Code tips from “Introduktion til LaTeX”*

Or just the dlfltxbcodetips package

Lars Madsen†

March 4, 2010

Contents

Introduction 1

1 Extra symbols 2
1.1 A big version of \times . . . 2
1.2 Negated up- and downarrows . . . 2

2 Fun with theorems 2
2.1 Shaded or framed theorems with the ntheorem package . . . 2
2.2 Theorems that start with a list . . . 3

3 Various features regarding alignment 4
3.1 Alignment with material encased in delimiters on different lines . . . 4

4 Declaring sets 5

5 Misc 6
5.1 Additional layout for the pfgpages package . . . 6
5.2 Overloading _ in math-mode . . . 6

Bibliography 6

Introduction

In my LaTeX book (Madsen, 2010) we present some macros that might be helpful to the readers. Some of these extra macros might be useful to others as well so these macros have been included in the `dlfltxbcodetips` package. The package is published on CTAN and the package is released under the normal lppl license.

The »`dlfltxb`« part of the name simply stands for »daleif« and LaTeX book. The `dlfltxbcodetips` package is the first package in the »`dlfltxb`«-bundle which, over

*Version: 0.2

†Web: http://home.imf.au.dk/daleif Email: daleif@imf.au.dk

time, will contain most of the home made packages that I use to create my book (though not the book source itself).

Some of the macros might be better of included in the `mathtools` package by Morten Høgholm, but he is quite busy elsewhere at the moment.

Note: The macro `\dbx` will often be used to simulate some text or mathematical material.

1 Extra symbols

1.1 A big version of `\times`

`\bigtimes` A few extra symbols have been created. First of is `\bigtimes` which is a large operator version of `\times`, but without having to load special fonts.[1]

```
$\bigtimes_{n=1}^k A_n$
\[ \bigtimes_{n=1}^k A_n \]
```

$\bigtimes_{n=1}^{k} A_n$

$$\bigtimes_{n=1}^{k} A_n$$

1.2 Negated up- and downarrows

`\nuparrow` `\ndownarrow` The package creates `\nuparrow` and `\ndownarrow` by rotating and reflecting `\nrightarrow` and `\nleftarrow` respectively.[1]

```
$ A  \nuparrow B$ \qquad
$ B \ndownarrow C$
```

$A \nuparrow B \qquad B \ndownarrow C$

Remark. The `mathdesign` package is incompatible with `amssymb`, but it does define the symbols we need from it to define `\nuparrow` and `\ndownarrow`. Use

```
\usepackage[noamssymb]{dlfltxbcodetips}
```

to disable the autoloading of `amssymb`, and remember to load `dlfltxbcodetips` *after* `mathdesign`.

2 Fun with theorems

2.1 Shaded or framed theorems with the ntheorem package

`\NewShadedTheorem` The `ntheorem` pacakge can create shaded or framed theorems, but they take up to much space (in my opinion). So we make our own macro `\NewShadedTheorem`. It has exactly the same syntax as the ordinary `\newtheorem`. Requirements: the `framed`, `ntheorem` (loaded with the `framed` option), and `color` or `xcolor`. You will have to `\theoremframecommand` redefine `\theoremframecommand` to get a background color or a frame. This package initialises `\theoremframecommand` to do nothing.

[1] Updated version due to Enrico Gregorio.

Caveat. The theorem environment constructed will not have a stared companion as `ntheorems` normal `\newtheorem` does.

```
\def\theoremframecommand{\fboxsep=10pt\fbox}
\NewShadedTheorem{sthm}{Theorem}[chapter]
\def\theoremframecommand{%
\colorbox{red}}
\NewShadedTheorem{slemma}[sthm]{Lemma}
\newtheorem{prop}[sthm]{Proposition}
\begin{sthm}
  normal test.
\end{sthm}
\begin{slemma}
  a lemma.
\end{slemma}
\begin{prop}
  a theorem with no background.
\end{prop}
```

Theorem. normal test.

Lemma. a lemma.

Proposition. a theorem with no background.

2.2 Theorems that start with a list

A theorem that starts with a list looks odd because the first item comes directly after the heading.[2]

```
\begin{thm}
  \begin{enumerate}
  \item \dbx[2cm]
  \item \dbx[2cm]
  \item \dbx[2cm]
  \end{enumerate}
\end{thm}
```

Theorem 1. 1. ☐

2. ☐

3. ☐

`\InsertTheoremBreak` The macro `\InsertTheoremBreak` helps.

```
\begin{thm}
  \InsertTheoremBreak
  \begin{enumerate}
  \item \dbx[2cm]
  \item \dbx[2cm]
  \item \dbx[2cm]
  \end{enumerate}
\end{thm}
\begin{thm}
  \InsertTheoremBreak*
  \begin{enumerate}
  \item \dbx[2cm]
  \item \dbx[2cm]
  \item \dbx[2cm]
  \end{enumerate}
\end{thm}
```

Theorem 2.

1. ☐

2. ☐

3. ☐

Theorem 3.

1. ☐

2. ☐

3. ☐

The un-stared version remove the space above the list, the stared version does not.

[2] Depends on the configuration.

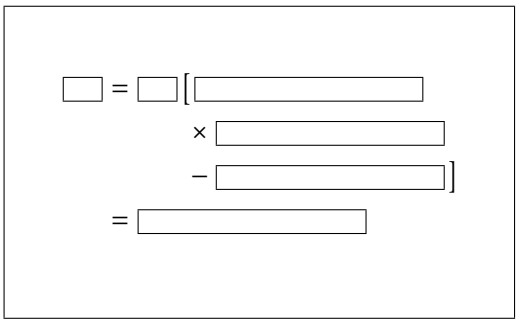
Caveat. If the theorem comes too close to a page break it is quite likely that the page break will end up between the theorem header and the start of the list.

3 Various features regarding alignment

3.1 Alignment with material encased in delimiters on different lines

Swanson also recommends that if one has material encased with delimiters and the delimiters are on different lines, then if space permits, the material should be indented such that the relationship is evident. Usually we would use a `\phantom` to do this, such as in the next example.

```
\begin{align*}
  \dbx[5mm]&= \dbx[5mm]\bigl[{} \dbx[3cm]\\
   &\phantom{=\dbx[5mm]\bigl[{}}
   \times \dbx[3cm]{}\\
   &\phantom{=\dbx[5mm]\bigl[{}}
   - \dbx[3cm]{}\bigr]\\
   &=\dbx[3cm]
\end{align*}
```



The problem with this is that it gets tedious and prone to human error. How about instead maintaining a stack of material determining the indentation together with tools to reset, add to and pop the stack. For this you can use the following macros

`\MathIndent` `\MathIndent`

`\SetMathIndent` `\SetMathIndent{`⟨*math code*⟩`}`

`\AddtoMathIndent` `\AddtoMathIndent{`⟨*math code*⟩`}`

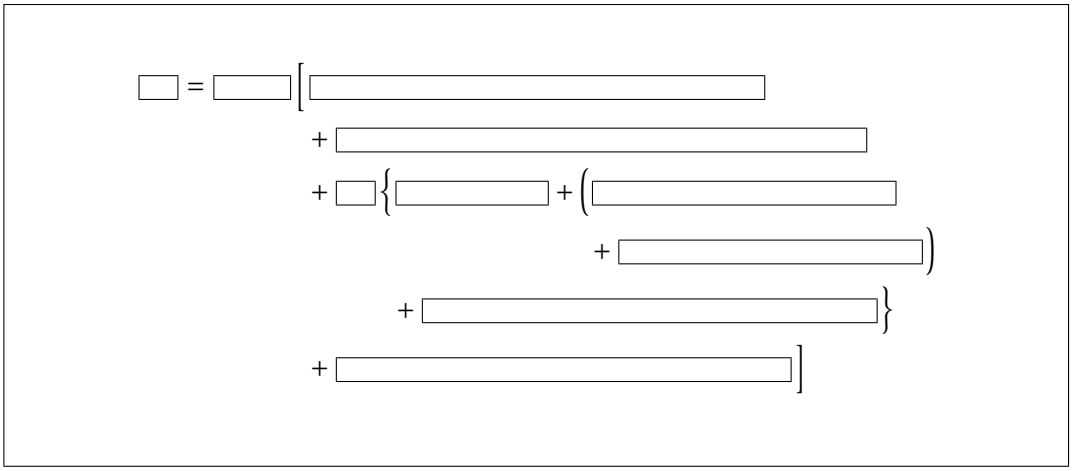
`\PopMathIndent` `\PopMathIndent`

`\PopMathIndent*` `\PopMathIndent*`

`\MathIndent` is used to set a space corresponding to the current indentation saved on the stack. `\SetMthIndent` takes its argument and saves it on the stack, calculates the current math indent length and ends by typesetting the given argument, i.e. no need to copy anything. Similarly the `\AddtoMathIndent` adds it argument to the stack and adds the length of it to the saved math indent. So instead of copying code, now we simply encase it with either `\SetMathIndent` (for the initialisation) or `\AddtoMathIndent`. `\PopMathIndent` is similar to `\MathIndent`, in that it sets a blank space corresponding to the contents of the stack after we have popped off the top item. `\PopMathIndent*` pops the stack but does *not* set any space.

Now, an illustrative example might be in order:

```
\begin{align*}
\dbx ={} & \SetMathIndent{\dbx[1cm] \Bigl[} \dbx[6cm]  \\
   & \MathIndent   + \dbx[7cm]  \\
   & \MathIndent
   \AddtoMathIndent{{} + \dbx \Bigl\{}
   \AddtoMathIndent{\dbx[2cm] + \Bigl(} \dbx[4cm] \\
   &
   \MathIndent + \dbx[4cm] \Bigr) \\
   & \PopMathIndent + \dbx[6cm] \Bigr\} \\
   & \PopMathIndent + \dbx[6cm] \Bigr]
\end{align*}
```



Notice the dual use of `\AddtoMathIndent` such that we can return to the indentation set by the »{«.

Of course, non-balanced `\left`–`\right` constructions may not be used.

4 Declaring sets

This still needs some work

It is a good idea to avoid the one (or two) letter shortcuts for sets etc., e.g. `\R` for `\mathbb{R}`, it can cause problems when co-writing articles with people with other naming habits.

To help with this we provide

`\DeclareMathSet` `\DeclareMathSet[`⟨*options*⟩`]{`⟨*identifier*⟩`}`

By default it can be used as

```
\DeclareMathSet{R}% => \numbersR = \mathbb{R}
$ \numbersR $
```

$\mathbb{R}$

But we have several options to change things. Options (⟨*key*⟩=⟨*value*⟩ style, note that ⟨*identifier*⟩ is what is given to the formating macro).

format the macro used to format the output, default: `\mathbb`

name this defaults to ⟨*identifier*⟩, but can be used to change a part of the macro name, e.g. `\DeclareMathSet[name=Cat,format=\matcal]{C}` results in `\setCat=\mathcal{C}`.

prefix this defaults to `set`, but one might want to use

```
\DeclareMathSet[prefix=group,format=\mathrm]{U}
```

to get `\groupU`, the unitary group.

overwrite boolean, if true, then you can overwrite an existing macro, otherwise it will throw an error.

Note that for sets it is not a good idea to declare them as math operators, as some mathematical operations take sets as their argument, so the spacing before an operator would be wrong in that case.

Note that this will also be added to the mathtools package, and then removed from this package

5 Misc

5.1 Additional layout for the pfgpages package

`\ProvidePGFPagesFourOnOneWithSpaceForNotes` The macro `\ProvidePGFPagesFourOnOneWithSpaceForNotes` will activate a »4␣on␣1␣with␣space␣for␣notes« layout to be used with the `pgfpages` package. It is basically the same as the »8␣on␣1« layout but leaving the second column empty for reader to fill in their own notes.

5.2 Overloading _ in math-mode

`\OverloadUnderscoreInMath` Placing `\OverloadUnderscoreInMath` in the preamble will overload the _ character in math-mode such that

```
\[
   X_{ab} = X_|max|
\]
```

$$X_{ab} = X_{\max}$$

that is `_|...|` is that same as `_{\textup{...}}`.

Bibliography

Lars Madsen. Introduktion til LaTeX. http://www.imf.au.dk/system/latex/bog/, 2010. The current version of the book is 3rd edition beta.