The logictools Package

Miles Min Yin Cheang

May 3, 2025

Contents

1 Purpose of this package

The star of the show here is the formallogic environment. Prior to the development of this environment, spending way too much time fiddling around with spacing commands was a familiar experience for every logician. Most of the spacing you need in a logical statement is context sensitive, so only so much can be done through basic macros. Furthermore, using too many macros destroys the readability of the code, and slows down writing to a crawl.

In an effort to change this, I wrote an environment that both *speeds up* writing formal logic (by offering shorter syntax) and improves the output considerably. The details of how this works will be presented in the upcoming sections. The default settings were made with LaTeX's default math font in mind, with the intention that the user come up with a preset that matches their preferences. The options can be changed on the fly, so more than one preset can be used in different parts of the document.

Other than this, the option 'oxford' will load a few neat macros that might be of particular interest to those studying logic at the University of Oxford; they provide shortcuts to notations that are commonly used in the first-year courses. It is likely that this section of the package will be updated with more content as I go through my degree.

2 The formallogic environment

2.1 Introduction

This interface, accessed through the environment named `formallogic`, or the command `\fmllgc{<content>}`, helps to type formal logic in LaTeX. Here are some of its uses:

Code:	Output:
`|forall, x ; exists, y| (Ryx)`	$\forall x \; \exists y \; (Ryx)$
`|f,x;e,y|(Ryx)`	$\forall x \; \exists y \; (Ryx)$
`((P \land Q) \liff R)`	$\llparenthesis\!\llparenthesis\, P \land Q \,\rrparenthesis \leftrightarrow R \,\rrparenthesis\!\rrparenthesis$
`((P \land Q) \liff R)`	$((P \land Q) \leftrightarrow R))$
`((P \land Q) \liff R)`	$(\;(\;\; P \land Q \;\;) \leftrightarrow R \;\;)\;)$

You will be shown how to accomplish every one of these in the following documentation. Note that there are various user-defined parameters controlling the typesetting (e.g. spacing, kerning, parenthesis style); this is how the same code can produce wildly different outputs. Furthermore, the user can control certain parts of the syntax (e.g. the names of quantifiers).

2.2 Quantifier stacks

Quantifier stacks are a concept introduced for typesetting logical quantifiers:

$$\forall x \; \exists y \; \forall x_1 \; \exists z \in \mathbb{R}$$

```
| forall, x ; exists, y ; forall, x_1 ; exists, z\in\mathbb{R} |
```

Quantifier stacks are used by the formallogic environment. They are delimited by '`|`'. A quantifier stack is made up of quantifiers, written in the form `<label>,<argument>`. The label consists of some text that indicates which quantifier will be used, while the argument can be any math mode code. These quantifiers are separated by ';'. The formallogic environment processes these stacks, turning them into a fully typeset sequence of quantifiers.

Spacing on either side of the label and argument is trimmed, but spacing inside the label is not. This means 'for all' is a distinct label from 'forall'.

2.2.1 Declaring quantifiers

A declared quantifier has the following form:

$$\cdots\underbrace{\sqcup}_{\texttt{<left padding>}}\boxed{\texttt{<command>}}\underbrace{\sqcup}_{\texttt{<right padding>}}\cdots$$

$$\text{e.g.}\underbrace{\sqcup\!\sqcup}_{\texttt{10mu}}\exists!\underbrace{\sqcup}_{\texttt{5mu}}x$$

Syntax for quantifier declaration:

```
\DeclareQuantifier{<label>}{<command>}[<left pad>][<right pad>]
```

This command will globally declare (or redeclare) a quantifier with the associated properties. The label consists of some text that refers to this quantifier; the command should be a LaTeX command providing the quantifier symbol; left and right padding are optional padding values on either side of the command. For instance: `\DeclareQuantifier{ex!}{\exists !}[5mu][1.5mu]` provides a quantifier that can be used like: `|ex!,x;forall,y|` $\to$ $\exists!x\;\forall y$.

```
\LDeclareQuantifier{<label>}{<command>}[<left pad>][<right pad>]
```

This command does the same thing, but locally. With this, one can quickly redefine a quantifier in the middle of the environment, and not worry about the changes carrying over to other instances of the environment.

2.3 Other syntax

- Parentheses written consecutively (without spaces) will become parenthesis stacks, and use `parstackkern` instead of `parinnerpad` as spacing.
- `[<arg 1>/<arg 2>]` $\to$ $[{}^{\texttt{<1>}}/{}_{\texttt{<2>}}]$, allowing one to write easy variable substitutions inline[1].
- `.=` $\to \doteq$, providing quick access to `\doteq`.

One can use `"<content>"` within the environment to escape `<content>`, preventing it from being parsed by the environment; this is useful when one wishes to use a character that is active in the syntax of the environment. The delimiter used here is the double quote, `"` (U+0022).

For example, this can be used to write a function with single parentheses in a double parenthesis environment, or a list using commas inside of a quantifier stack[2]:

[1]Note that this means '[' is by default active in the syntax, and so requires escaping if one wishes to use it without following it with '/' and then ']'.

[2]Actually, '`,`' and '`;`' only need escaping when inside of a quantifier stack, delimited by '`|`'.

```
\logictoolsoptions{partype=double, parinnerpad=3.5mu}
\begin{formallogic}
|forall,"x_1,x_2,x_3,\ldots";exists,y|(f"(y)"=x_1+x_2+x_3+\ldots\land Py)
\end{formallogic}
```

Produces: $\forall x_1, x_2, x_3, \ldots\ \exists y \;\llparenthesis f(y) = x_1 + x_2 + x_3 + \ldots \land Py \rrparenthesis$

The only syntax that is not escaped like this is `.=` $\to \doteq$, since checks for escaping slightly lower performance, and this can already be escaped with a space between the two characters.

2.4 Typesetting features

2.4.1 Customisation

The formallogic environment offers many customisation options through user-adjustable keys. They may be changed with the command `\logictoolsoptions{<key>=<value>, ... }` (in either the document or the preamble). A list of key-value pairs may also be given in an optional argument to the formallogic environment. The following keys are available:

Key	Description	Accepts	Default
`partype`	Determines the type of parenthesis used, single '(...)' or double '⦅ ... ⦆'.	`single`, `double`	`single`
`parinnerpad`	Extra space inserted between parentheses and their content.	mu	`0.9mu`
`parstackkern`	Kern applied to stacked parentheses.	mu	`-0.9mu`
`italiccorrection`	Extra kern between closing parentheses and their content, to offset italic math font.	mu	`1.12mu`
`parvoffset`	Amount to raise parentheses by; helps center them on text in some fonts.	ex	`0.2ex`
`quantskip`	Default skip inserted between quantifiers.	mu	`4.32mu`
`lastquantskip`	Default skip inserted after last quantifier.	mu	`4.32mu`
`scriptspace`	Determines space after sub/superscript, same as the LaTeX primitive.	em	`-0.025em`

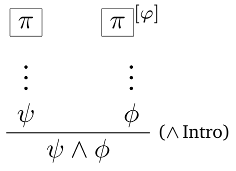
3 The ‘oxford’ package option

This package option adds a few macros for common notations at University of Oxford.

3.1 Good looking ‘proof from φ to ψ’

‘A proof π from φ to ψ’ (perhaps with some discharged assumptions) might be notated like this:

```
\begin{prooftree}
    \alwaysNoLine
    \AxiomC{\fbox{$\pi$}}
    \UnaryInfC{$\vdots$}
    \UnaryInfC{$\psi$}
    \AxiomC{\fbox{$\pi$}$^{[\varphi]}$}
    \UnaryInfC{$\vdots$}
    \UnaryInfC{$\phi$}
    \alwaysSingleLine
    \andlabel{Intro}[3]
    \BinaryInfC{$\psi \land \phi$}
\end{prooftree}
```



The output is not ideal; introducing discharged assumptions puts the box off center in an annoying way, and the `\vdots` are not aligned correctly. The following command achieves better output with nicer syntax:

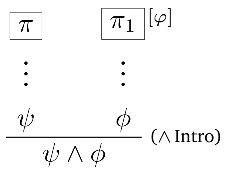
$$\texttt{\textbackslash prooffrom\{<1>\}}\underbrace{\ \ }_{\texttt{<2>}}\texttt{\{<3>\}}$$

<2> = Either ‘^’ (for superscript), ‘_’ (for subscript), or nothing (for centered script).

+

Some `content` delimited by `[···]` (for square-bracketed content) or `<···>` (for non-square-bracketed content).

So `\prooffrom{$\pi$}{$\psi$}`, `\prooffrom{$\pi_1$}^[$\varphi$]{$\phi$}` become:



[3]The macro `\andlabel{#1}` gives `\RightLabel{\scriptsize($\land$\hspace{1px}#1)}`.

3.2 Bits and Bobs

[Math mode only.] `\difmost{<variable>}`

Gives the variable assignment notation: $\alpha \overset{v}{\sim} \beta$, meaning '$\beta$ differs from α in at most v'.

`\lcma`

Gives ˒, the 'logical comma' that Professor Beau Mount uses in the PTLP lecture notes.

`\semval{<sent.>}{<structure>}[<var.assign.>]`

Gives $|$`<sent.>`$|^{\alpha}_{\mathcal{A}}$, the semantic value of some sentence over model $\mathcal{A}$ with variable assignment α. The input `<structure>` is converted to `\mathcal{...}`. If the input `<var.assign.>` is a single latin letter (e.g. 'a', 'b', 'd' 'g'), it is converted into an appropriate greek one[4].

`\lsym{<language>}[<signature>]`[5]

Gives $\mathcal{L}_{\circ}$, where $\circ$ can be 1,2,= or something else. Also optionally allows the addition of a superscript, for a signature. The '=' uses `\@ltoolsshorteq`, '=', which is prettier in most fonts.

---

[4] This respects capitalisation, so one gets γ from 'g', and Γ from 'G'.

[5] This command loads even without the package option 'oxford'. Why? Because I couldn't get it to work otherwise.