APPENDIX **F**

An Overview of the PDP Software

In this appendix we provide a brief description of the PDP software, focusing particularly on the user interface and the data structures used by the programs. This description stands in lieu of detailed comments in the source code itself. We assume familiarity with the C programming language, as described in Kernighan and Ritchie (1978). Descriptions of the core routines that define each model are given in the appropriate chapter of the book.

Command and Variable Tables

The user interface is built around a table called the *command table*, in which each command string that the user might enter is stored. The command table contains the name of the command, the menu it is in, a pointer to a function to run if the command is entered, and a pointer to an argument to give to the command. The function *do_command* takes the command entered on the keyboard, searches through the table until it finds a match that is in the current menu, then executes the function associated with the command. If the command is the name of a menu of subcommands, then the function that is executed is *do_command* again, with its argument being the menu associated with the command. When the program is initialized, a number of commands are installed in the command tables, generally in functions whose names begin with *init_*.

There is also a variable table. Each entry in the variable table begins with a string that is used as the name for the variable. This string is used for accessing the variable from the user interface and does not need to be the same as the symbolic name used by the programmer in the program itself. Each entry also contains a variable type identifier and a pointer to the variable. The types include scalar variables (characters, integers, and floating-point numbers), strings (sequences of characters), vector variables (vectors of integers, floats, and strings) and matrix variables (here treated as lists of vectors of either integers or floats). For vector variables, there is an entry specifying the length of the vector. For matrix variables, there is one entry specifying the length of the list of vectors and another entry specifying the length of each vector. There is also a special type of variable called a weight-matrix variable, which will be described specifically later on.

Part of the initialization process involves installing variables as well as commands. Actually, when a variable is installed in the variable table, it is also installed in the command table, generally in the *set/* menu or a submenu of that menu. For example, the variable *nupdates* in the **cs** program is installed as a command in the *set/* menu. This command, when executed, calls the function *change_variable* with an argument that is a pointer to the table entry for *nupdates* in the variable table. The *change_variable* function, in turn, calls a function appropriate for the particular type of variable, as determined from the value of the type field of the entry for *nupdates*.

The Template List

In addition to the command and variable tables, there is also a template list. The template list has in it an entry for each template encountered in the *.tem* file; these entries are installed as the template file is read. Each template list entry has a string that is the external name by which the template can be assessed by the user, a specifier indicating what type of template this is (e.g., vector, matrix, label_array, etc.), a pointer to the variable associated with the template (this variable must be installed in the variable table before the template can be used), and a large number of other variables specifying the parameters of the template, including its screen location, display level, and so on. Once the templates are installed, they govern the process of displaying information in the display area, either when the *update_display* routine is called or when the user issues a command to display a particular template, as when using the *disp/* command; each template name is in fact installed in the command table in the *disp/* menu, with a pointer to the associated entry in the template list as the argument. Each template is also installed in the *disp/ opt/* menu, so that some of the parameters associated with the template can be modified by the user during running.

Matrices as Lists of Pointers to Vectors

Most of the variable types used in the programs are very standard, but two are somewhat special in the way we have implemented them. Matrices are not implemented in the standard way as two-dimensional arrays, but as lists (or vectors) of pointers to vectors. The entry *a[i][j]* refers to the *j*th entry in the *i*th vector, and, in cases where the vectors are all the same length, this notation is functionally equivalent to a conventional two-dimensional array. Weights, however, are stored in lists of pointers to vectors, in which the length of the vectors need not always be the same.

Indeed, the weight arrays are built around the idea that the vector of weights associated with each receiving unit need only be as long as the number of units that project to it.[1] To implement this, there are special arrays called *first_weight_to* and *num_weights_to*. These list the first unit that projects to each unit and the number of units that project to it. To the user, the connection to unit *i* from unit *j* is thought of as *weight[i][j]*, but internally it is stored in the element *j* − *first_weight_to[i]* of the vector *weight[i]*, and so its true index is *weight[i][j* − *first_weight_to[i]]*. We stress that from the user's point of view, a particular weight is indexed just as it would be in a full two-dimensional array.

The *first_weight_to* and *num_weights_to* arrays are determined by the % specifications found in the network specification files. When no % specification is given, the program sets *first_weight_to* to 0 for each unit and sets *num_weights_to* to *nunits*; thus the weight array in this case is functionally equivalent to a two-dimensional array. The only proviso is that the vectors of weights are not necessarily contiguous.

Extensive Use of Malloc

All of the programs can be configured for networks of arbitrary size and connectivity, limited only by the total available memory in the user's computer. Likewise, the number of variables, patterns, templates, and commands are arbitrary. This is achieved by using the function *malloc* to allocate memory for the various tables, vectors, and lists of pointers to vectors. Initially, many of these data structures are given an arbitrary, reasonable length (e.g., the command table is initially set up for 100 commands). If more entries are needed, a larger number of entries is allocated, the existing entries are copied into it, and the old list of entries is freed for later reallocation. Many of the data structures used by the program cannot be

[1] Actually, *num_weights_to* is equal to the number of units from the first unit that projects to the receiver to the last one that projects to it; there could be null weights between the first and the last, but such null weights have entries in the weight vector.

allocated until the network configuration has been defined. For this reason, the programs all have a routine called *define_system*, which is called when the user enters a command that initiates processing. Once called, a variable called *System_Defined* is set to 1, indicating that *define_system* does not need to be called again.

An Overview of the Structure of the Programs

Each program consists of a set of basic parts, each found in a separate file. The core of each program is located in a file whose name is the name of the program followed by *.c*; thus the guts of **bp** is in *bp.c*. These core files contain the routines that define the actual computations performed by the model and that compute statistics relevant to the model (e.g., *goodness*, *harmony*, or *tss*). The other parts are shared by most or all the programs. One part is concerned with the installation of commands and the routines that read commands and execute the appropriate functions as a result; it is located in the file *command.c*. Another part is concerned with the installation of templates; it is located in the file *template.c*. The file *display.c* contains the routines used to display the templates. The file *variable.c* contains functions relevant to the installation and modification of the entries in the variable list. The *weights.c* file contains routines for reading network specification files and for reading and writing weights to files. The file *patterns.c* contains routines for reading patterns and pairs of patterns into the program. The *io.c* file contains low-level (largely machine-dependent) functions for actually reading characters entered by the user and displaying characters on the screen, and *general.c* contains some low-level utility functions of general use. Finally, the file *main.c* contains the *main* function. This simply calls a number of initialization functions and then calls *do_command*, which first processes the *.str* file and then processes commands entered from the standard input by the user.