LaTeX Sugar for Isabelle documents

Florian Haftmann, Gerwin Klein, Tobias Nipkow, Norbert Schirmer

April 19, 2009

Abstract

This document shows how to typset mathematics in Isabelle-based documents in a style close to that in ordinary computer science papers.

1 Introduction

This document is for those Isabelle users who have mastered the art of mixing LaTeX text and Isabelle theories and never want to typeset a theorem by hand anymore because they have experienced the bliss of writing `@{thm[display]setsum_cartesian_product[no_vars]}` and seeing Isabelle typeset it for them:

$$(\sum x{\in}A.\ \sum y{\in}B.\ f\ x\ y) = (\sum (x,\ y){\in}A \times B.\ f\ x\ y)$$

No typos, no omissions, no sweat. If you have not experienced that joy, read Chapter 4, *Presenting Theories*, [1] first.

If you have mastered the art of Isabelle's *antiquotations*, i.e. things like the above `@{thm...}`, beware: in your vanity you may be tempted to think that all readers of the stunning ps or pdf documents you can now produce at the drop of a hat will be struck with awe at the beauty unfolding in front of their eyes. Until one day you come across that very critical of readers known as the "common referee". He has the nasty habit of refusing to understand unfamiliar notation like Isabelle's infamous $[\![\ ]\!] \Longrightarrow$ no matter how many times you explain it in your paper. Even worse, he thinks that using $[\![\ ]\!]$ for anything other than denotational semantics is a cardinal sin that must be punished by instant rejection.

This document shows you how to make Isabelle and LaTeX cooperate to produce ordinary looking mathematics that hides the fact that it was typeset by a machine. You merely need to load the right files:

- Import theory `LaTeXsugar` in the header of your own theory. You may also want bits of `OptionalSugar`, which you can copy selectively into your own theory or import as a whole. Both theories live in `HOL/Library` and are found automatically.

- Should you need additional LaTeX packages (the text will tell you so), you include them at the beginning of your LaTeX document, typically in `root.tex`. For a start, you should `\usepackage{amssymb}` — otherwise typesetting $\neg(\exists\, x.\ P\ x)$ will fail because the AMS symbol $\nexists$ is missing.

2 HOL syntax

2.1 Logic

The formula $\neg(\exists\, x.\ P\ x)$ is typeset as $\nexists\, x.\ P\ x$.

The predefined constructs *if*, *let* and *case* are set in sans serif font to distinguish them from other functions. This improves readability:

- *if* b *then* e_1 *else* e_2 instead of *if* b *then* e_1 *else* e_2.
- *let* $x = e_1$ *in* e_2 instead of *let* $x = e_1$ *in* e_2.
- *case* x *of* *True* $\Rightarrow e_1$ | *False* $\Rightarrow e_2$ instead of *case* x *of* *True* $\Rightarrow e_1$ | *False* $\Rightarrow e_2$.

2.2 Sets

Although set syntax in HOL is already close to standard, we provide a few further improvements:

- $\{x \mid P\}$ instead of $\{x.\ P\}$.
- $\emptyset$ instead of $\{\}$, where $\emptyset$ is also input syntax.
- $\{a,\ b,\ c\} \cup M$ instead of *insert* a (*insert* b (*insert* c M)).

2.3 Lists

If lists are used heavily, the following notations increase readability:

- $x{\cdot}xs$ instead of $x\ \#\ xs$, where $x{\cdot}xs$ is also input syntax. If you prefer more space around the $\cdot$ you have to redefine `\isasymcdot` in LaTeX: `\renewcommand{\isasymcdot}{\isamath{\,\cdot\,}}`
- $|xs|$ instead of *length* xs.
- $xs_{[n]}$ instead of *nth* xs n, the nth element of xs.

- Human readers are good at converting automatically from lists to sets. Hence `OptionalSugar` contains syntax for suppressing the conversion function *set*: for example, $x \in set\ xs$ becomes $x \in xs$.

- The @ operation associates implicitly to the right, which leads to unpleasant line breaks if the term is too long for one line. To avoid this, `OptionalSugar` contains syntax to group @-terms to the left before printing, which leads to better line breaking behaviour:

  $term_0$ @ $term_1$ @ $term_2$ @ $term_3$ @ $term_4$ @ $term_5$ @ $term_6$ @ $term_7$ @ $term_8$ @ $term_9$ @ $term_{10}$

2.4 Numbers

Coercions between numeric types are alien to mathematicians who consider, for example, *nat* as a subset of *int*. `OptionalSugar` contains syntax for suppressing numeric coercions such as $int :: nat \Rightarrow int$. For example, *int 5* is printed as *5*. Embeddings of types *nat*, *int*, *real* are covered; non-injective coercions such as $nat :: int \Rightarrow nat$ are not and should not be hidden.

3 Printing theorems

3.1 Question marks

If you print anything, especially theorems, containing schematic variables they are prefixed with a question mark: `@{thm conjI}` results in $[\![P;\ Q]\!] \Longrightarrow P \wedge Q$. Most of the time you would rather not see the question marks. There is an attribute `no_vars` that you can attach to the theorem that turns its schematic into ordinary free variables: `@{thm conjI[no_vars]}` results in $[\![P;\ Q]\!] \Longrightarrow P \wedge Q$.

This `no_vars` business can become a bit tedious. If you would rather never see question marks, simply put

```
reset show_question_marks;
```

at the beginning of your file `ROOT.ML`. The rest of this document is produced with this flag reset.

Hint: Resetting `show_question_marks` only suppresses question marks; variables that end in digits, e.g. *x1*, are still printed with a trailing *.0*, e.g. *x1.0*, their internal index. This can be avoided by turning the last digit into a subscript: write `x\<^isub>1` and obtain the much nicer x_1.

3.2 Qualified names

If there are multiple declarations of the same name, Isabelle prints the qualified name, for example *T.length*, where T is the theory it is defined in, to distinguish it from the predefined *List.length*. In case there is no danger of confusion, you can insist on short names (no qualifiers) by setting `short_names`, typically in `ROOT.ML`:

```
set short_names;
```

3.3 Variable names

It sometimes happens that you want to change the name of a variable in a theorem before printing it. This can easily be achieved with the help of Isabelle's instantiation attribute `where`: $[\![\varphi;\ \psi]\!] \Longrightarrow \varphi \wedge \psi$ is the result of

```
@{thm conjI[where P = \<phi> and Q = \<psi>]}
```

To support the "_"-notation for irrelevant variables the constant `DUMMY` has been introduced: *fst* $(a,\ \_) = a$ is produced by

```
@{thm fst_conv[where b = DUMMY]}
```

3.4 Inference rules

To print theorems as inference rules you need to include Didier Rémy's `mathpartir` package [2] for typesetting inference rules in your LaTeX file.

Writing `@{thm[mode=Rule] conjI}` produces $\dfrac{P \qquad Q}{P \wedge Q}$, even in the middle of a sentence. If you prefer your inference rule on a separate line, maybe with a name,

$$\frac{P \qquad Q}{P \wedge Q}\ \textsc{conjI}$$

is produced by

```
\begin{center}
@{thm[mode=Rule] conjI} {\sc conjI}
\end{center}
```

It is not recommended to use the standard `display` option together with `Rule` because centering does not work and because the line breaking mechanisms of `display` and `mathpartir` can clash.

Of course you can display multiple rules in this fashion:

```
\begin{center}
@{thm[mode=Rule] conjI} {\sc conjI} \\[1ex]
@{thm[mode=Rule] conjE} {\sc disjI$_1$} \qquad
@{thm[mode=Rule] disjE} {\sc disjI$_2$}
\end{center}
```

yields

$$\frac{P \qquad Q}{P \wedge Q}\ \textsc{conjI}$$

$$\frac{P}{P \vee Q}\ \textsc{disjI}_1 \qquad \frac{Q}{P \vee Q}\ \textsc{disjI}_2$$

The `mathpartir` package copes well if there are too many premises for one line:

$$\frac{A \longrightarrow B \qquad B \longrightarrow C \qquad C \longrightarrow D \qquad D \longrightarrow E \qquad E \longrightarrow F \qquad F \longrightarrow G \qquad G \longrightarrow H \qquad H \longrightarrow I \qquad I \longrightarrow J \qquad J \longrightarrow K}{A \longrightarrow K}$$

Limitations: 1. Premises and conclusion must each not be longer than the line. 2. Premises that are $\Longrightarrow$-implications are again displayed with a horizontal line, which looks at least unusual.

In case you print theorems without premises no rule will be printed by the `Rule` print mode. However, you can use `Axiom` instead:

```
\begin{center}
@{thm[mode=Axiom] refl} {\sc refl}
\end{center}
```

yields

$$\frac{}{t = t}\ \textsc{refl}$$

3.5 Displays and font sizes

When displaying theorems with the `display` option, e.g. `@{thm[display] refl}`

$t = t$

the theorem is set in small font. It uses the LaTeX-macro `\isastyle`, which is also the style that regular theory text is set in, e.g.

lemma $t = t$

Otherwise `\isastyleminor` is used, which does not modify the font size (assuming you stick to the default `\isabellestyle{it}` in `root.tex`). If you prefer normal font size throughout your text, include

```
\renewcommand{\isastyle}{\isastyleminor}
```

in `root.tex`. On the other hand, if you like the small font, just put `\isastyle` in front of the text in question, e.g. at the start of one of the center-environments above.

The advantage of the display option is that you can display a whole list of theorems in one go. For example, `@{thm[display] foldl.simps}` generates

$foldl\ f\ a\ [] = a$
$foldl\ f\ a\ (x{\cdot}xs) = foldl\ f\ (f\ a\ x)\ xs$

3.6 If-then

If you prefer a fake "natural language" style you can produce the body of

Theorem 1 *If $i \le j$ and $j \le k$ then $i \le k$.*

by typing

```
@{thm[mode=IfThen] le_trans}
```

In order to prevent odd line breaks, the premises are put into boxes. At times this is too drastic:

Theorem 2 *If longpremise and longerpremise and $P\ (f\ (f\ (f\ (f\ (f\ (f\ (f\ (f\ x)))))))))$ and longestpremise then conclusion.*

In which case you should use `IfThenNoBox` instead of `IfThen`:

Theorem 3 *If longpremise and longerpremise and $P\ (f\ (f\ (f\ (f\ (f\ (f\ (f\ (f\ x)))))))))$ and longestpremise then conclusion.*

3.7 Doing it yourself

If for some reason you want or need to present theorems your own way, you can extract the premises and the conclusion explicitly and combine them as you like:

- `@{thm_style prem1` *thm*`}` prints premise 1 of *thm* (and similarly up to `prem9`).
- `@{thm_style concl` *thm*`}` prints the conclusion of *thm*.

For example, "from Q and P we conclude $P \wedge Q$" is produced by

```
from @{thm_style prem2 conjI} and @{thm_style prem1 conjI}
we conclude @{thm_style concl conjI}
```

Thus you can rearrange or hide premises and typeset the theorem as you like. The `thm_style` antiquotation is a general mechanism explained in §5.

lemma *True*
proof −
  — pretty trivial
  show *True* **by** *force*
qed

Figure 1: Example proof in a figure.

3.8 Patterns

In §3.3 we shows how to create patterns containing "_". You can drive this game even further and extend the syntax of let bindings such that certain functions like *fst*, *hd*, etc. are printed as patterns. `OptionalSugar` provides the following:

let $(x, \_) = p$ *in* t	produced by `@{term "let x = fst p in t"}`
let $(\_, x) = p$ *in* t	produced by `@{term "let x = snd p in t"}`
let $x{\cdot}\_ = xs$ *in* t	produced by `@{term "let x = hd xs in t"}`
let $\_{\cdot}x = xs$ *in* t	produced by `@{term "let x = tl xs in t"}`
let *Some* $x = y$ *in* t	produced by `@{term "let x = the y in t"}`

4 Proofs

Full proofs, even if written in beautiful Isar style, are likely to be too long and detailed to be included in conference papers, but some key lemmas might be of interest. It is usually easiest to put them in figures like the one in Fig. 1. This was achieved with the **text_raw** command:

```
text_raw {*
  \begin{figure}
  \begin{center}\begin{minipage}{0.6\textwidth}
  \isastyleminor\isamarkuptrue
*}
lemma True
proof -
  -- "pretty trivial"
  show True by force
qed
text_raw {*
  \end{minipage}\end{center}
  \caption{Example proof in a figure.}\label{fig:proof}
  \end{figure}
*}
```

Other theory text, e.g. definitions, can be put in figures, too.

5 Styles

The `thm` antiquotation works nicely for single theorems, but sets of equations as used in definitions are more difficult to typeset nicely: people tend to prefer aligned = signs.

To deal with such cases where it is desirable to dive into the structure of terms and theorems, Isabelle offers antiquotations featuring "styles":

```
@{thm_style stylename thm}
@{term_style stylename term}
```

A "style" is a transformation of propositions. There are predefined styles, namely `lhs` and `rhs`, `prem1` up to `prem9`, and `concl`. For example, the output

$$\begin{aligned} \mathit{foldl}\ f\ a\ []\ &= a \\ \mathit{foldl}\ f\ a\ (x{\cdot}xs)\ &= \mathit{foldl}\ f\ (f\ a\ x)\ xs \end{aligned}$$

is produced by the following code:

```
\begin{center}
\begin{tabular}{l@ {~~@{text "="}~~}l}
@{thm_style lhs foldl_Nil} & @{thm_style rhs foldl_Nil}
@{thm_style lhs foldl_Cons} & @{thm_style rhs foldl_Cons}
\end{tabular}
\end{center}
```

Note the space between `@` and `{` in the tabular argument. It prevents Isabelle from interpreting `@ {~~...~~}` as an antiquotation. The styles `lhs` and `rhs` extract the left hand side (or right hand side respectively) from the conclusion of propositions consisting of a binary operator (e. g. $=$, $\equiv$, $<$).

Likewise, `concl` may be used as a style to show just the conclusion of a proposition. For example, take `hd_Cons_tl`:

$$(xs :: 'a\ \mathit{list}) \neq [] \Longrightarrow \mathit{hd}\ xs{\cdot}\mathit{tl}\ xs = xs$$

To print just the conclusion,

$$\mathit{hd}\ (xs :: 'a\ \mathit{list}){\cdot}\mathit{tl}\ xs = xs$$

type

```
\begin{center}
@{thm_style [show_types] concl hd_Cons_tl}
\end{center}
```

Beware that any options must be placed *before* the name of the style, as in this example.

Further use cases can be found in §3.7.

If you are not afraid of ML, you may also define your own styles. A style is implemented by an ML function of type `Proof.context -> term -> term`. Have a look at the following example:

```
setup {*
let
  fun my_concl ctxt = Logic.strip_imp_concl
  in TermStyle.add_style "my_concl" my_concl
end;
*}
```

This example shows how the `concl` style is implemented and may be used as as a "copy-and-paste" pattern to write your own styles.

The code should go into your theory file, separate from the LaTeX text. The `let` expression avoids polluting the ML global namespace. Each style receives the current proof context as first argument; this is helpful in situations where the style has some object-logic specific behaviour for example.

The mapping from identifier name to the style function is done by the `TermStyle.add_style` expression which expects the desired style name and the style function as arguments.

After this `setup`, there will be a new style available named `my_concl`, thus allowing antiquoations like `@{thm_style my_concl hd_Cons_tl}` yielding $hd\ xs \cdot tl\ xs = xs$.

References

[1] T. Nipkow, L. Paulson, and M. Wenzel. *Isabelle/HOL — A Proof Assistant for Higher-Order Logic*, volume 2283. 2002. http://www.in.tum.de/~nipkow/LNCS2283/.

[2] D. Rémy. mathpartir. http://cristal.inria.fr/~remy/latex/.