Code tips from “Introduktion til LaTeX”*

Or just the dlfltxbcodetips package

Lars Madsen†

June 1, 2007

Contents

Introduction

In my LaTeX book (Madsen, 2007) 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 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. over time some of the macros might be imcluded in

*Version: 0.2

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

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$
```

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

The macros require the use of the graphicx package, which is not auto loaded.

2 Fun with theorems

2.1 Shaded theorems with the ntheorem package

Even though we have the shadedthm package, we can easily do better if we are already using the ntheorem package. Simply use

\NewShadedTheorem `\NewShadedTheorem`

It has exactly the same syntax as the ordinary \newtheorem. Requirements: the framed, ntheorem, (x)color packages and the definition of the »shadecolor« (required by the framed package). Inside the \NewShadedTheorem the environment surrounding the \NSTshadeenvironment theorem is given by the macro \NSTshadeenvironment which is initialised to »shaded«. You can change it using \renewcommand.

[1] Updated version due to Enrico Gregorio.

```
\theoremheaderfont{\bfseries}
\theoremseparator{.}
\NewShadedTheorem{thm}{Theorem}[chapter]
\NewShadedTheorem{lemma}[thm]{Lemma}
\newtheorem{prop}[thm]{Proposition}
\begin{thm}
  normal test.
\end{thm}
\begin{thm*}
  un-numbered.
\end{thm*}
\begin{lemma}
  a lemma.
\end{lemma}
\begin{prop}
  a theorem with no background.
\end{prop}
```

Theorem 2.1. normal test.

Theorem. un-numbered.

Lemma 2.2. a lemma.

Proposition 2.3. a theorem with no background.

In contrast to `ntheorem`, the `*`-ed version created by `\NewShadedTheorem` is a version that does not print a number.

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 2.4. 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 2.5.

1. ☐

2. ☐

3. ☐

Theorem 2.6.

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 Arrow between lines

Sometimes lines in an alignment are related in the sense of *from which it follows*, usually indicated by ⇒. We would like to place this between the lines in a nice way. The following macro is due to Morten Høgholm.

`\ArrowBetweenLines` `\ArrowBetweenLines[`⟨*arrow*⟩`]`
`\ArrowBetweenLines*` `\ArrowBetweenLines*[`⟨*arrow*⟩`]`

Simply hold back one column of alignment for the arrow:

```
\begin{alignat}{2}
  && f(x) & = \dbx[2cm] \\
  \ArrowBetweenLines
  && g(x) & = - \dbx[2cm] \\
  \ArrowBetweenLines[\Downarrow]
  && h(x) & = 0
\end{alignat}
```

$$f(x) = \qquad (1)$$
$$\Updownarrow$$
$$g(x) = - \qquad (2)$$
$$\Downarrow$$
$$h(x) = 0 \qquad (3)$$

Note the height of the arrow line and that the line automatically does not contain equation numbers. The stared version can be used to place the arrows on the right. Though several alignment columns on each line might become a problem.

```
\begin{alignat}{2}
  f(x) & = \dbx[2cm] \\
  \ArrowBetweenLines*
   g(x) & = - \dbx[2cm] \\
  \ArrowBetweenLines*[\Downarrow]
   h(x) & = 0
\end{alignat}
```

$$f(x) = \qquad (4)$$
$$\Updownarrow$$
$$g(x) = - \qquad (5)$$
$$\Downarrow$$
$$h(x) = 0 \qquad (6)$$

3.2 Switch dead space

The standard ***equation*** environment has the feature that if the text before it is short and the formula likewise, then the drop between the preceding text and the formula will be `\abovedisplayshortskip` instead of the usual `\abovedisplayskip`. The macros from the `amsmath` package does not have this feature (except ***equation****).

```
text
\[ a=b \]
text
\begin{align*}
  a=b
\end{align*}
```

text

$$a = b$$

text

$$a = b$$

\SwapDeadSpace The macro \SwapDeadSpace can simulate this feature

```
text
\[ a=b \]
text
\begin{align*}
  \SwapDeadSpace
  a=b
\end{align*}
```

text

$$a = b$$

text

$$a = b$$

3.3 Left alignment

In Swanson (1999), Ellen Swanson presents some recommendations regarding how to arrange displayed formulas. One thing she recommends is alignment to the left and with all subsequent lines indented by 2 em.

$$\begin{aligned} \square &= \square \\ &= \square \\ &= \square . \end{aligned}$$

or

$$\begin{aligned} \square \\ &= \square \\ &= \square . \end{aligned}$$

or

$$\begin{aligned} \square &= \square \\ &+ \square \\ &\times \square . \end{aligned}$$

and

$$\begin{aligned} \square \\ &- \square \\ &\otimes \square . \end{aligned}$$

But having to do all those indentations by hand quickly becomes quite dull. Instead, align everything to the left, and pull back the first line. This is what the following macro does.

`\PullBack` `\PullBack[⟨number⟩]`

The optional argument is a number (without unit, em will be used) indicating the amount of indentation. 2 em is the default. Now simply use `\PullBack` instead of the »&« on the first line.

```
\begin{align*}
  \PullBack f(x) + g(x) + \dbx[4cm] \\
  & = \dbx[5cm] \\
  & = \dbx[5cm]
\end{align*}
```

$$f(x) + g(x) + \square$$
$$= \square$$
$$= \square$$

The optional argument is usually used whenever the alignment is moved to the right of the equal signs.

```
\begin{align*}
  \PullBack[3] f(x) + g(x) + \dbx[4cm] \\
   ={} & \dbx[5cm] \\
   & + \dbx[5cm]
\end{align*}
```

$$f(x) + g(x) + \square$$
$$= \square$$
$$+ \square$$

3.4 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*}
```

$$\square = \square\left[\square\right.$$
$$\times \square$$
$$\left. - \square\right]$$
$$= \square$$

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⟩}`

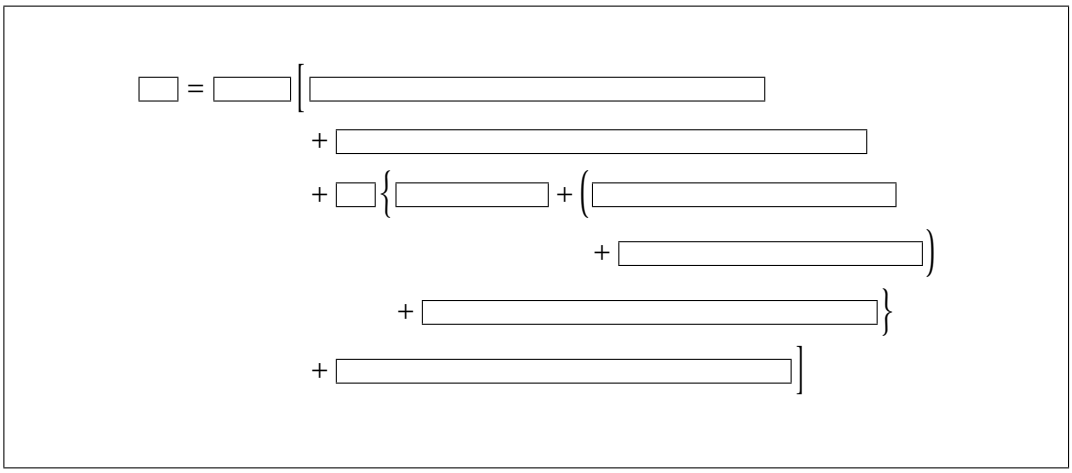
`\PopMathIndent` `\PopMathIndent`

`\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.

Bibliography

Lars Madsen. Introduktion til LaTeX. http://www.imf.au.dk/system/latex/bog/, 2007. The current version of the book is 3rd edition beta.

Ellen Swanson. *Mathematics into Type*. American Mathematical Society, 1999. An set of old notes, updated in 1999 by Arlene O'Sean og Antoinette Schleyer.