Guidelines for Authors

by Shae Matijs Erisson ⟨shae@scannedinavian.com⟩
and Andres Löh ⟨kstmr@andres-loeh.de⟩
and Wouter Swierstra ⟨wss@cs.nott.ac.uk⟩
and Brent Yorgey ⟨byorgey@cis.upenn.edu⟩

March 23, 2010

This text, written in the style of a typical The Monad.Reader *article, gives guidelines and advice for* The Monad.Reader *authors. We explain the TEXnical constructs needed to write an article, and give a few hints on how to improve style.*

Getting started

If you want to write an article for The Monad.Reader [1], you need a special LaTeX class file called `tmr.cls`. Currently, the best way to retrieve the latest version is to say

```
darcs get http://code.haskell.org/~byorgey/TMR/Guidelines
```

assuming you have `darcs` [2] installed. If you do not use darcs, you can also download a zipfile containing the same files from

```
http://code.haskell.org/~byorgey/TMR/TMR.zip
```

Place the file in a directory where your TeX installation can find it. If you do not know how to do this, you can put it in the same directory where the sources of your article reside – this should always work.

Then, you have to instruct LaTeX to use the class file, by using

```
\documentclass{tmr}
```

as the first line of your document. There is no need to specify any class options (such as options affecting the default font or paper size), because `tmr.cls` automatically sets everything according to the defaults for The Monad.Reader.

The zip file and darcs repository contain several other useful files. The file `Author.tex` contains the source code that generated this file. The other files, `tmr.bst` and `tmr.dbj`, are needed to format the bibliography. Make sure your TEX installation can locate these files as well.

The title page

Each article starts with meta-information about the title and the authors. The title is set using the `\title` command.

- Capitalize all relevant words in a title.

The authors are given by one or more `\author` commands.

- Include e-mail addresses of the authors using the `\email` command.
- Put no space between the author and the `\email` command.
- Include no further information about the authors.

All of these commands should appear in the **preamble** of the document, i.e., before the `\begin{document}` command. The header of this document is shown in Figure 1.

```
\title{\TMR\ Guidelines for Authors}
\author{Shae Matijs Erisson\email{shae@scannedinavian.com}}
\author{Andres L\"oh\email{loeh@iai.uni-bonn.de}}
\author{Wouter Swierstra\email{wss@cs.nott.ac.uk}}
```

Figure 1: Example header with title and author information

Introduction / abstract

Abstracts are common practice for academic papers, to give an overview of the contents of a paper. The Monad.Reader is not an academic journal, so a more casual introduction is in place. For this purpose, there is an `introduction` environment available.

- Keep the introduction short. The first section of the article should always start on the first page.
- Use as little markup in the introduction as possible. Avoid itemizations or enumerations.

The introduction is printed in italics, and belongs between the header and the first section title.

As an example, we show the introduction of this document in Figure 2.

```
\begin{introduction}
This text, written in the style of a typical \TMR\ article, gives
guidelines and advice for \TMR\ authors. We explain the \TeX nical
constructs needed to write an article, and give a few hints on how
to improve style.
\end{introduction}
```

Figure 2: Example introduction

Structuring the article

Sections

An article is structured in sections and subsections, created by the commands `\section` and `\subsection` respectively. Avoid more levels of structuring, and don't create your own sectioning constructs by starting paragraphs with emphasized words. Avoid math or code in sections. If you must, use the same fonts as elsewhere.

Both sections and subsections do not have numbers. If you'd like to refer to a section, try to refer to it by name and page, using `\pageref`.

If you write a very long article and really need numbers, you can always make them part of the name of a section, such as "Explanation – Part 1".

Paragraphs

You can start a new paragraph using a blank line or by saying `\par`. Never use `\\` to start a new line in ordinary paragraphs. Avoid using manually inserted vertical spaces.

Enumerations

Avoid extreme use of enumerations. In particular, try to avoid nested enumerations. The default of the The Monad.Reader class is to display enumerations rather tight, like this

- foo

- ▶ bar

If you have very long items, possibly even including paragraph breaks, use the `longitem` or `longenum` environments.

1. This is an example of the `longenum` environment. It has some vertical space to separate it from the surrounding text . . .

2. . . . and the different items are also separated by vertical space.

Blocks (theorems, examples, exercises etc.)

A good way to structure results is to use labelled blocks. Theorems, definitions, examples, exercises, are all examples of this. The class file provides a large number of predefined environments for this purpose.

The `theorem`, `lemma`, and `corollary` environments create blocks in the following style:

Theorem 1. *This is an important theorem.*

Any block can have an optional argument, which will be used as its name.

Corollary 2 (Adams). *The answer is 42.*

Remark. The `remark` block is predefined and looks like this. All other blocks are numbered, and use the same counter. Predefined blocks with a counter are `definition`, `example`, and `exercise`. They appear as follows:

Exercise 3. New sorts of blocks can be defined using the `\theoremstyle` and `\newtheorem` commands. The style can be either `plain` (as for `theorem`, `lemma` etc.) or `definition` (as for `definition`, `example` etc.). The `exercise` environment can be defined using the code in Figure 3.

```
\theoremstyle{definition}
\newtheorem{exercise}{Exercise}
```

Figure 3: Definition of the `exercise` environment

Proof. Proofs can be typeset using the `proof` environment. Please don't use your own environment for proofs. Also, don't use anything else than the standard end-of-proof symbol. If the placement of the end-of-proof symbol is not optimal, it can be forced to a different position using `\qedhere`. □

Floats and images

Use floats for everything that is longer than a few lines and should not break across pages. Most code examples, tables, diagrams etc. should use a floating environment. Don't be afraid that they do as their name says and appear in a different place as they have been defined.

All floating environments need a caption and are numbered, so they can be referred to by number.

There are three predefined floating environments: `figure`, `table`, and `listing`.

Images should be included using `\includegraphics` or using the commands provided by the `pgf` package (`pgfdeclareimage` and `pgfuseimage`). Avoid using `\epsfig` or the like.

Cross-references

Never refer to page numbers absolutely, always use `\pageref`. In the official The Monad.Reader, your article will not start on page 1.

LaTeX offers powerful facilities to cross-reference correctly to figures, code listings, theorems, equations etc. Use `\label` and `\ref/pageref` whenever possible.

If you refer to something by number, always mention what it is, and capitalize the category. I.e., say "Figure 1" rather than "figure 1".

Footnotes

Avoid them at as much as possible. Footnotes are disabled by default. If you need one, you have to use `\musthavefootnote` rather than `\footnote`.

Verbatim and code

Code examples should, if they exceed a few lines, be placed in floats, preferably of the `listing` category. Code can be either plain verbatim or formatted.

For displayed verbatim, use the `Verbatim` rather than the `verbatim` environment. This reimplementation, offered by the `fancyvrb` [3] package, offers many additional features. The class also allows to write inline verbatim between vertical bars, i.e., `|foo|` produces `foo`.

For formatted code, we recommend the `listings` [4] package or lhs2TeX [5].

Diagrams and images

For diagrams, we recommend METAPOST [6] and `pgf` [7], but other tools are possible, too. Try to use the same (i.e., Computer Modern) fonts in pictures, if possible without too much effort.

Math

Displayed math

Avoid using `$$` for display-style math. Use `\[` and `\]` or any of the `amsmath` [8] environments, such as `align`, `multline`, or `gather`. Similarly, don't use `eqnalign`, but rather `align` or `alignat`.

Don't be afraid of using display-style math. There is no page limit for articles in TMR, so space is not an issue. If a formula is too high to fit on a line and TEX increases the interline spacing in order to cope with it, this is usually a good indication that the formula should be displayed.

Numbering of equations

Don't use equation numbers if you do not need them for reference. On the other hand, do use equation numbers if you refer to them! Use `\eqref` to refer to equations. Don't use symbols to mark equations, use the standard mechanism. Example:

$$1 + 1 \tag{1}$$

The equation (1) evaluates to 2.

Text in math

Text in math mode should be written using `\text`. If you want to print words in italic within math mode, use `\text{\textit{foo}}` or `\mathit{foo}`, but never simply `foo`. Compare the results

$$foo \text{ (plain foo)} \qquad \mathit{foo} \text{ (math italics).}$$

General advice

Spelling

Please, please, please use a spell checker. There are plenty of free spell checkers that know to ignore LATEX commands, such as ispell or aspell, readily available. It makes the life of an editor much easier.

Don't touch the defaults

Most default settings are chosen consciously, to allow a consistent appearance of The Monad.Reader to the reader. Therefore, most things should just be left alone, for example:

- don't change the page layout
- don't change the default fonts
- don't change the font sizes (i.e., avoid generally the use of `\small`, `\large`, etc.)
- don't change the vertical spacing (i.e., avoid the use of `\vskip` or `\vspace`, `\\`, `\bigskip`, `\medskip`, `\smallskip`, and do not change the interline space)

Line and page breaks

It is the job of the editor to get page breaks right. Avoid inserting commands such as `\enlargethispage`, `\pagebreak`, or `\newpage` commands in your article. On the other hand, try to help LaTeX to break lines where it fails on its own. Prevent undesired line breaks using non-breakable spaces `~`. Prevent hyphenation of words using `\mbox`. Help with the hyphenation of words using `\-`. Try not to have any overfull horizontal boxes in your final document.

Usage of dashes

Use the `-` symbol only to connect words as in "type-checking algorithm". Use `--` for ranges as "1–5". In this case, the '–' is not surrounded by spaces. Use '–' also to separate thoughts – such as this one – from the rest of the sentence. In this case, the '–' is surrounded by spaces. Do not use the `---` symbol '—' at all.

Quotation marks

Use two backticks to produce an opening quotation mark, and two single quotes to produce a closing one. For example, ` ``this'' ` will produce "this". Do not use actual double quote symbols in a LaTeX document, since it will look bad, ”like this”.

Put punctuation before a closing quotation mark only if the punctuation actually belongs to the phrase or sentence being quoted. Do not do "this," instead, do "this".

e.g. and i.e.

You should generally avoid using the abbreviations *e.g.* and *i.e.* However, if you must:

- Don't confuse them. *e.g.* stands for *exempli gratia* and means "for example"; *i.e.* stands for *id est* and means "that is".
- Use the provided commands `\ie` and `\eg` so they will be properly typeset.
- Do not follow them by a comma.

Use complete sentences

Try to use complete sentences, even if they involve enumerations or math.

Try to avoid starting sentences with mathematical symbols or function names. Good: The function `map` rocks. Bad: `map` rocks.

Use consistent markup

Use italics for mathematical symbols, also when they appear embedded in text: We have n items. Use consistent markup also for Haskell function names. If you use verbatim code, then use function `map`. If you use lhs2TEX or something comparable, use function `map`.

Prefer `\emph` for emphasis, over `\textbf` or `\textit`. More importantly, don't use `\bf`, `\it`, `\sc`, etc. – use `\textbf`, `\textit`, `\textsc` etc. instead.

Try to define your own macros and use logical markup, so that changes in layout are easy at a later stage. If you write an article using lots of Haskell code, lhs2TEX [5] can make your life much easier!

Footnotes

Try to avoid using footnotes. Footnotes tend to disrupt the typesetting and are usually unneccessary: remember that The Monad.Reader is not an academic publication!

Bibliography

Don't link to web pages directly; make them a reference instead. Try to include web pages for as many references as possible.

The citations are ordered in order of appearance. Citations appear as numbers. Try to avoid using bibliography numbers as nouns. Don't say: In [9], we see this. Rather say: Peyton Jones [9] shows that. Don't be afraid to use the names of authors in your sentences. It can save your readers the effort of looking up who wrote the paper.

Use a space before a `\cite`, or better, a `~`. Generally, try to use `~` frequently to avoid bad line breaks. For example, writing `a function~$f$` is better than `a function $f$`.

Final remarks

If you have any specific instructions for the editor, add an attachment to your submission. Don't include in-line remarks to editors or reviewers.

The Monad.Reader is meant to be a light-hearted and easy to read. Don't write code that is gratuitously complex. Don't shun mathematical formulas, but avoid using too many technical terms without explaining what you mean. Not everyone knows what a left Kan extension is – relish the opportunity to explain these things to a wider audience.

Try to keep your sentences short and to the point. Keep your writing informal, but precise. Use plenty of examples to drive your point home. Don't try to write an academic paper about the unified theory of computing science; just try to show how cool functional programming can be. Above all, however, don't forget to have fun!

References

[1] `http://www.haskell.org/haskellwiki/TheMonadReader`.

[2] `http://darcs.net/`.

[3] `http://ctan.org/tex-archive/macros/latex/contrib/fancyvrb/`.

[4] `http://ctan.org/tex-archive/macros/latex/contrib/listings/`.

[5] `http://www.informatik.uni-bonn.de/~loeh/lhs2tex/`.

[6] `http://cm.bell-labs.com/who/hobby/MetaPost.html`.

[7] `http://sourceforge.net/projects/pgf/`.

[8] `http://ctan.org/tex-archive/macros/latex/required/amslatex/math/`.

[9] Simon Peyton Jones. Tackling the awkward squad: monadic input/output, concurrency, exceptions, and foreign-language calls in Haskell. In Tony Hoare, Manfred Broy, and Ralf Steinbruggen (editors), **Engineering theories of software construction**, pages 47–96. IOS Press (2001). `http://research.microsoft.com/Users/simonpj/papers/marktoberdorf/`.