Author instructions

Daniel R. Grayson*

The abstract should summarize the contents of the paper using at least 70 and at most 150 words. It will be set in 9-point font size and be inset 1.0 cm from the right and left margins. There will be two blank lines before and after the Abstract.

Introduction

Run TeX on the files `lncse-doc.tex` and `lncse-dem.tex` in the directory `lncse-macros` to produce the documentation and a demonstration of the macro package `lncse` provided to us by Springer. It's almost compatible with AMS-Latex and Latex-2e, but multiple authors are handled with a single `\author` macro, the macros `\email` and `\urladdr` are missing, the macro `\address` is replaced by `\institute`, and the macro `\thanks` should be inserted where the corresponding footnote indicator is needed. Another difference is that the command `\theoremstyle` does not exist, and the command `\newtheorem` has been modified to take a few extra arguments.

To create your own chapter, copy the entire directory `TEMPLATE` to create a directory for your own chapter, and edit the files there. For example, the command

```
cp -a TEMPLATE foo
```

will copy the files to a new directory named `foo`. Then you should also add a line

```
\getChapter{foo/chapter}
```

to the file `book/book.tex`, and a pair of dependency lines to the top of `book/Makefile`.

The file `foo/chapter.tex` is the one you where you should insert your text, after deleting the TeX code for these paragraphs. It is not a complete TeX file, because it gets eventually input into TeX by the file `book/book.tex` or the file `foo/chapter-wrapper.tex`. It is intentionally missing the

```
\documentclass{lncse}
```

and

```
\begin{document}
```

macros at the top, and the

```
\bibliography{papers}
```

* Supported by the NSF

and

```
\end{document}
```

macros at the bottom, so don't add them; these things have to be done in a special way when the chapter is incorporated into the book.

If you want to change the name of the file `foo/chapter.tex`, you should edit `book/Makefile`, `book/book.tex`, the top line of `foo/Makefile`, and in the value of `tex-main-file` at the bottom of `foo/chapter.tex`.

We include a file

`inputs/book-macros.tex`,

which authors should not modify, which provides macros and commands which can be used throughout the book. It loads the `amsmath`, `amscd`, `amssymb`, `latexsym` packages.

We can produce an entry in the index for the whole book using the macro `\index`, illustrated here with

`\index{indexing}`.

Use the macro

```
\Mtwo
```

to incorporate the name *Macaulay 2* into your text. This ensures the name will be uniformly italicized. Spaces after the macro will be obeyed, and an italic correction will be inserted.

References to the literature [1] are created as usual with `\cite`. Make sure you put the corresponding bibtex entries, obtainable from MathSciNet, in the file `papers.bib` in the same directory as your chapter. Bibtex entries that duplicate those in other chapters will cause harmless warning messages.

The file `Makefile` contains the information needed for GNU make to run TeX and various auxiliary programs in the correct sequence. To make sure you have GNU make installed, execute the command `make -v`. GNU make will identify itself with a first line like this:

```
GNU Make version 3.77, by Richard Stallman and Roland McGrath.
```

You can type `make` in your chapter's directory to process just your chapter, or you can type `make` in the top level directory to create the whole book. It runs TeX in scroll mode, so you can get many error messages at once; you can change that if you'd like it to be more interactive.

If `make` runs successfully, your chapter can be viewed in the file

`foo/chapter-wrapper.dvi`,

and the book can be viewed in the file

`book/book.dvi`.

1 Incorporating *Macaulay 2* code in your chapter.

To incorporate *Macaulay 2* code into your text, surround each statement to be executed with `<<<` and `>>>`, and the result will be automatically run through *Macaulay 2* for you.

```
i1 : A = QQ[x,y]

o1 = A

o1 : PolynomialRing
```

For example, the code above was inserted by the following line.

```
<<<A = QQ[x,y]>>>
```

A single statement can span multiple lines, but try to keep it short.

```
i2 : N = cokernel random (
             A^1,
             A^{-2,-2}
             )

o2 = cokernel | 5/2xy+y2 -8/9xy-2/5y2 |

                             1
o2 : A-module, quotient of A

i3 : res N

      1      2      1
o3 = A  <-- A  <-- A  <-- 0

     0      1      2      3

o3 : ChainComplex
```

If the output from *Macaulay 2* is too wide, dots will appear indicating where it was truncated.

```
i4 : 100!

o4 = 93326215443944152681699238856266700490715968264381621468592963895 ···
```

Make sure you always have the most recent version of *Macaulay 2* installed, for you will be running other authors' code through it.

2 Running *Macaulay 2* in emacs

Because some answers can be very wide, it is a good idea to run *Macaulay 2* in a window which does not wrap output lines and allows the user to scroll horizontally to see the rest of the output. We provide a package for `emacs` which implements this, in `emacs/M2.el` (the directory `emacs` is in the *Macaulay 2* distribution). It also provides for dynamic completion of symbols in the language.

There is an ASCII version of this section of the documentation distributed in the file `emacs/emacs.hlp`. It might be useful for you to visit that file with emacs now, thereby avoiding having to cut and paste bits of text into emacs buffers for the demonstrations below.

If you are a newcomer to emacs, start up emacs with the command `emacs` and then start up the emacs tutorial with the keystrokes `C-H t`. (The notation `C-H` indicates that you should type `Control-H`, by holding down the control key, and pressing `H`.) The emacs tutorial will introduce you to the basic keystrokes useful with emacs. After running through that you will want to examine the online emacs manual which can be read with `info` mode; you may enter or re-enter that mode with the keystrokes `C-H i`. You may also want to purchase (or print out) the emacs manual. It is cheap, comprehensive and informative. Once you have spent an hour with the emacs tutorial and manual, come back and continue from this point.

Edit your `.emacs` initialization file, located in your home directory, creating one if necessary. (Under Windows, this file is called `_emacs`.) Insert into it the following lines of emacs-lisp code.

```
(setq auto-mode-alist (append auto-mode-alist '(("\\.m2$" . M2-mode))))
(autoload 'M2-mode "M2-mode.el" "Macaulay 2 editing mode" t)
(global-set-key "\^Cm" 'M2) (global-set-key [ f12 ] 'M2)
(global-set-key "\^Cm" 'M2) (global-set-key [ SunF37 ] 'M2)
(autoload 'M2 "M2.el" "Run Macaulay 2 in a buffer." t)
(setq load-path (cons "/usr/local/Macaulay2/emacs" load-path))
(make-variable-buffer-local 'transient-mark-mode)
(add-hook 'M2-mode-hook '(lambda () (setq transient-mark-mode t)))
(add-hook 'comint-M2-hook '(lambda () (setq transient-mark-mode t)))
```

The first two lines cause emacs to enter a special mode for editing *Macaulay 2* code whenever a file whose name has the form `*.m2` is encountered. The next three lines provide a special mode for running *Macaulay 2* in an emacs buffer. The sixth line tells emacs where to find the emacs-lisp files provided in the *Macaulay 2* emacs directory - you must edit the string in that line to indicate the correct path on your system to the *Macaulay 2* emacs directory. The files needed from that directory are `M2-mode.el`, `M2-symbols.el`, and `M2.el`. The seventh line sets the variable `transient-mark-mode` so that it can have a different value in each buffer. The eighth and ninth lines set hooks so that `transient-mark-mode` will be set to `t` in M2 buffers. The effect of this is that the mark is only active occasionally, and then emacs functions which act on a region of text will refuse to proceed unless the mark is active. The `set-mark` function or the `exchange-point-and-mark` function will activate the mark, and it will remain active until some change occurs to the buffer. The only reason we recommend the use of this mode is so the same key can be used to evaluate a line or a region of code, depending on whether the region is active.

Exit and restart emacs with your new initialization file. If you are reading this file with emacs, then use the keystrokes `C-x 2` to divide the buffer containing this file into two windows. Then press the `F12` function key to start up *Macaulay 2* in a buffer named `*M2*`.

If this doesn't start up *Macaulay 2*, one reason may be that your function keys are not operable. In that case press `C-C m` instead. (The notation `C-C` is standard emacs notation for Control-C.) Another reason may be that you have not installed *Macaulay 2* properly - the startup script (`M2` or `M2.bat`) should be on your path. A third reason may be that you are in Windows-98 and are using anti-virus software such as `Dr. Solomon's`, which can interfere with emacs when it tries to run a subprocess.

You may use `C-x o` freely to switch from one window to the other. Verify that *Macaulay 2* is running by entering a command such as `2+2`. Now paste the following text into a buffer, unless you have the ASCII version of this documentation in an emacs buffer already, position the cursor on the first line of code, and press the `F11` function key (or `C-C s`) repeatedly to present each line to *Macaulay 2*.

```
i1 = R = ZZ/101[x,y,z]
i2 = f = symmetricPower(2,vars R)
i3 = M = cokernel f
i4 = C = resolution M
i5 = betti C
```

Notice that the input prompts are not submitted to *Macaulay 2*.

Here is a way to conduct a demo of *Macaulay 2* in which the code to be submitted is not visible on the screen. Paste the following text into an emacs buffer.

```
20!
4 + 5 2^20
-- that's all folks!
```

Press `M-F11` with your cursor in this buffer to designate it as the source for the *Macaulay 2* commands. (The notation `M-F11` means that while holding the `Meta` key down, you should press the `F11` key. The Meta key is the Alt key on some keyboards, or it can be simulated by pressing Escape (just once) and following that with the key you wanted to press while the meta key was held down.) Then position your cursor (and thus the emacs point) within the line containing `20!`. Now press `M-F12` to open up a new frame called `DEMO` for the `*M2*` window with a large font suitable for use with a projector, and with your cursor in that frame, press `F11` a few times to conduct the demo. (If the font or frame is the wrong size, you may have to create a copy of the file `M2.el` with a version of the function `M2-demo` modified to fit your screen.)

One press of `F11` brings the next line of code forward into the `*M2*` buffer, and the next press executes it. Use `C-x 5 0` when you want the demo frame to go away.

There is a way to send a region of text to *Macaulay 2*: simply select a region of text, making sure the mark is active (as described above) and press `F11`. Try that on the list below; put it into an emacs buffer, move your cursor to the start of the list, press `M-C-@` or `M-C-space` to mark the list, and then press `F11` to send it to *Macaulay 2*. (The notation `M-C-@` means: while holding down the Meta key and the Control key press the `@` key, for which you'll also need the shift key.)

```
{a,b,c,d,e,f,
g,h,i,j,k,l,
m,n}
```

We have developed a system for incorporating *Macaulay 2* interactions into TeX files. Here is an example of how that looks. Paste the following text, representing something you might see in a TeX file, into an emacs buffer.

```
The answer, 4, is displayed after the output label ``{\tt o1\ =}''.
Multiplication is indicated with the traditional {\tt *}.
<<<1*2*3*4>>>
Powers are obtained as follows.
<<<2^100>>>
```

The bits in brackets can be submitted to *Macaulay 2* easily. Position your cursor at the top of the buffer and press `F10`. The cursor will move just past the first `<<<`, and the emacs mark will be positioned just before the `>>>`. Thus `1*2*3*4` is the region, and it will even be highlighted if you have set the emacs variable `transient-mark-mode` to `t` for this buffer. Pressing `F11` will send `1*2*3*4` to *Macaulay 2* for execution: try it now. A sequence of such *Macaulay 2* commands can be executed by alternately pressing `F10` and `F11`. You may also use `M-F10` to move backward to the previous bracketed expression.

Now let's see how we can handle wide and tall *Macaulay 2* output. Execute the following line of code.

```
random(R^20,R^{6:-2})
```

Notice that the long lines in the *Macaulay 2* window, instead of being wrapped around to the next line, simply disappear off the right side of the screen, as indicated by the dollar signs in the rightmost column. Switch to the other window and practice scrolling up and down with `M-v` and `C-v`, and scrolling left and right with the function key `F3` (or `C-C <`) and the function key `F4` (or `C-C >`). Notice how the use of `C-E` to go to the end of the line sends the cursor to the dollar sign at the right hand side of the screen; that's where the cursor will appear whenever you go to a position off the screen to the right. Then use the `F2` function key (or `C-C .`) to scroll the text so the cursor appears at the center of the screen. Use `C-A` to move to the beginning of the line and then the `F2` function key (or `C-C .`) to bring the left margin back into view.

You may use the `F5` function key or (or `C-C ?`) to toggle whether long lines are truncated or wrapped; initially they are truncated.

Now go to the very end of the `*M2*` buffer with `M->` and experiment with keyword completion. Type `reso` and then press the `TAB` key. Notice how the word is completed to `resolution` for you. Delete the word with `M-DEL`, type `res` and then press the `TAB` key. The possible completions are displayed in a window. Switch to it with the `F8` key, move to the desired completion, select it with the `RETURN` key, and then return to the `*M2*` buffer with `C-X o`. Alternatively, if you have a mouse, use the middle button to select the desired completion.

Experiment with command line history in the `*M2*` buffer. Position your cursor at the end of the buffer, and then use `M-p` and `M-n` to move to the previous and next line of input remembered in the history. When you get to one you'd like to run again, simply press return to do so. Or edit it slightly to change it before pressing return.

3 Editing *Macaulay 2* code with emacs

In this section we learn how to use emacs to edit *Macaulay 2* code. Assuming you have set up your emacs init file as described in `running Macaulay 2 in emacs` when you visit a file whose name ends with `.m2` you will see on the mode line the name `Macaulay 2` in parentheses, indicating that the file is being edited in *Macaulay 2* mode. (Make sure that the file `emacs/M2-mode.el` is on your `load-path`.)

To see how electric parentheses, electric semicolons, and indentation work, move to a blank line of this file and type the following text.

```
f = () -> (
     a := 4;
     b := {6,7};
     a+b)
```

Observe carefully how matching left parentheses are indicated briefly when a right parenthesis is typed.

Now position your cursor in between the 6 and 7. Notice how pressing `M-C-u` moves you up out of the list to its left. Do it again. Experiment with `M-C-f` and `M-C-b` to move forward and back over complete parenthesized expressions. (In the emacs manual a complete parenthesized expression is referred to as an sexp, which is an abbreviation for S-expression.) Try out `C-U 2 M-C-@` as a way of marking the next two complete parenthesized expression, and see how to use `C-W` to kill them and `C-Y` to yank them back. Experiment with `M-C-K` to kill the next complete parenthesized expression.

Position your cursor on the 4 and observe how `M-;` will start a comment for you with two hyphens, and position the cursor at the point where commentary may be entered.

Type `res` somewhere and then press `C-C TAB` to bring up the possible completions of the word to documented *Macaulay 2* symbols.

Finally, notice how `C-H m` will display the keystrokes peculiar to the mode in a help window.

The text below is for testing purposes:

a: `<----- 3.33333pt plus 1.66666pt minus 1.11111pt`

a. `<----- 3.33333pt plus 1.66666pt minus 1.11111pt`

a? `<----- 3.33333pt plus 1.66666pt minus 1.11111pt`

a; `<----- 3.33333pt plus 1.66666pt minus 1.11111pt`

References

1. Goro Shimura: *Introduction to the arithmetic theory of automorphic functions.* Publications of the Mathematical Society of Japan, No. 11. Iwanami Shoten, Publishers, Tokyo, 1971. Kanô Memorial Lectures, No. 1.

Index