IV. Programming Organization

Throughout the project, we used Mathematica to generate and examine posets. This chapter describes the overall structure of this computational project. Because different kinds of programs are used, we will in this section explain the different kinds that arise as well as how they interact with each other and the user. In our examination of posets, we used data files to store a variety of information for inspection and comparison. The majority of the files created contain posets while the other files contain an assortment of relevant information for posets. Detailed description of the various programs and data files are provided in Chapters V and VI.

A. Terminology and Overall Organization

Certain naming conventions are used to distinguish between the ways Mathematica is implemented. Some structures we use are enclosed functions that should not be modified by anyone after creation while others need to be updated by a user for any given run. We refer to any of our programming constructions as **programs**. Each program takes one of three forms. **Subroutines** are fairly basic programs that are used solely within other programs. Our subroutines are designed to complete one task such as writing a nested list to a file or removing a minimal element from a poset. Most subroutines are used within several other programs; so their primary purpose is to prevent repetition of the same code in many places. Other subroutines fulfill a very specific purpose and were created to avoid clutter within the larger calling program.

The programs in the next category are generally longer and more complicated. We refer to these programs as **functions**. Although functions are also called by other programs, they are usually only used within one other program. Functions fulfill a specific purpose such as generating inverse extensions or testing a poset to see if it has a certain property. Most functions require subroutines or other data to be run. Therefore, we always call functions from within another function or a program of the next type. Both subroutines and functions are designed to be used without any modification after creation.

The final form a program can have is a **main program**. Main programs do not have an enclosed structure and so cannot be called by other programs. Main programs consist of lines of code to be evaluated as a Mathematica notebook. Changes must be made in a main program each time it is run. These programs provide an environment for reading in data, declaring global variables, as well as calling subroutines and functions.

In naming and referring to our programs, we use several conventions of note. In order to distinguish programs we created from the programs provided by Mathematica, the names of

our programs are misspelled. For example, the program used to generate an ideal of a poset is called Ideel[]. Besides using incorrect spellings of words, we also left off vowels in the names of programs. ThrghBldUp.nb is a good example of this convention. Furthermore we capitalized the first letter in each word making up a program name to make interpreting the name easier. For example, the program to generate all posets with a unique maximal element is BldUMxmls.nb.

The way we refer to a program also indicates its type. Because subroutines and functions are Mathematica modules, functions which are called with arguments within brackets, we refer to these programs in the following way: ProgramName[]. Therefore, Ideel[] must be either a subroutine or a function. All subroutines lie in one of two files, GenrlUtils.m or PosetUtils.m, and these files contain only subroutines. Then since Ideel[] is in PosetUtils.m, it can be identified as a subroutine. A main program is referred to by its file name. For example, the program mentioned above that generates all posets with a unique maximal element is a main program and the code for this program is in the file BldUMxmls.nb and so we refer to the program using this name.

B. Variable Usage and Reading in Data

For our subroutines and functions, information is usually passed in by arguments. Usually all information needed for a subroutine is provided from its arguments. Occasionally, however, functions require data from files or global variables. The main program in which the function is called must provide this data.

InvExtsWRI[] and JDTLRQ[] are the only two functions that require outside data. For InvExtsWRI[], we use a global variable and refer to certain equations. These equations, described in section VI.B, are stored in the files stdisos* and lookups*, for $4 \leq * \leq 7$. Therefore these files need to be read-in by any main program that calls this function, such as JDTLRscan.nb. Also InvExtsWRI[] requires that the lists of inverse extensions in invexts*, again for $4 \leq * \leq 7$, be stored in a list as the global variable invrsex[*]. Since JDTLRQ[] calls InvExtsWRI[], all of the above data also needs to be provided by any main program that calls it. In addition, JDTLRQ[] refers to another global variable. The variable szord is a list of integers that correspond to ideal sizes. It must be defined in any main program, such as JDTLRscan.nb, that calls JDTLRQ[].

Main programs are the only programs in which data from other files is read-in and global variables are defined. Because of the format of main programs, all variables within a main program are global. As mentioned above, any global variables needed by functions that a main program calls are also defined here. At the beginning of a main program, the files containing all needed subroutines and functions as well as any files containing additional data

are read-in using the Mathematica command: <<. Using this command, Mathematica can only find files in the currently set directory. Therefore users must specify the directory in which they have these files saved before the program is run.

C. User Interaction

Along with setting the directory, the user must make other changes within main programs. In this section we will mention these and other steps a user must take to obtain the results of this project or apply our programs to other problems.

First of all the user needs to obtain the files containing the programs she needs, as well as files containing any related programs or data, from our website. All main programs are saved in Mathematica notebooks while subroutines and functions are grouped into text files with .m extensions. All the program files downloaded from the website should be saved in the same directory. Data files should be saved within subdirectories of this directory. See Chapter VI for our subdirectory names and how we grouped the data files into subdirectories. The files containing subroutines and functions as well as data files should not be altered by the user. The functions and subroutines should be used from one of our main programs or from another Mathematica notebook.

To use one of the main programs, the user should open the file containing the program in Mathematica. The technical note below describes how to achieve our original spacing and layout if you are unable to open our .nb files. When a main program has been opened, the user can check the beginning comment lines to see how much he needs to change as well as which other programs will be needed. Next the user should search the main program for comment lines beginning with 'Change'. These comments indicate that the user must make a change in the next line of code. In some instances, the code is located on the same line as the comment while in others the code that is changed is below the comment. For example, suppose the user wants to find all connected posets of sizes 6-8 using Selct.nb. The first change the user must make is to set the directory in which the files have been stored. This is done within the SetDirectory[] command below the comment 'Change directory:'. If the desired test function is not in PstPropTsts.m, the user would also need to change the file containing the test. Then the user must set the variables nstrt=6 and nend=8 in this program. These changes are made to the right of the comments 'Change starting size:' and 'Change ending size:' respectively. Next, the user must specify an input subdirectory and file family such as Stdpsts/stdpsts. This is done to the right of the comment 'Change input family:'. Below this and to the right of the comment 'Change output family:' the user must specify a subdirectory and name for the file that will contain the connected posets, such as Connets/connets. The final modification the user must make is to change the property test to

ConnctdQ[] within a WrtLstOfLst[] call. This change is made three lines below the comment 'Change poset property test:'. When the user has located and updated the code in the necessary places, the main program is ready to be run. The user may then hit 'Shift+Enter' to run the program.

In order to view the data generated by a main program, the user will usually need to take an additional step after running the program. The only two programs that print their results to the screen are the data file inspection and comparison main programs. While the results from the other main programs are not displayed immediately, the user can access this information using Inspct.nb. In Inspct.nb, the user can view some lines of a file generated by one of the other main programs and check its length. A user can also assess the contents of a poset list file by comparing it to another such file using the program Compar.nb. The user may also view the contents of any data file generated by our main programs using a text application or the command: !!*filename* in Mathematica. For some of our main programs, time and memory considerations are an issue for larger poset sizes. Therefore in the poset generation programs and in JDTLRscan.nb, the maximum consumption of RAM, the time used by the CPU, and the amount of actual time that a run took are displayed on the screen.

D. Technical Notes

Aside from the information above, there are several other issues that arose during our project of which the user should be aware.

- **Creating Subdirectories:** Each program that generates data files creates these files in subdirectories. Before running a main program for the first time, the user must create the associated data subdirectory folders by hand.
- **Executable Attribute:** When Mathematica is used on an XP computer to create data files in a Unix file system, the new data files have the executable attribute. This can be removed by hand without adversely affecting the readability of these files for future input.
- **Reformatting/Running Program Files:** As noted above, our subroutines and functions are being saved in 'text'(.m) files before being posted on the website. Also, we provide .m versions of our main program files for those unable to use our notebooks. When these files are downloaded and opened within Mathematica, the tabs will be set to the default of 4 spaces instead of our convention of 3 spaces. Also all lines will be within one initialization input cell. To view the code in the main programs in the most readable way, the user should follow the following steps. Select the right cell bracket in the .m file and then select 'Copy'. Next open an empty template notebook (as described in Appendix C) and then select 'Paste'. After this, open the Format menu and select 'Remove Options'. After choosing 'OK', the cell in the template notebook is no longer an initialization cell. Next open the 'Option Inspector' from the format menu. Under Formatting Options/Text Layout Options set 'PageWidth' to Infinity. This notebook then has close to our original spacing and layout. The user can save this or run the program from the template after making any necessary changes.
- **Setting the Directory:** In the final versions of each of our main programs, the directory is set using the command: SetDirectory[" G:/.isis/home/c/a/cagann/public_html/Posets"]. This of course must be updated to reflect the path to the directory in which the user saves the package files. For the Windows XP machine in Proctor's office, the command must be changed to SetDirectory[" H:/public_html/Posets"].

E. Table 4.1: Program Interaction and Storage

The following table displays all the programs used in this project. We have grouped the programs according to their use. For each subroutine and function, we give the programs that use it. We also indicate the file in which every program is stored. Since the three auxiliary files described in Chapter VI play an important role in the project, they are included in this table as well.

Please see Table 8.1 in Chapter VIII for the analogous information for the hook length poset computation programs.

	Program	Used By	Stored In
General Subroutines	CmprssBiList[]	BasInvExts[], InvExtsWRI[], StdFormIso[], ThrghBldUp.nb	GenrlUtils.m
	NoSpcs[], WriteLstOfLst[]	Most main programs.	GenrlUtils.m
	PrntCnsmptn[]	Poset Generation programs, JDTLRscan.nb	GnerlUtils.m
Poset	Ideel[]	AntiChnsIdls[], JDTLRQ[]	PosetUtils.m
Subroutines	AntiChnsIdls[]	InvExtsWRI[], ThrghBldUp.nb, QckBldUp.nb, JDTLRQ[]	PosetUtils.m
	PllOff[]	BasInvExts[], StdFormIso[]	PosetUtils.m
	Relabl[]	InvExtsWRI[], ThrghBldUp.nb	PosetUtils.m
	NewChdLst[]	StdFormIso[]	PosetUtils.m
	ShrnkCnvxSt[]	InvStdTblx[]	PosetUtils.m
	InvStdTblx[]	InvExtsWRI[], JDTLRQ[]	PosetUtils.m
Inverse	BasInvExts[]	ThrghBldUp.nb	InvExIsoFncts.m
Extension and	InvExtsWRI[]	JDTLRQ[]	InvExIsoFncts.m
Isomorphism Functions	StdFormIso[]	QckBldUp.nb, Compar.nb	InvExIsoFncts.m
Poset Generation	ThrghBldUp.nb		ThrghBldUp.nb
Main Programs	QckBldUp.nb		QckBldUp.nb
	BldUMxmls.nb		BldUMxmls.nb
Routine Poset	ConnctdQ[]	Selct.nb	PstPropTsts.m
Property Test	UniqMaxmlQ[]	Selct.nb	PstPropTsts.m
Functions	dCompltQ[]	Selct.nb	PstPropTsts.m
File Inspection	Inspct.nb		Inspct.nb
And Comparison	Compar.nb		Compar.nb
Main Programs			
Data File	Selct.nb		Selct.nb
Creation Main	Intersct.nb		Intersct.nb
Programs	Complemnt.nb		Complemnt.nb
JDT & L-R Test Function and	JDTLRQ[]	JDTLRscan.nb	JDTLRQ.m
Main Program	JDTLRscan.nb		JDTLRscan.nb
Auxiliary	invexts*	InvExtsWRI[], JDTLRQ[]	
Data Files	stdisos*	InvExtsWRI[], JDTLRQ[]	
	lookups*	InvExtsWRI[], JDTLRQ[]	