Chapter 1

Files and directories

There are six programs that we use to parse and translate definitions, and gather and view data on them:

1. `lptparse`
2. `Lpt2dzfc`
3. `SymbolCounter`
4. `Qcounter`
5. `DagMaker`
6. `GraphViewer`

These programs, and their supporting files, come in a directory structure that has a `main` directory with the following subdirectories:

1. `dags`
2. `data`
3. `defs`
4. `dFiles`
5. `expand`
6. `expandDG`
7. `graphs`
8. `lpt2dzfc`
9. `lptparse`
10. `tex`
11. `util`

Chapter 2

Using the software

Start by typing definitions in LPT, and saving them in a file in the directory `lptparse/lptfiles`, with extension `.lpt`. You will find in the directory `lptparse/lptfiles` that I have provided as examples all the `.lpt` files that I used in my project.

2.1 Parsing and translation

We'll consider how to process the file `MunkTop12.lpt`, which contains the definitions from Munkres Topology section 12, written in LPT.

We begin by typing

```
./partran MunkTop12
```

at the UNIX command prompt. The file `partran` is a shell script that applies both the parser `lptparse`, and the translator `lpt2dzfc`, one after the other (it parses and translates, hence "partran"). The results are as follows:

1. A bracketed (i.e. parsed) version of `MunkTop12.lpt`, is saved as

   ```
   lptparse/lptoutput/MunkTop12.br
   ```

   where the file extension `.br` stands for "bracketed".

2. The translation into DZFC of each of the definitions in `MunkTop12.lpt` is computed, and the results are saved in the `.xml` files in the `defs` directory. Also, entries are added to the `0index.xml` file in the `main` directory. These files are not designed for viewing (although they can be viewed using a web browser); they are meant to be used by the other programs in the suite.

3. A file named `MunkTop12.tex` is stored in the `tex` directory. It contains, for each definition in `MunkTop12.lpt`,

   (a) the LaTeX version of the definition as written in LPT, and

   (b) the LaTeX version of the definition as translated into DZFC.

4. A file named `MunkTop12.d` is stored in the `dFiles` directory. A `.d` file is used as a "batch list" for all the data-gathering programs, which we will go over in Section 2.2.

Alternatively, you may use `lptparse` and `Lpt2dzfc` individually.

2.1.1 lptparse

To run `lptparse` on a file in the directory `lptparse/lptfiles`, say the file `MunkTop12.lpt` for example, begin by entering the `lptparse` directory, with

```
cd lptparse
```

Then type

```
./lptparse < lptfiles/MunkTop12.lpt
```

This results in the creation of a file called `lptout`, in the `lptparse` directory. It contains the bracketed version of the input.

If you wish, you may then type

```
cp lptout lptoutput/MunkTop12.br
```

in order to save this output in a `.br` file in the `lptoutput` directory, as `partran` does for you automatically. For the purposes of further illustration, let's suppose you choose to do this. (Note that if you do not do this, then your data in the file `lptout` will be overwritten the next time you run `lptparse`.)

In any case, return to the `main` directory by typing

```
cd ..
```

2.1.2 Lpt2dzfc

By running `Lpt2dzfc` manually you can take advantage of a few command line options that are not available to you if you simply use `partran`.

The command line syntax is:

```
java lpt2dzfc/Lpt2dzfc (-t|-p|-s|-o) brFile
```

Here, `brFile` is your file containing bracketed LPT definitions. You have your choice of the switches `-t`, `-p`, `-s`, and `-o`.

Use `-s` in order to save your definitions in the `.xml` files in the `defs` directory, and make changes to the `0index.xml` file in the `main` directory, but *skipping* any duplicate definitions. This means that, for example, if you type

```
java lpt2dzfc/Lpt2dzfc -s algebra1.br
```

and if `algebra1.br` contains a definition of the relation `GROUP` with arity 3, and if there already exists a definition of `GROUP` with arity 3, then your definition will not be recorded.

Use `-o` in order to achieve the same thing as with the `-s` switch, only *overwriting* any duplicate definitions. So in the example of `GROUP` with arity 3, your new definition overwrites the old one.

Use `-p` to perform the translation of each definition into DZFC, but to simply print each translation to the screen, and make no alterations to the files in the `defs` directory, or to the `0index.xml` file.

As with the `-p` switch, the `-t` switch alters no files. It simply prints to the screen the parse tree for each bracketed definition in the passed input file.

2.2 Counting symbols and quantifiers in expanded definitions

The data-gathering programs are designed to be applied to `.d` files, and thereby process all the definitions named in them, at one go. They also take command line parameters that make it easier to gather the data from separate `.d` files in one place.

2.2.1 SymbolCounter

Consider first the `SymbolCounter` program. It counts the length of definitions, taken at various states of expansion, as was done in my thesis project.

For example, we may begin by typing

```
java expandDG/SymbolCounter -b dFiles/MunkTop12.d -l expandDG/leaves -d data/SymbCounts
```

at the command prompt.

The

```
-b dFiles/MunkTop12.d
```

field tells SymbolCounter to use `dFiles/MunkTop12.d` as the "batch" file. (The `-b` switch stands for "batch".) This means that SymbolCounter will process each definition named in `dFiles/MunkTop12.d` in turn.

The

```
-l expandDG/leaves
```

field tells SymbolCounter to use `expandDG/leaves` as the "leaves" file. This is the set of defined concepts that will *not* be expanded whenever they may be encountered.

The

```
-d data/SymbCounts
```

field tells SymbolCounter to store the data that it computes in the file `data/SymbCounts`.

Output:

1. A file named `MunkTop12_SymbolData.tex` is saved in the `tex` directory, containing a formatted table presenting the data gathered on the definitions named in `MunkTop12.d`. For each definition, we get the number of symbols in:
   (a) its translation into DZFC,
   (b) the full expansion thereof, and
   (c) the partial expansion thereof, relative to the leaves named in `expandDG/leaves`.
2. In the file `data/SymbCounts` named in the command line we get the same data as is presented in the `tex/MunkTop12_SymbolData.tex` file, only in a simple comma-delimited format that can be read by a spreadsheet program such as Microsoft Excel.

Consider now what happens with a subsequent run of `SymbolCounter`. If we type

```
java expandDG/SymbolCounter -b dFiles/MunkTop13.d -l expandDG/leaves -d data/SymbCounts
```

at the command prompt (note that this time we're working on the next file in our Topology series, `dFiles/MunkTop13.d`), then, since we used the same

```
-d data/SymbCounts
```

field, the comma-delimited data this time will be *appended* to the file `data/SymbCounts`. This way we can gather all the data from, say, our topology definitions, in one file, and all our algebra data in another file, etc.

Options:

The `-l` and `-d` fields are optional; we may omit either or both of them. If the leaves field is omitted then the second and third columns in the data tables will always be the same, since partial expansion with respect to an empty set of leaves is the same thing as total expansion.

If the data field is omitted then the formatted table will still be stored in `MunkTop12_SymbolData.tex`. But the simple comma-delimited data will not be stored anywhere.

There is one more switch, the `-c` switch, that we may use in the command line. It goes after the `-b` field and before the `-l` or `-d` fields. Thus, we may type

```
java expandDG/SymbolCounter -b dFiles/MunkTop12.d -c -l expandDG/leaves
```

for example.

The `-c` switch stands for "clear" and its purpose is to clear certain internal data used by the `SymbolCounter` algorithm. As I explain in my thesis paper, the algorithm that I use for computing the lengths of expanded definitions works by storing, for each definition, a representation of a linear function, which gives the length of the expanded definition as a function of the lengths of the arguments plugged in for its free variables.

Each time you run `SymbolCounter` on a new `.d` file, the new linear functions are added to a running list. There are two things to understand about this list: (1) If you want to expand a definition D that depends on a prior definition C, then you need to run `SymbolCounter` on C before you can run it on D, so that the linear function for C will be available when it is needed. (2) If you have already run `SymbolCounter` on a definition B and then later make a change to B, then you need to change the linear function stored for B, or else subsequent data will be inaccurate.

This is where the `-c` switch comes in. If you add it to the command line, then *all* prior stored linear functions will be cleared from memory, and you will start fresh.

2.2.2 Qcounter

The `Qcounter` program takes the same command line parameters as does `SymbolCounter`, but instead of counting symbols it counts quantifier depth, both alternating and non-alternating.

An example command line would be:

```
java expandDG/Qcounter -b dFiles/MunkTop12.d -l expandDG/leaves -d data/QCounts
```

The only difference between `Qcounter` and `SymbolCounter` then, is the output. Like `SymbolCounter`, `Qcounter` yields two output files: one giving a formatted LaTeX table; the other giving a simple comma-delimited presentation of the same data, saved in the file named in the `-d` field in the command line.

As for the formatted output, it is saved (continuing with our example) in the `tex` directory, in a file named `MunkTop12_QData.tex`. The columns of the table give the maximum quantifier nesting depth occurring in the definitions in `MunkTop12.d` in the following ways:

1. in the original LPT definition
2. in the translation into DZFC
3. in the total expansion thereof
4. in the partial expansion thereof, with respect to the leaves named in the command line
5. in the original LPT definition, but only counting alternations
6. in the translation into DZFC, but only counting alternations
7. in the total expansion thereof, but only counting alternations
8. in the partial expansion thereof, with respect to the leaves named in the command line; but only counting alternations.

Note that the `-c` switch for `Qcounter` clears just its internal functions, not those of `SymbolCounter`.

2.3 DAGs

2.3.1 DagMaker

The `DagMaker` program takes a simpler command line than the programs discussed in the last section. Like those, it takes a `-b` field, giving the batch file; and a `-d` field naming a file in which to store the computed data. But it has no `-l` field, and there is no `-c` switch.

An example command line is:

```
java dags/DagMaker -b dFiles/MunkTop12.d -d data/DagData
```

The `-d` field works just as it does with the `SymbolCounter` and `Qcounter` programs. If the file named there does not yet exist, it is created, and the data computed are stored there; if it does already exist then the data computed are appended there.

Output:

1. A file named `MunkTop12_DAG.tex` is saved in the `tex` directory, containing, for each definition named in `MunkTop12.d`, LaTeX code which will plot the DAG for this definition, using the `dcpic` LaTeX package (which I have included in the `tex` directory, for convenience).

2. A file named `MunkTop12_DAGdata.tex` is also saved in the `tex` directory, containing a formatted table listing the *depth* and *size* of each DAG plotted by the previous file.

3. According to the `-d` field in the command line, a file named `DagData` is saved in the `data` directory, with a simple comma-delimited version of the data presented in the formatted table in the previous file.

Options:
The `-d` field is optional. If it is omitted, then only the first two output files will be created.

2.3.2 GraphViewer

The final program gives a graphical user interface with which to view visual representations of the DAGs for the definitions that have been entered into the database using `lptparse` and `lpt2dzfc` (or just using `partran`).

To run it, type

```
java graphs/GraphViewer kmap
```

at the command prompt.

A window opens. In the textbox at the top, type the name of a defined concept, in the same format in which they appear in `.d` files. This consists of the name of the defined concept, a colon, and the arity of the defined concept. For example, you might type `GROUP:3`, or `TOPOLOGY:2`, or `Galoisgp:2`.

Hit enter, and you are asked whether or not to begin a new graph. Say yes. Now you may move the nodes of the graph by left-clicking on them and dragging. And you may expand and collapse the DAG by right-clicking on the nodes appearing in it, and selecting the appropriate option.