LaTeX Sugar for Isabelle Documents

Florian Haftmann, Gerwin Klein, Tobias Nipkow, Norbert Schirmer

October 24, 2026

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,mode=latex_sum] sum_Suc_diff [no_vars]}` and seeing Isabelle typeset it for them:

$$m \leq Suc\ n \Longrightarrow (\sum_{i=m}^{n} f\ (Suc\ i) - f\ i) = f\ (Suc\ n) - f\ m$$

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 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`.

- 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:

- $\mathsf{if}\ b\ \mathsf{then}\ e_1\ \mathsf{else}\ e_2$ instead of $\mathit{if}\ b\ \mathit{then}\ e_1\ \mathit{else}\ e_2$.
- $\mathsf{let}\ x = e_1\ \mathsf{in}\ e_2$ instead of $\mathit{let}\ x = e_1\ \mathit{in}\ e_2$.
- $\mathsf{case}\ x\ \mathsf{of}\ \mathit{True} \Rightarrow e_1 \mid \mathit{False} \Rightarrow e_2$ instead of $\mathit{case}\ x\ \mathit{of}\ \mathit{True} \Rightarrow e_1 \mid \mathit{False} \Rightarrow e_2$.

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 $\mathit{insert}\ a\ (\mathit{insert}\ b\ (\mathit{insert}\ c\ M))$.
- $|A|$ instead of $\mathit{card}\ A$.

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.
- $|xs|$ instead of $\mathit{length}\ xs$.
- $xs_{[n]}$ instead of $\mathit{nth}\ xs\ n$, 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, $x \in \mathit{set}\ xs$ 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 constants and their type

Instead of `@{const myconst}` `@{text "::"}` `@{typeof myconst}`, you can write `@{const_typ myconst}` using the new antiquotation `const_typ` defined in `LaTeXsugar`. For example, `@{const_typ length}` produces *length* :: *?'a list* ⇒ *nat* (see below for how to suppress the question mark). This works both for genuine constants and for variables fixed in some context, especially in a locale.

4 Printing theorems

The $[\![P;\ Q]\!] \Longrightarrow R$ syntax is a bit idiosyncratic. If you would like to avoid it, you can easily print the premises as a conjunction: $P \wedge Q \Longrightarrow R$. See `OptionalSugar` for the required "code".

4.1 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]}
⇝   [|P; Q|] ⟹ P ∧ Q
```

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

```
options [show_question_marks = false]
```

into the relevant `ROOT` file, just before the `theories` for that session. The rest of this document is produced with this flag set to `false`.

4.2 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 the `names_short` configuration option in the context.

4.3 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`:

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

$\rightsquigarrow$ $[\![\varphi; \psi]\!] \Longrightarrow \varphi \wedge \psi$

To support the "_"-notation for irrelevant variables the constant `DUMMY` has been introduced:

```
@{thm fst_conv[of _ DUMMY]}
```

$\rightsquigarrow$ *fst* (*x1.0*, _) = *x1.0*

As expected, the second argument has been replaced by "_", but the first argument is the ugly *x1.0*, a schematic variable with suppressed question mark. Schematic 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\<^sub>1` and obtain the much nicer x_1. Alternatively, you can display trailing digits of schematic and free variables as subscripts with the `sub` style:

```
@{thm (sub) fst_conv[of _ DUMMY]}
```

$\rightsquigarrow$ *fst* (x_1, _) = x_1

The insertion of underscores can be automated with the `dummy_pats` style:

```
@{thm (dummy_pats,sub) fst_conv}
```

$\rightsquigarrow$ *fst* (x_1, _) = x_1

The theorem must be an equation. Then every schematic variable that occurs on the left-hand but not the right-hand side is replaced by `DUMMY`. This is convenient for displaying functional programs.

Variables that are bound by quantifiers or lambdas can be renamed with the help of the attribute `rename_abs`. It expects a list of names or underscores, similar to the `of` attribute:

```
@{thm split_paired_All[rename_abs _ l r]}
```
$\rightsquigarrow$ $(\forall x.\ P\ x) = (\forall l\ r.\ P\ (l,\ r))$

Sometimes Isabelle η-contracts terms, for example in the following definition:

fun *eta* **where**
$eta\ (x \cdot xs) = (\forall y \in set\ xs.\ x < y)$

If you now print the defining equation, the result is not what you hoped for:

```
@{thm eta.simps}
```
$\rightsquigarrow$ $eta\ (x \cdot xs) = Ball\ xs\ ((<)\ x)$

In such situations you can put the abstractions back by explicitly η-expanding upon output:

```
@{thm (eta_expand z) eta.simps}
```
$\rightsquigarrow$ $eta\ (x \cdot xs) = (\forall z{\in}xs.\ x < z)$

Instead of a single variable `z` you can give a whole list `x y z` to perform multiple η-expansions.

4.4 Breaks and boxes

Printing longer formulas can easily lead to line breaks in unwanted places. This can be avoided by putting LaTeX-like mboxes in formulas. There is a function $mbox :: 'a \Rightarrow 'a$ that you can wrap around subterms and that is pretty-printed as a LaTeX `\mbox{ }`. If you are printing a term or formula, you can just insert *mbox* wherever you want. You can also insert it into theorems by virtue of the fact that *mbox* is defined as an identity function. Of course you need to adapt the proof accordingly.

Unless the argument of *mbox* is an identifier or an application (i.e. of the highest precedence), it will be enclosed in parentheses. Thus `x < mbox(f y)` results in $x < f\ y$ but `x < mbox(y+z)` results in $x < (y + z)$. You can switch off the parentheses by using the variant *mbox0* instead of *mbox*: `x < mbox0(y+z)` results in $x < y + z$.

4.5 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:

$$\frac{\begin{array}{ccccc} A \longrightarrow B & B \longrightarrow C & C \longrightarrow D & D \longrightarrow E & E \longrightarrow F \\ F \longrightarrow G & G \longrightarrow H & H \longrightarrow I & I \longrightarrow J & J \longrightarrow K \end{array}}{A \longrightarrow K}$$

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

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

4.6 Displays and font sizes

When displaying theorems with the `display` option, for example as in `@{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] append.simps}` generates

$[] \mathbin{@} ys = ys$
$x \cdot xs \mathbin{@} ys = x \cdot xs \mathbin{@} ys$

4.7 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* $P\ (f\ (f\ (f\ (f\ (f\ (f\ (f\ (f\ (f\ x)))))))))$ *and longestpremise then conclusion.*

4.8 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 (prem 1)` *thm*`}` prints premise 1 of *thm*.
- `@{thm (concl)` *thm*`}` prints the conclusion of *thm*.

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

```
from @{thm (prem 2) conjI} and @{thm (prem 1) conjI}
we conclude @{thm (concl) conjI}
```

Thus you can rearrange or hide premises and typeset the theorem as you like. Styles like `(prem 1)` are a general mechanism explained in §5.

4.9 Patterns

In §4.3 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"}`

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) thm}
@{prop (style) thm}
@{term (style) term}
@{term_type (style) term}
@{typeof (style) term}
```

A "style" is a transformation of a term. There are predefined styles, namely `lhs` and `rhs`, `prem` with one argument, and `concl`. For example, the output

$$\begin{array}{lcl} [] \mathbin{@} ys & = & ys \\ x \cdot xs \mathbin{@} ys & = & x \cdot xs \mathbin{@} ys \end{array}$$

is produced by the following code:

```
\begin{center}
\begin{tabular}{l@ {~~@{text "="}~~}l}
@{thm (lhs) append_Nil} & @{thm (rhs) append_Nil}\\
@{thm (lhs) append_Cons} & @{thm (rhs) append_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`:

$$xs \neq [] \Longrightarrow hd\ xs \cdot tl\ xs = xs$$

To print just the conclusion,

$$hd\ xs \cdot tl\ xs = xs$$

type

```
\begin{center}
@{thm (concl) hd_Cons_tl}
\end{center}
```

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

Further use cases can be found in §4.8. If you are not afraid of ML, you may also define your own styles. Have a look at module `Term_Style`.

6 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:

lemma *True*
proof −
  show *True* **by** (*rule TrueI*)
qed

Figure 1: Example proof in a figure.

```
text_raw \<open>
  \begin{figure}
  \begin{center}\begin{minipage}{0.6\textwidth}
  \isastyleminor\isamarkuptrue
\<close>
lemma True
proof -
  show True by (rule TrueI)
qed
text_raw \<open>
  \end{minipage}\end{center}
  \caption{Example proof in a figure.}\label{fig:proof}
  \end{figure}
\<close>
```

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

7 Theory snippets

This section describes how to include snippets of a theory text in some other LaTeX document. The typical scenario is that the description of your theory is not part of the theory text but a separate document that antiquotes bits of the theory. This works well for terms and theorems but there are no antiquotations, for example, for function definitions or proofs. Even if there are antiquotations, the output is usually a reformatted (by Isabelle) version of the input and may not look like you wanted it to look. Here is how to include a snippet of theory text (in LaTeX form) in some other LaTeX document, in 4 easy steps. Beware that these snippets are not processed by any antiquotation mechanism: the resulting LaTeX text is more or less exactly what you wrote in the theory, without any added sugar.

7.1 Theory markup

Include some markers at the beginning and the end of the theory snippet you want to cut out. You have to place the following lines before and after the snippet, where snippets must always be consecutive lines of theory text:

```
\text_raw\<open>\snip{snippetname}{1}{2}{%\<close>
```

theory text
`\text_raw\<open>%endsnip\<close>`

where *snippetname* should be a unique name for the snippet. The numbers `1` and `2` are explained in a moment.

7.2 Generate the `.tex` file

Run your theory `T` with the `isabelle build` tool to generate the LaTeX-file `T.tex` which is needed for the next step, extraction of marked snippets. You may also want to process `T.tex` to generate a pdf document. This requires a definition of `\snippet`:

```
\newcommand{\repeatisanl}[1]
  {\ifnum#1=0\else\isanewline\repeatisanl{\numexpr#1-1}\fi}
\newcommand{\snip}[4]{\repeatisanl#2#4\repeatisanl#3}
```

Parameter 2 and 3 of `\snippet` are numbers (the `1` and `2` above) and determine how many newlines are inserted before and after the snippet. Unfortunately `text_raw` eats up all preceding and following newlines and they have to be inserted again in this manner. Otherwise the document generated from `T.tex` will look ugly around the snippets. It can take some iterations to get the number of required newlines exactly right.

7.3 Extract marked snippets

Extract the marked bits of text with a shell-level script, e.g.

`sed -n '/\\snip{/,/endsnip/p' T.tex >` *snippets*`.tex`

File *snippets*`.tex` (the name is arbitrary) now contains a sequence of blocks like this

`\snip{`*snippetname*`}{1}{2}{%`
theory text
`}%endsnip`

7.4 Including snippets

In the preamble of the document where the snippets are to be used you define `\snip` and input *snippets*`.tex`:

```
\newcommand{\snip}[4]
  {\expandafter\newcommand\csname #1\endcsname{#4}}
\input{snippets}
```

This definition of `\snip` simply has the effect of defining for each snippet *snippetname* a LaTeX command `\`*snippetname* that produces the corresponding snippet text. In the body of your document you can display that text like this:

```
\begin{isabelle}
\snippetname
\end{isabelle}
```

The `isabelle` environment is the one defined in the standard file `isabelle.sty` which most likely you are loading anyway.

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/.