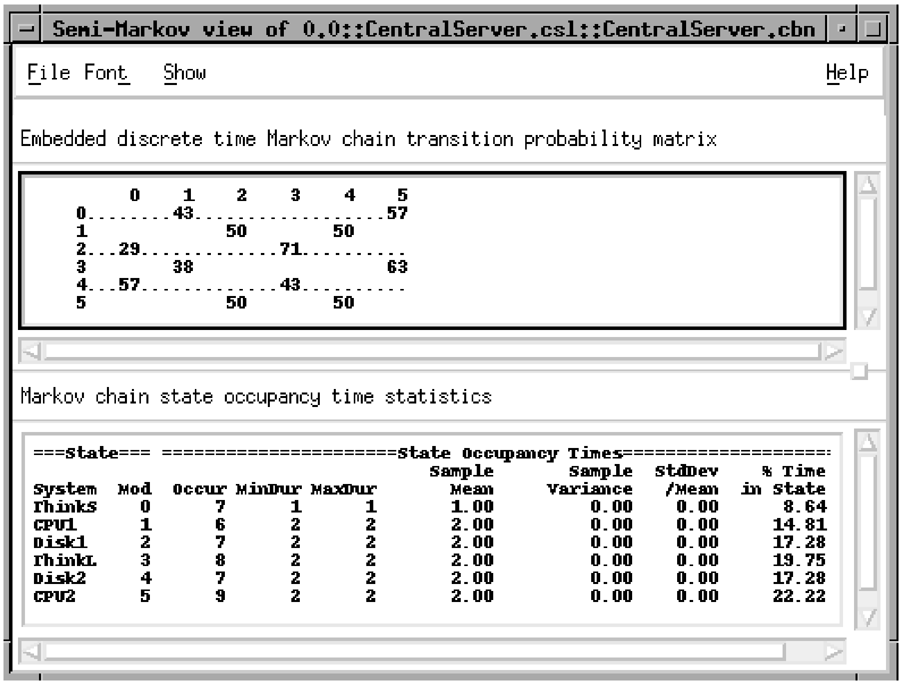


Figure 5.2: Semi-Markov model of central server model of Fig. 5.1.

different value of $\hat{l}$ for each state analyzed, while "joint" finds one value of $\hat{l}$ that is best for *all* states analyzed. For example, you could test the validity of the Markov assumption by selecting "joint" and seeing if CHAID concludes that "$\hat{l} = -1$". "Joint" can only be used if you select two or more states to analyze.

Click the diamond to the left of the word "joint".

Dependent position

The window size is selected by the three radio buttons under "dependent position". One can analyze just the past (implying that $W_u = 0$), just the future (implying $W_l = 0$), or both (implying $W_l \neq 0$, $W_u \neq 0$). Selecting any of the three radio buttons displays a second dialog box, shown in Fig. 5.4, in which the non-zero values of W_l and W_u are entered.

Click on the diamond to the left of "Analyze past". Then set the past window size (e.g., W_l) to 2 in the window that pops up, and hit "OK".

Significance level

This value specifies α, the desired probability of a Type I error. The same comments that apply to Type I and Type II errors given in §3.3.3 apply to the

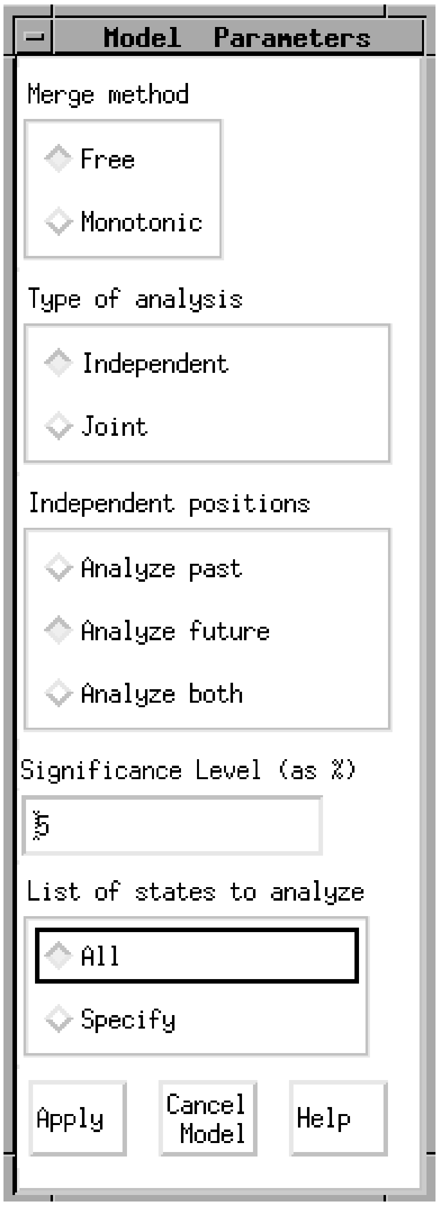


Figure 5.3: CHAID-based model parameters visible when model is first opened or when "File → About this view" is selected.

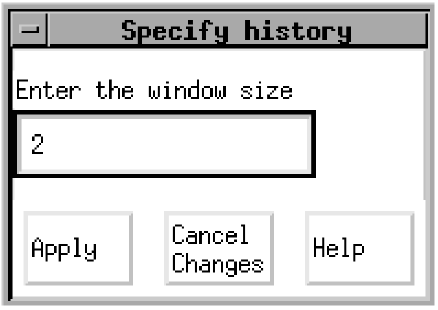


Figure 5.4: Dialog box that pops up when radio buttons "Analyze past" or "Analyze future" are selected in Model Parameters window (Fig. 5.3)

CHAID-based model.

List of states to analyze

Finally, the "List of states to analyze" radio buttons control how the set of samples passed to CHAID is formed. Let $\mathcal{S}$ denote a set of model states. The "All" button sets $\mathcal{S}$ to all model states. Clicking on the "Specify" button pops up a second box (not illustrated) containing a list of all model states. When "Type of analysis" is joint, the user must select at least two model states, forming set $\mathcal{S}$. The result is to pass only those model state subsequences of length W in all TSSS's in $\mathcal{E}_v$ satisfying $s_k \in \mathcal{S}$ to CHAID (which is all subsequences in the case of "All".)

Click on "Specify". In the resulting list box, click on state "Disk1". Then move the mouse pointer to state "Disk2", hold down the control key, and select "Disk2". The result should highlight just the two states, as shown in Fig. 5.5. Thus $\mathcal{S} = \{$ Disk1, Disk2 $\}$ in our example.

Click on the "Apply" button to start the CHAID calculation; the "Model Parameters" window will vanish, focus will change to the background window (entitled "CHAID-based Model view of ..."), and after a short delay, you will see window content similar to Fig. 5.6.

5.2.2 The CHAID-based Model Output

The CHAID-based model (Fig. 5.6) is displayed in three window panes.

Top Pane

The top pane contains a matrix with one column per state in $\mathcal{S}$. The numbers in square brackets ("[-2]" for both states Disk1 and Disk2 in Fig. 5.6) list the best predictor position, $\hat{l}$. *Therefore the best predictor for the branch*

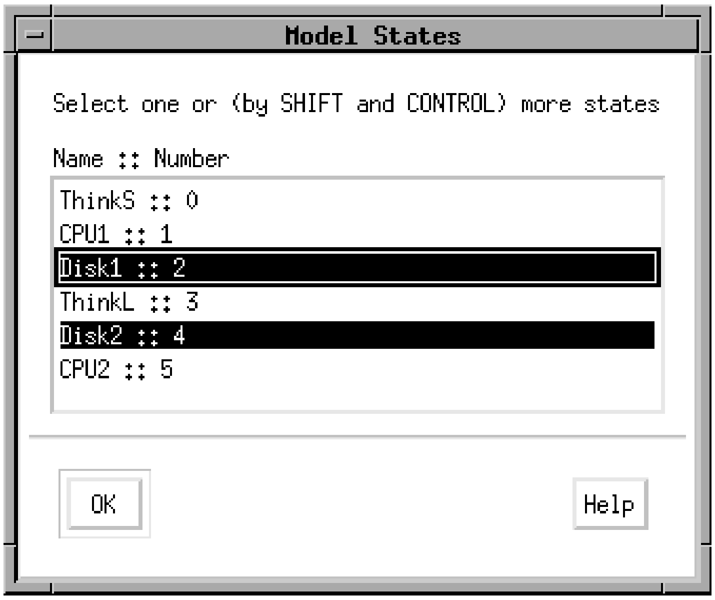


Figure 5.5: Dialog box used to select which states are to be analyzed by CHAID.

outcomes is the state two steps back, thereby violating the Markov assumption. The rows correspond to each model state found in any independent position. The percentages in the matrix list the number of times that the predecessor of the model state labeling the column was the model state labeling the row. In this case 100% of the time state 2 (Disk1) is preceded two steps back by state 0 (ThinkS). Also, 100% of the time state 4 (Disk2) is preceded two steps back by state 3 (ThinkL). *Thus the CHAID-based model has correctly identified the correlation of routing to a disk based on think time duration, in unlike the semi-Markov model.*

Middle Pane

The middle pane, labeled "P-values ...", contains two tables. The first contains on column for each state in $\mathcal{S}$. The rows each correspond to a position in the selected window. The cells list the probability that a position is the "worst" predictor of a state. Thus under the column labeled "2" (for state Disk1), the probability (or "p-value") that position -1 is the worst predictor is 1.0, while the probability that -2 is the worst predictor is 0.000183. §5.3 explains how the probabilities are calculated.

The second table in the middle pane also contains one column per selected state and one row per position. The table cells contain a *critical p-value*. To understand the use of these values, recall that the KW test (Fig. 3.8) reports

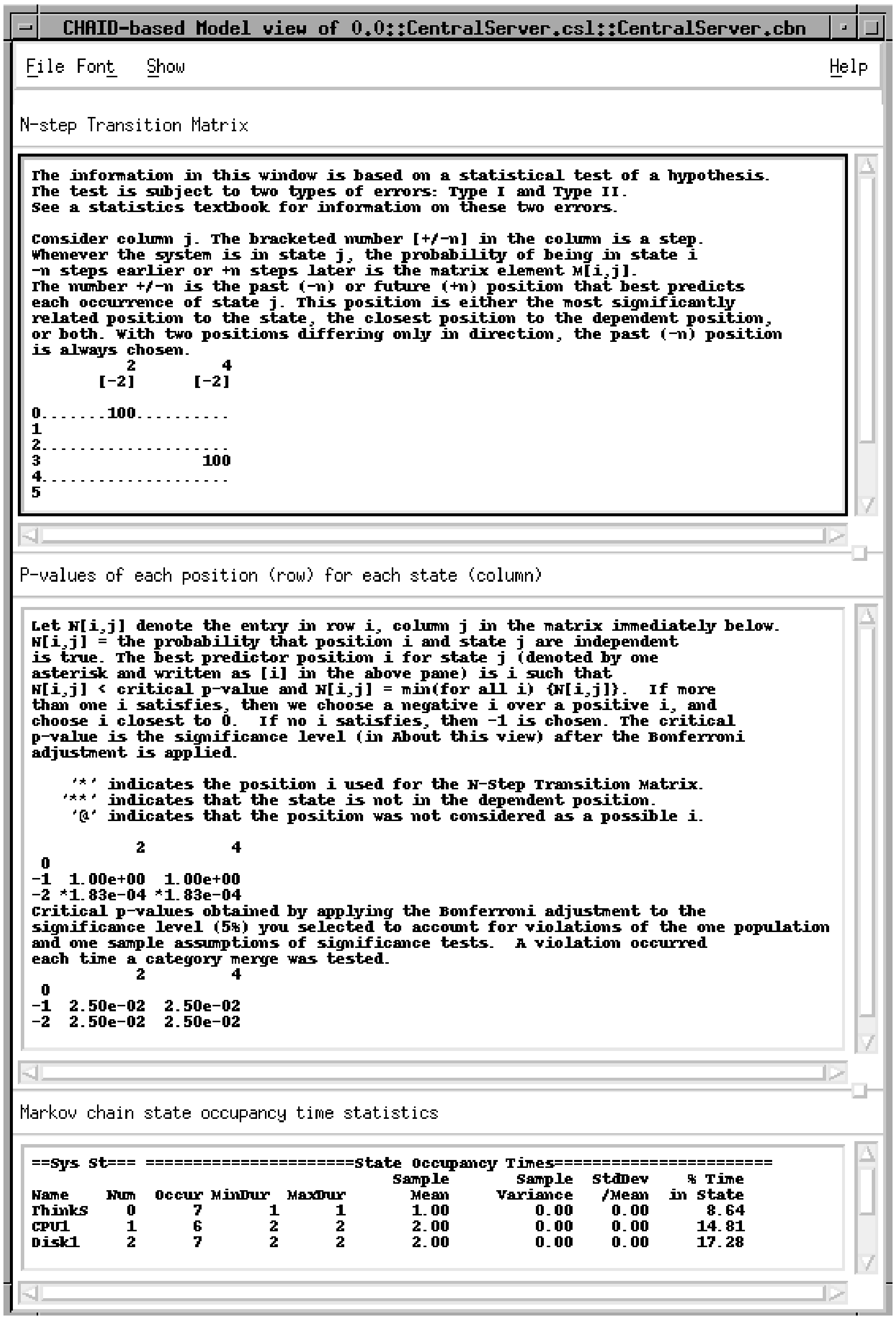


Figure 5.6: CHAID-based model of states Disk1 and Disk2 in the central server model of Fig. 5.1.

the value of a statistic H and the "critical" Chi-square value statistic; if the critical value exceeds the statistic H then the hypothesis is accepted. Similarly, CHAID tests a hypothesis, as explained in §5.3, that a position independent, which means that it is a bad predictor. If the p-value in the upper table of the middle pane exceeds the critical value in the lower table, then the independence hypothesis is accepted. Thus a position is only a candidate for the best predictor if the independence hypothesis is *rejected*. Therefore only if the p-value in the top table is less than the corresponding critical value in the lower table is the position considered a candidate for best predictor. In this case only the value 0.000183 is less than the critical value 0.025; hence only position -2 is a candidate for best predictor. Thus CHAID concludes that $\hat{l} = -2$.

If there are more than one candidate positions, the value with *minimum* p-value is selected, because it is the position with the smallest probability of being the worst predictor, and hence the highest probability of being the best predictor.

If more than one position has the minimum p-value, then the value is chosen as follows: If any of the positions is a negative number, the largest of these is used. Otherwise the smallest positive number is used. In this way, the modeler knows the minimum amount of history that is required to best predict a model state. In fact, the rule automatically reveals when the Markov assumption yields the best prediction of a state because if the p-value for $l = -1$ equals the minimum, then $\hat{l} = -1$ will be reported as the best predictor. Thus a user can see when CHAID cannot improve on the semi-Markov model.

In any event, a single asterisk is printed just to the left of the p-value that is selected as the best predictor.

Bottom Pane

This is identical to the table in the bottom pane of the semi-Markov model.

5.3 How the CHAID-based Model Works

The heart of the CHAID-based model is a test of the following independence assumption: If the state in a certain position l is independent of occurrences of a state σ of interest, then *we should observe all model states in position l, and the relative frequency of each model state should be equal*, excluding variations due to random sampling. A Chi square test is used to tell us whether deviations from equal frequency can be accounted for solely by random sampling, or whether there is in fact a dependency between position l and occurrences of state σ. The Chi square test yields the p-values reported in the upper table of the middle pane of the CHAID-based model window.

Given a data set that contains vectors in which one component represents the discrete values of a dependent variable and the remainder represent the discrete values of a set of independent variables, the CHAID method partitions the independent variable values into mutually exclusive and exhaustive subsets. Let d (respectively, c) denote the number of values of the dependent (respectively, independent) variable. The partitioning procedure first constructs c contingency tables [15, pp. 249–253], each of dimension $c \times d$, one for each vector component that represents an independent variable. In a contingency table, each row (column) represents one dependent (independent) variable value. Let $d(i)$ (respectively, $c(j)$) denote the value corresponding to row i (column j) of the contingency table (for $0 \leq i < c$ and $0 \leq j < d$). Table element i, j represents the fraction of the data set in which the value of the vector components to which the table corresponds has $d(i)$ as the dependent variable value and $c(i)$ as the independent variable value. A procedure is then applied to reduce the number of rows by combining (according to a constraint) values so as to maximize the significance level of the table. The procedure calculates, for all possible value combinations, a χ^2 statistic; the table with the highest statistic is the most significant. The reduced table with maximum χ^2 statistic for *all* independent variables are compared; the one with the highest χ^2 value determines the final answer of which independent variable best predicts the dependent variable.

CHITRA93 implements the CHAID method, but employs a slightly different significance test based on the Bonferroni inequality recommended by Biggs, DeVille, and Suen [6]. The modified test keeps the type 1 error rate in calculating the significance of contingency table(s) yielding the best partition below the user-specified significance level. (A type 1 error is to call a partition significant when in reality no relationship exists between the dependent and independent variables.) Hence we call the model "CHAID-based".

The precise algorithm used is stated in [7, Appendix A].

5.3.1 "Independent" Versus "Joint"

When "Type of analysis" in the Model Parameters window is set to "Independent", then for each state σ selected for analysis, the contingency tables always have $d-2$ columns corresponding to states σ and $\overline{\sigma}$ (i.e., the complement of σ). Otherwise, for "joint", if n states are selected for analysis, then $d = n$, and each column corresponds to one of the selected states.

5.3.2 How Critical P-values Are Derived

The critical p-values in the bottom table of the middle pane of the CHAID-based model are calculated as follows. If no merge is applied to the contingency table used for a position, then the critical p-value for this position equals the significance level α chosen by the user in the "Model Parameters" window (See §5.3). Otherwise the Bonferroni inequality is used to compute the p-value based on the user chosen value of α and the number of merges that CHAID applies to the contingency table.

5.3.3 CHAID-based Model's Computational Cost

The computational cost of using CHAID grows linearly with W. Thus the CHITRA93 user must choose the window size by balancing the time he will allow for analysis with his estimate of the distance (in number of positions) between the state in position k and the best predictor state. A logical choice for W is a positive integral multiple of the period in number of transitions of the mean number of transitions required for the chain (matrix P) to return to state s_k. The CHAID-based model time and space complexity is analyzed in [7, Appendix A].

5.3.4 "Free" Versus "Monotonic"

Normally you should always use "free", and it is always safe to use "free".

The system states in the original trace files may be either *numerical* or *categorical*. Numerical states are totally ordered, such as 1-dimensional state vectors representing the length of packets in a trace of network traffic, or the number of busy processors in a trace of a parallel computer. A system state of 64 (meaning a packet of length 64, or 64 busy processors, respectively) is less than a system state of 65. Rarely are the system states in a trace file numerical. In contrast, categorical system states are the complement of numerical states: either no order exists, or a partial order exists. The dining philosophers states from the preceeding chapters as well as the central server states in this chapter are categorical: state "Eat,T" is neither "less than" or "greater than" state "A1,R1". (§1.3 of [4] contains further information on numeric versus categorical states.)

If you load an ensemble that contains numerical system states, if CHITRA93 retains this order in mapping system to model states, and you have either not transformed the ensemble or only applied clipping and the "ped" projection (Table 4.3), then the *model* states of the ensemble are also categorical.[1] Select the "monotonic radio button under "Merge method" only if the model (not system) states are numerical. Otherwise, for categorical states, select the "free" radio button.

We stated earlier that CHAID combines rows according to a constraint. The constraint is whether the categories of a dependent variable (which are all model states occurring in a position in the window) are a totally ordered set (e.g., numerical) or not (e.g., categorical). If the set is ordered (referred to as *monotonic* clustering), then CHAID will cluster successive values so that a row can be labeled by a range of values.

[1] CHITRA94 automatically determines when states are numerical versus categorical.

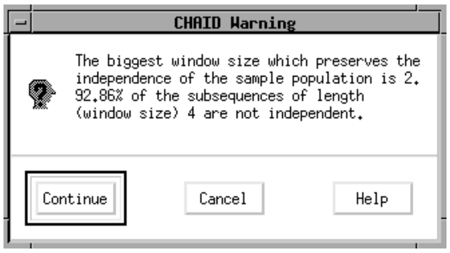


Figure 5.7: Result of selecting a window that is too large to satisfy the independence assumption.

5.4 Violation of the Independence Assumption

An assumption underlying proper use of CHAID is that each window of W states extracted from each TSSS in an ensemble is independent: no state is common to two windows. This was the case in §5.2. However, if occurrences of any two states in S occur less than W positions apart in a TSSS, then the independent sample assumption is violated.

CHITRA93 will check whether the assumption is violated and inform you when you click on "Apply" in the "Model Parameters" dialog box. To try this, select "File → About this view" to display the "Model Parameters" box again. Click on "Analyze past" and select "3". Then click on "Apply". The information box in Fig. 5.7 will appear. The window reports the largest window size that meets the assumption (e.g., "2"), so that you can reset the "Dependent position" in the "Model Parameter" box. It also reports the percentage of windows that violate the assumption (i.e., 92.86% in this case).

If the independence assumption is violated, then the chance of a Type I or Type II error in the best predictor positions that CHAID reports increases. The degree to which the chance of error increases grows as the percentage reported in Fig. 5.7 grows.

5.5 Limitations of the CHAID-based Model

There are several limitations in the CHAID-based model. (A new version as well as a simpler model based on the univariate distribution are being explored for CHITRA94.)

- You must use a small sample size. If the number of occurrences of a state in $\mathcal{S}$ is a small integer, the CHAID-based model is effective. However, if the number of occurrences is in the hundreds or larger, the Chi square test underlying CHAID will always accept that the hypothesis that a state in $\mathcal{S}$ and every position are independent for all states.

- You cannot analyze a case that generates contingency tables with many zero values. The "independent" analysis in particular generates many zero entries. You are less likely to encounter this problem with "joint" analysis.

  The Chi square statistic for a contingency table grows as the number of zero entries grows. Unless a small number of positions is used, then the Chi square statistic for the table will always exceed the Chi square statistic, and hence the test will always accept that the hypothesis that a state in $\mathcal{S}$ and every position are independent for all states.

One final note concerns the transitions that one may make inferences about using the CHAID-based model. In general, one may only make inferences about those transitions in $\mathcal{E}$ that are included in the population used for all contingency tables that the CHAID method analyzes. If $\mathcal{E}$ contains N transitions, then selecting the "Independent" radio button in Fig. 5.6 results in every contingency table representing a population of size N; hence "Independent" allows inference about all N transitions. In contrast, selecting the "Joint" radio button will pass to the CHAID algorithm a data set containing M transitions, where M is the number of transitions into a state in set $\mathcal{S}$. Only if the "All" radio button is used does $M = N$. Hence use of the "joint" radio button allows inference only about transitions into states in set $\mathcal{S}$. In our example, $\mathcal{S} = \{\text{Disk1, Disk2}\}$, and we only use the model to make inferences about the conditions under which transitions into these two states occur. The semi-Markov model is used to make inferences about transitions into the remaining states.

5.6 For Further Information

Please refer to Horacio Cadiz's M.S. project report [7] for a detailed discussion of CHAID. The file $CHITRA/examples/TCP_IP/TCP-IP_sorted_unique.csl is used in the thesis to exemplify CHAID.

Chapter 6

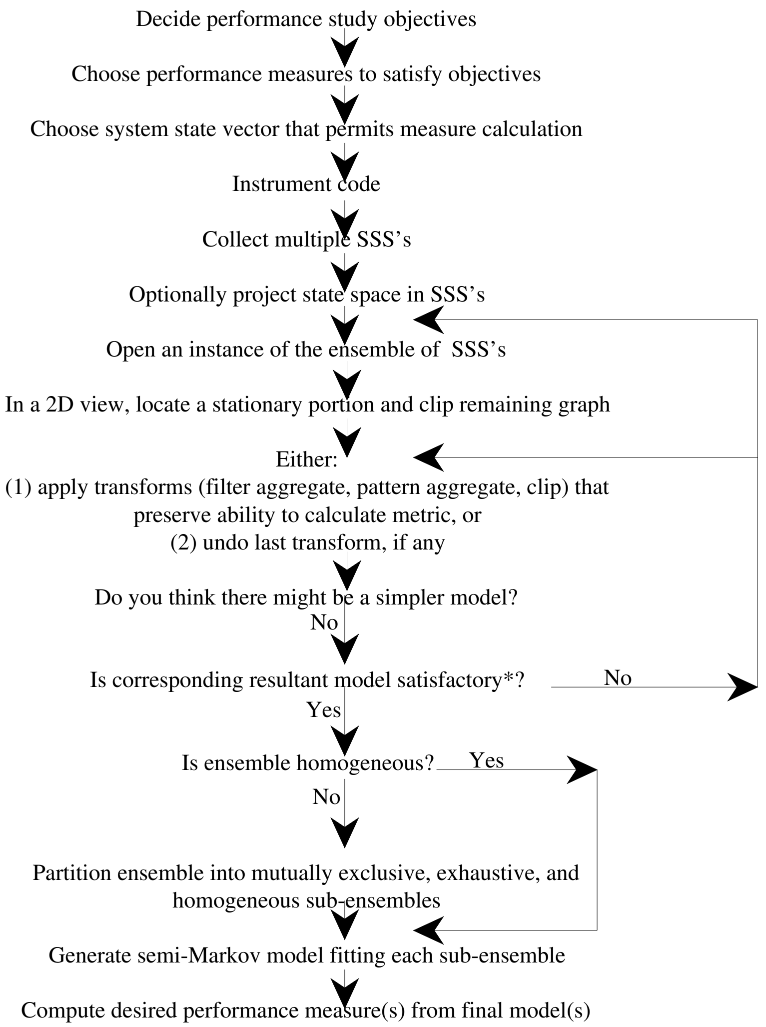
Suggested Model Building Process

Building models of trace data is to some degree an art. CHITRA93 provides a variety of visualization, transform, and modeling methods for you to choose from. However different individuals, given the same ensemble, may arrive at different models. This chapter suggests a procedure that you may wish to follow when you begin using CHITRA93.

6.1 Procedure

Fig. 6.1 suggests an initial process that you might use to building a model with CHITRA93. Note that one must choose the "system state" in the original .csl files to fit the objective of the performance study which the models that CHITRA93 generates will be used for. Also, transforms that are applied must preserve in the top level model the ability to calculate any desired performance metric from the model ultimately created.

Fig. 6.1 contains a step that says "Open an *instance* of the ensemble of TSSS's". Note that later on, if the model is unsatisfactory, then one path on "No" leads back to open another instance. What this refers to is that one can open the *same* ensemble multiple times. Each open copy is called an *instance*, and is listed in a separate line in the list box in the root window. One can transform a view of one instance, and only that instance is affected. The other instances are not affected. Thus one can compare transform sequences by loading multiple copies of an ensemble and transforming each instance separately.



"Satisfactory" means (1) measure computed from model is sufficiently close to measure computed from SSS, and (2) model state space is sufficiently small and simple to permit insight.

Figure 6.1: Process of model building with CHITRA93.

6.2 Suggested Use for CHAID

A state σ' is called *branching* if there are transitions out of σ' into two or more states in the ensemble.

We have found that a good use for CHAID is the following. Always try the semi-Markov model as the initial model. After all, it quickly gives a model of *all* states. Then use the CHAID-based model to analyze as set $\mathcal{S}$ with analysis type "joint" each set of branch state outcomes. The purpose of using CHAID is to determine if the one-step probabilistic semi-Markov model is a good way to predict which branch outcome is taken, or whether some a step other than -1 should be used.

For example, in Fig. 5.2, each CPU state (model states 1 and 5) are branching states with two outcomes: states Disk1 and Disk2. Hence we apply CHAID with $\mathcal{S}$ = { Disk1, Disk2 } as shown in §5.2. In addition the two think states are branching states with common outcomes (i.e., the two CPU states), and the two disk states are branching states with common outcomes (i.e., the two think states).

Another illustration of using CHAID to analyze branching state outcomes is given in [3].

Bibliography

[1] M. Abrams, N. Doraswamy, and A. Mathur, "Chitra: Visual Analysis of Parallel and Distributed Programs in the Time, Event, and Frequency Domains," *IEEE Transactions on Parallel and Distributed Systems*, Vol. 3, No. 6, pp 672-684, November 1992.

[2] M. Abrams and N. Doraswamy, *An Introduction to Visualizing Program Execution Dynamics with Chitra,* Department of Computer Science, Virginia Tech, TR 92-23, May 1992.

[3] M. Abrams, T. J. Lee, H. Cadiz, and K. Ganugapati, "Beyond Software Performance Visualization," to appear in *Concurrency: Practice and Experience*; also appeared as Technical Report TR-94-07, Dept. of Computer Science, Virginia Tech.

[4] M. Abrams, A. Batongbacal, R. Ribler, D. Vazirani, "CHITRA94: A Tool to Dynamically Characterize Ensembles of Traces for Input Data Modeling and Output Analysis," submitted for publication; also appeared as Technical Report TR-94-21, Dept. of Computer Science, Virginia Tech.

[5] D. P. Bertsekas, "Dynamic Behavior of Shortest Path Routing Algorithms for Communication Networks", *IEEE Trans. on Automata Control AC-27*, 60–74, 1982.

[6] D. Biggs, B. de Ville, E. Suen, "A Method of Choosing Multiway Partitions for Classification and Decision Trees", *J. of App. Stat. 18* (1), Jan. 1991, "49–62".

[7] H. Cadiz, *The Development of a CHAID-based Model for* CHITRA93, M.S. project report, Dept. of Computer Science, Virginia Tech, February 1994.

[8] E. W. Dijkstra, *Cooperating Sequential Processes*, Technological University, EWD-123, Eindhoven, The Netherlands, 1965.

[9] N. Doraswamy, *Chitra: A Visualization System to Analyze the Dynamics of Parallel Programs*, M.S. thesis, Dept. of Computer Science, Virginia Tech, Dec. 1991.

[10] K. Ganugapti, *The Design and Implementation of* CHITRA92*, a System to Empirically Model Concurrent Software Performance,* M.S. thesis, Dept. of Computer Science, Virginia Tech, April 1993.

[11] G. V. Kass, "An exploratory technique for investigating large quantities of categorical data," *Applied Statistics*, 29(2), 119-127, 1980.

[12] D. E. Knuth, *Literate Programming*, Center for the Study of Language and Information, CSLI Lecture Notes Number 27, 1992.

[13] T. J. Lee, CHITRA93*: a Tool to Analyze System Behavior by Visualizing and Modeling Ensembles of Traces*, M.S. project report, Dept. of Computer Science, Virginia Tech, April 1994.

[14] A. Mathur and M. Abrams, *Toward a Machine Assisted Software Performance Diagnosis Methodology,* Department of Computer Science, Virginia Tech, TR 93-12, April 1993.

[15] L. Ott, *An Introduction to Statistical Methods and Data Analysis*, 3rd ed., Boston: PWS-Kent, 1988.

Appendix A

Installing Chitra93

You can obtain the latest version of CHITRA93 using anonymous ftp from vtopus.cs.vt.edu. Retrieve file /pub/chitra/Chitra93v<version>.tar.Z, where <version> is a number similar to 4.122.

A.1 System Requirements

CHITRA93 runs on a Unix workstation with Motif 1.1 or 1.2. At present, CHITRA93 has been tested on the platforms listed in Table A.1. In Table A.1, "*" denotes that a version preceding 4.122 was tested on the platform, and you can expect to do some work to get the current version of CHITRA93 to run on the platform, chiefly modifying file src/Makefile.

Platform name	*version*	*denoted by ⟨platform⟩*
DECstation 3000 + OSF 1.3 or 2.0	4.122	alpha
DECstation 5000 + Ultrix 4.3	4.122	mips
IBM RS/6000 + AIX	*	rs6k
Intel Paragon + OSF	*	paragon
KSR-1 + OSF	*	ksr
NeXTstation + NEXTSTEP + Co-Xist	*	next
SGI Indigo	*	sgi
Sparcstation 10 + Solaris 2.3	4.122	sparc

Table A.1: Platforms under which CHITRA93 has been tested.

The following system is recommended to use CHITRA93:

- at least 32 Mbytes of RAM in the workstation

Directory name	*Space required (Mbytes)*	*Description*
docs	37[1]	This user manual, as a postcript file
examples	14.3	Sample TSSS files in .csl format (See Appendix D)
man	0.025	Manual entries for all binaries
src	13.8	CHITRA93 source code for chitra93 binary (See Appendix E)
⟨platform⟩	8 to 10	Object files compiled from src
tests	1.6	Test suite used to verify CHITRA93 (See Appendix E)
Total:	74.7 to 76.7	

Table A.2: Contents of the top level CHITRA93 directory.

- a color console or a color X-terminal; we suggest a minimum of 10 Mbytes of RAM in the X-terminal. CHITRA93 has been carefully tuned to provide quick response whenever windows are manipulated or redrawn. If you experience sluggish behavior, it may be due to inadequate memory in the workstation or the X-terminal.
- Motif 1.1 or 1.2
- disk space equal to the total listed in Table A.2 plus 1 to 3 Mbytes for the actual executable, binary of CHITRA93. The 1 Mbyte figure is for an operating system with dynamic linking, such as OSF-1. The 3 Mbyte figure is for static linking. The 37Mbyte size of the docs/ directory is uncompressed.
- a swap area of at least 40M (150M is recommended)
- ANSI C compiler (only if you must compile the source code)
- RCS (revision control system) if you plan to modify the source code.
- unix `compress` and the GNU `gzip` package if you wish to open compressed data files with extensions `.Z` and `.gz` from within CHITRA93. CHITRA93 was tested with `gzip` version 1.2.2.

A.2 Installing Chitra93

You should create a directory (i.e., `/usr/local/Chitra`) in which the Chitra source tree will be located; we will denote this directory name by ⟨chitradir⟩.

Copy file Chitra93v<version>.tar.Z via anonymous ftp as stated at the start of this chapter into ⟨chitradir⟩. Then you should (things you type are in the courier font):

1. `cd` ⟨`chitradir`⟩
2. `zcat Chitra93v<version>.tar.Z | tar -xvf`

   The tar command produces the set of directories listed in Table A.2.
3. `rm Chitra93v<version>.tar.Z`
4. `cd src`
5. `make` ⟨platform⟩

   ⟨platform⟩ is one of the names in the rightmost column of Table A.1, such as *alpha*.
6. `make install`
7. Edit file ⟨`chitradir`⟩`/.chitra_fonts` (e.g., Fig. A.1) and replace any font name that are *not* available on all workstations or X-terminals on which CHITRA93 will be run by fonts that are available. Use the xlsfonts command given in §1.1 to identify such fonts.

A.3 Support

If you have suggestions on CHITRA93, we would like to hear them; please send them to chitra@cs.vt.edu. Because CHITRA93 is being completely redesigned in CHITRA94, we seek feedback.

If you encounter any bugs in using Chitra that are not listed in Appendix F, please send the simplest instructions you can devise that reproduces the bug to chitra-bugs@cs.vt.edu. However, we make no promises to fix bugs.

```
# This file contains fonts that will be used by the chitra views
# and dialog boxes that require fixed fonts.  The order in which
# the fonts are listed below corresponds to the names listed
# in the "Font" menu in a Chitra window.  The first item is the default
# font and will be substituted for any of the other three missing
# font.  In the case that all four fonts cannot be opened, the
# font inherited from Chitra root window or from the environment
# will be used; however, all views and dialog boxes that use the
# inherited font may not be aligned correctly.
# Lines starting with a # are ignored.
#
# The first font listed must name a fixed-width font.
#
fixed
#
# The next three fonts listed must name three fixed fonts.  The fonts
# must be  listed in ascending size order.
#
-adobe-courier-bold-r-normal--11-80-100-100-m-60-iso8859-1
-adobe-courier-bold-r-normal--14-100-100-100-m-90-iso8859-1
-adobe-courier-bold-r-normal--17-120-100-100-m-100-iso8859-1
```

Figure A.1: Sample `.chitra_fonts` file.

Appendix B

Grammar of a .csl File

Figure B.1 shows the .csl file grammar. In the grammar,

- letter is one of A through Z and a through z;
- integer is a string of digits without a sign;
- stringconst is a single quote followed by any characters except a single quote followed by a single quote;
- identifier is a letter followed by a string consisting of letters, numbers and underscores; and
- words in upper case are keyword terminal symbols (The parser is case insensitive to keywords.).

Finally, a comment is any sequence of characters excluding *) and enclosed by (* and *). A comment may appear anywhere that white space may go.

$$
\begin{aligned}
\langle program \rangle &\rightarrow PROGRAM \langle program\_name \rangle \; ; \; \langle block \rangle . \\
\langle program\_name \rangle &\rightarrow identifier \\
\langle block \rangle &\rightarrow BEGIN \langle setup \rangle \langle trace \rangle END \\
\langle setup \rangle &\rightarrow SETUP : MAXCOMP = \\
& \quad \langle num\_of\_components \rangle \langle component\_list \rangle \\
\langle num\_of\_components \rangle &\rightarrow integer \\
\langle component\_list \rangle &\rightarrow \langle component \rangle \langle component\_list \rangle \\
& \quad | \; \langle component \rangle \\
\langle component \rangle &\rightarrow COMP \langle component\_id\_range \rangle = \\
& \quad \langle component\_name \rangle \; , \\
& \quad \langle component\_details \rangle \\
\langle component\_id\_range \rangle &\rightarrow \langle component\_id \rangle , \langle component\_id\_range \rangle \\
& \quad | \; \langle component\_id \rangle \\
\langle component\_id \rangle &\rightarrow integer - integer \; | \; integer \\
\langle component\_name \rangle &\rightarrow stringconst \\
\langle component\_details \rangle &\rightarrow \langle component\_size \rangle \; ; \\
& \quad | \; \langle component\_size \rangle \; , \\
& \quad \langle num\_of\_states \rangle \; ; \; \langle state\_names \rangle \\
\langle component\_size \rangle &\rightarrow integer \; BYTE \\
\langle num\_of\_states \rangle &\rightarrow integer \\
\langle state\_names \rangle &\rightarrow \langle statename \rangle ; \langle state\_names \rangle \\
& \quad | \; \langle statename \rangle \\
\langle statename \rangle &\rightarrow integer = identifier \\
\langle trace \rangle &\rightarrow TRACE : \langle trace\_component\_list \rangle \\
\langle trace\_component\_list \rangle &\rightarrow \langle trace\_component \rangle \\
& \quad \langle trace\_component\_list \rangle \\
& \quad | \; \langle trace\_component \rangle \\
\langle trace\_component \rangle &\rightarrow \langle time\_stamp \rangle \langle statevector \rangle \\
\langle time\_stamp \rangle &\rightarrow integer \\
\langle state\_vector \rangle &\rightarrow \langle \langle integer\_list \rangle \rangle \\
\langle integer\_list \rangle &\rightarrow integer , \langle integer\_list \rangle \\
& \quad | \; integer
\end{aligned}
$$

Figure B.1: Chitra Specification Language (.csl file) grammar

Appendix C

Frequently Asked Questions

Question: I would like to know what model states were aggregated into a filter state by chitra, but I don't know an easy way to do it.

> **Answer:** To see what states constitute an aggregate state, go to the root window. Select "Options", then "Define aggregate state", then either "pattern" or "filter". A dialog box pops up asking for an ensemble instance id (0 or 1 or 2, etc., from the list box in the root window) and a model state number. Just hit "apply" to see the definition. (See §4.4.1.4 and 4.4.2.4 for further information.)

Question: CHITRA93 does not appear to use the resource values set in my .Xdefaults file in my home directory.

> **Answer:** There are several reasons why this might occur. First, the command name you use to invoke CHITRA93must be identical to the program names listed in your .Xdefaults file. For example, only the commands `chitra93`, `chitra93_alpha`, `chitra93_mips`, and `chitra93_sparc` can be used with the .Xdefaults file shown in Fig. 1.4. Second, you cannot invoke CHITRA93 using a program name containing a period, such as chitra93v4.122_alpha, even if you add the program name to the .Xdefaults file. This is because the X Windows version X11R5 does not allow a period in a program name. Third, you can only set in .Xdefaults a resource which is not set to a value via `XtSetArgs` in the CHITRA93 source code. (See §1.1 for further information.)

Question: I ran CHITRA93 and I got the Unix message saying that a write failed because /tmp was full. What caused this?

> **Answer:** CHITRA93 requires space in /tmp to hold one to two copies of each member of each ensemble. In particular, each time an ensemble is opened, one temporary file, the *display file*, is created in /tmp for each ensemble member, with a name prefixed by "cHi" (see §1.4.1.2). Each temporary file contains a binary copy of the current TSSS corresponding to the member. When a transform is applied to an ensemble for the first time, a second display file is created for each ensemble member, reflecting the result of transformation. The original display file is used to implement the "undo" function in the Transform menu. Display files are deleted when an ensemble is closed.

Question: Sometimes all the CHITRA93 windows just vanish. What caused this?

> **Answer:** The CHITRA93 source code contains assertions that are tested during execution to help detect bugs. If an assertion fails, a message is printed to stderr (thereby appearing in the window in which you invoked CHITRA93), and all windows will vanish. Please e-mail precise instructions on how to recreate the problem to chitra-bugs@cs.vt.edu.

Question: Suppose I am using the CHAID-based view and select certain states to analyze. Later on, I apply a transform. The CHAID-based view then directs me to update the view. When I do this, CHAID analyzes *all* model states, not just the ones I selected earlier. Why does CHAID do this?

> **Answer:** This situation arises when a user has a CHAID view open, then applies a transform, and the ensemble version after transform has a *different* set of states than the version before transform (which is the case whenever pattern or filter aggregation modify the ensemble). It now may not be possible for CHITRA93 to update the CHAID view, because one or more states selected for analysis no longer exist in the current ensemble version. To prevent this possibility, CHAID will automatically revert to analyzing *all* model states. The user may wish to reselect which states are to be analyzed before updating CHAID.

Question: What does the error message "Cannot convert data from .cbn files successfully" mean?

> **Answer:** The error message can be due to any of the following problems:
>
> - the user has defined an invalid ensemble (e.g., the members the user selected do not have identical state vectors).

- the display file cannot be opened
- there are no components in the .csl file
- the reader, which parses .csl files, returns NULL, which can happen for many reasons:
  - chose a .cbn file for the wrong platform
  - outdated .cbn file
  - invalid .cbn file
- there is no legal file descriptor (pst_unit->binary_fd == -1)

Appendix D

Example Trace Files Included With Chitra93

A variety of TSSS's are included with CHITRA93 in the $CHITRA/examples subdirectories listed below. The TSSS's are provides for several reasons. First, you can try out CHITRA93 on the same TSSS's that we use in our papers, M.S. theses, and M.S. project reports. Second, you may want to try the TSSS's with other visualization and modeling tools or a tool that you are developing. We found that experimenting with a variety of trace data has been essential to increasing the robustness of Chitra.

ASP: Traces of the partition node selected by an adaptive shortest path network routing algorithm for a ring network with exponential and uniform traffic arrival distributions. The routing algorithm is described by Bertsekas [5].

CadizMSProject Contains the TSSS used in Horacio Cadiz's M.S. project report to exemplify CHAID. You may wish to try CHITRA93 using this TSSS if you read the report to learn more about CHAID.

CHAID Contains the TSSS used in Chapter 5.

Diners Contains the Dining Philosophers program TSSS's used in Chapters 1 to 4, as well as for one case study in the CHITRA91 paper [1].

Drinkers Contains TSSS's of an implementation of Chandy and Lamport's Drinking Philosophers problem.

GaussianElimination Contains a trace of data address references from a Gaussian elimination algorithm.

TCP_IP Contains the TCP/IP case study from the CHITRA91 paper [1].

93cpeALCG Contains the parallel database TSSS's used in the CHITRA93 paper [3] (in EL*.csl), along with a set which were not used in the paper (TL*.csl). Refer to the README file for further information.

alpha,mips,sparc These directories exist to store machine-dependent .cbn files of .csl files in the above directories. (See §2.1 for further information.)

Appendix E

If You Want to Modify Chitra93's Source Code

If you wish to add new view, models, or transforms to Chitra, then we suggest that you request from chitra@cs.vt.edu a copy of CHITRA94. That is because there is no documentation for the CHITRA93 source code other than the comments in the code itself. In contrast, CHITRA94 is designed to be user-extensible in a simple fashion that does not required any familiarity with existing source code. In addition much of the code is written using Donald Knuth's concept of literate programming [12], using funnelweb. The source code is written as a book interspersing English narrative with source code fragments. See [4] for further information on CHITRA94.

However, if you do modify the CHITRA93 source code, then you should check you modifications by following the instructions in the *acceptance test* in file $CHITRA/tests/AcceptanceTest to insure that your modifications do not break anything in CHITRA93. The acceptance test is a collection of tests that we accumulated while writing CHITRA93. Directory $CHITRA/tests contains a variety of .csl files that are used in the AcceptanceTest.

Appendix F

Unimplemented Items and Known Bugs

1. The help buttons have not been implemented yet.
2. Dragging the elevator-box in a scrollbar in the 2D views is not implemented.
3. Do not use the "close" from the Motif border (as opposed to the chitra menu's close item). If you do this from the root window, CHITRA93 will not delete its /tmp files. If you do this from a view, CHITRA93 may not let you exit, but rather tell you that a view is still open. If you do this elsewhere, CHITRA93 will not delete the display files it creates in /tmp.
4. An extraneous diagonal line appears in 2D event view when (1) the line style is unconnected, or (2) the x-axis zoom is set to about 2% or less. Ignore the line.
5. The value of "Number of Occurrences" in the box that pops up due to menu item "Root Window: Options → Define aggregate state → Pattern" appears to reflect the number of occurrences in the highest numbered member, not the entire ensemble.
6. Problems using a 680X0 NeXT computer with Co-Xist under NEXTSTEP 3.2 as the X server:
   (a) It is necessary to hit certain keyboard accelerators twice.
   (b) The operation of drawing a bounded box may result in extraneous lines being drawn on the screen.
7. The x-axis labels in the 2D views are occassionally subject to two errors:

(a) If there are a small number of states plotted in a 2D view, then the x-axis tick marks may not line up vertically with the crosses plotted to mark the start of the states.

(b) A time or event number of "-1" may appear. To correct the graph, the window must be redrawn. Try covering and exposing the window. If the "-1" does not go away, then change the x zoom to a slightly different value (e.g., from 100 to 99).