MMTTeX: Integrating Formal Mathematics in MMT and Narrative Mathematics in LaTeX

Florian Rabe

FAU Erlangen and LRI Paris

Abstract

This document serves as an announcement, documentation, and example of the MMTTeX tool that allows integrating formal/logical content written in MMT with informal/narrative one written in LaTeX. Background knowledge is formalized in MMT and can be exported as LaTeX sty files. Then documents written in LaTeX and may import this background knowledge. The documents may declare further formal knowledge, which is interpreted by MMT on the fly during the LaTeX compilation workflow.

In particular, documents may contain mathematical formulas in MMT syntax: MMT parses and type-checks these formulas, enriches them by inferring types and implicit arguments, and eventually produces LaTeX presentations for them. The latter then contains annotations generated by MMT such as tooltips and cross-references.

This document is itself written in MMTTeX.

1 Formalizing the Background Knowledge

The MMT system[1] provides a wide variety of logical frameworks and logics that are readily available for use with mmttex. Alternatively, users may formalize novel background knowledge in MMT.

For this document, we assume the definition of first-order logic in the MMT/examples archive. It can be found at https://gl.mathhub.info/MMT/examples/tree/master/source/logic.

2 Writing LaTeX Documents

MMTTeX documents generate two kinds of output: the usual presentation output produced by LaTeX, e.g., a pdf file, and a content output in the form of an MMT file. Correspondingly, we have to distinguish two kinds of MMTTeX commands:

[1] https://uniformal.github.io/

- **Content-only** commands declare formal knowledge that is independent of the presentation output. They set up the context in which MMT processes the presentation-relevant commands. They occur in the content output and produce no or almost no presentation output.
- **Presentation-relevant** commands produce output in the usual LaTeX way.

This document is itself an example of how to use MMTTeX, and all MMTTeX commands are duplicated as listings. As a running example, we build a theory of groups.

2.1 Namespace Abbreviations

These are content-only commands that declare namespace abbreviations to be used later.

```
\mmtimport{ex}{http://cds.omdoc.org/examples}
```

2.2 MMT Theories

The environment mmttheory is used to group the content-only commands into scopes. The begin/end commands of this environment are content-only, but the environment may contain arbitrary presentation-relevant commands. It takes two arguments: the name of the theory and the MMT URI of its meta-theory.

```
\begin{mmttheory}{Group}{ex:?FOLEQNatDed}
```

Now we have opened a theory named Group whose meta-theory is (using the namespace abbreviation from above) http://cds.omdoc.org/examples?FOLEQNatDed, a formalization of first-order logic from MMT's examples archive.

2.3 Constant Declarations

We declare the basic operations of monoids:

```
\mmtconstant{op}{term ⟶ term ⟶ term}{}{1 ∘ 2 prec 5}
\mmtconstant{unit}{term}{}{e}
\mmtconstant{inv}{term ⟶ term}{}{1 ' prec 10}
```

Here the content-only command mmtconstant takes 4 arguments: name, type, definiens, and notation. Apart from the names, all may be omitted by leaving them empty. prec sets the precedence, i.e., inversion binds stronger than composition.

As examples of defined constants, we define the commutator and division:

```
\mmtconstant{division}{term ⟶ term ⟶ term}{[x,y] x∘y'}{1 / 2}
\mmtconstant{commutator}{term ⟶ term ⟶ term}{[x,y] x∘y∘x'∘y'}{< 1 , 2 >}
```

2.4 MMT-Processed Formulas

So far all commands were content-only. Now we say the main example of a presentation-relevant command. The syntax "..." can be used as an analog to LaTeX's usual $...$. But the formula is interpreted by MMT.

As example, we use the commutator function again: The function $[x,y]x \circ y \circ x' \circ y'$ takes two group elements and returns their commutator. Using the above definition, we can also write it as $[x,y]<x,y>$.

Often we need to open local scopes in which we declare variables. For that, we use the content-only mmtcontext environment. Contexts can be nested.

Inside a context, we can use the mmtvar command to declare variables. It combines content and presentation aspects: it produces a content declaration of a variable that can be referred to in the same scope and the presentation of the variable name. There is also a content-only variant mmtvar*.

First we declare a variable as part of the text: consider a group element a. Now we can use it to write objects like a'. We silently declare a second variable, which we use below.

Now we can say that $a \circ b \circ a' \circ b'$ is the commutator of a and b.

2.5 Closing Theories

Eventually, we close our theory of groups:

```
\end{mmttheory}
```

2.6 Theory Inclusions

Theories may include other theories. It does not matter whether a theory is part of the background knowledge or declared in the current document.

For example, we create a new theory that includes the one above and the natural numbers from the background knowledge:

```
\begin{mmttheory}{Group2}{ex:?FOLEQNatDed}
\mmtinclude{?Group}
\mmtinclude{ex:?Nat}
```

In this theory, we declare a power operation:

```
\mmtconstant{power}{term ⟶ nat ⟶ term}{}{1 ^ 2}
```

Finally, we close the theory:

```
\end{mmttheory}
```

3 Semantic Enrichment

Consider the MMT-processed formula "[n,x,y]x'^n / y", which is rendered as $[n,x,y]\frac{x'^n}{y}$.

The rendered formula contains a number of advanced features:

- Several delimiters are interpreted as two-dimensional notations including ^ for superscripting and / for fractions.
- Every operator carries a hyperlink: try clicking on the λ or the inversion operator.
- The inferred types of the bound variables occur as tooltips over the variables: try hovering over the binding of x.

4 Compilation Workflow

MMTTeX is activated by including the mmttex.sty package.

In line with typical LaTeX workflows, MMTTeX requires two compilations of the source file with MMTTeX run in between.

4.1 Background Knowledge

MMT provides the build target mmt-sty. It is used to converts the background knowledge into LaTeX packages to make it available to LaTeX documents. It is executed as build ARCHIVE mmt-sty, where ARCHIVE is the name of an MMT archive. It produces a LaTeX sty file for each MMT theory.

For example, to compile this document, users should run

```
build MMT/urtheories mmt-sty
build MMT/examples mmt-sty
```

in the MMT shell.

4.2 MMT Server

mmttex can be run as a self-contained tool. However, if a LaTeX document uses a lot of background knowledge, it is advisable to run MMT permanently as a server so that the background does not have to be reloaded every time. This has the additional advantage that hyperlinks to the background knowledge can be resolved in the web browser.

To start an MMT server, you can execute for example

```
mmt --keepalive server on 8080
```

4.3 First LaTeX Pass

The first LaTeX pass exports all MMT content into an JOB.tex.mmt file that is produced in addition to the usual output files.

Example For this document itself, the file is (except for possibly some Unicode characters that are not displayed correctly when including it into LaTeX):

```
import ex http://cds.omdoc.org/Æexamples
theory Group
 : ex:?FOLEQNatDed
 =
constant op
 : term ⟶ term ⟶ term Ø
 # 1 ∘ 2 ØŒ

constant unit
 : term Ø
 # e ØŒ

constant inv
 : term ⟶ term Ø
 # 1 ' ØŒ

constant division
 : term ⟶ term ⟶ term Ø
 = [x,y] x∘y' Ø
 # 1 / 2 ØŒ

constant commutator
 : term ⟶ term ⟶ term Ø
 = [x,y] x∘y∘x'∘y' Ø
 # < 1 , 2 > ØŒ

constant mmt@2.4.0 = [x,y] x∘y∘x'∘yŒ'
constant mmt@2.4.1 = [x,y]<x,yŒ>
theory CONTEXT_1
 =
constant a
 : term ØŒ

constant mmt@2.4.2 = aŒ'
theory CONTEXT_2
 =
constant b
 : term ØŒ

constant mmt@2.4.3 = a∘b∘a'∘bŒ'
constant mmt@2.4.4 = Œa
constant mmt@2.4.5 = ŒbÆÆÆ



theory Group2
 : ex:?FOLEQNatDed
 =
include ?Group Œ
include ex:?Nat Œ
constant power
 : term ⟶ nat ⟶ term Ø
 # 1 ^ 2 ØŒÆ
```

```
theory Group3
 : ex:?FOLEQNatDed
 =
include ?Group2 Œ
constant mmt@3.0.0 = [n,x,y] x'^n / ŒyÆ
```

Explanations Two things are particularly interesting:

- All mmtcontext environments become nested theories with auto-generated names.
- All presentation-relevant objects become defined constants, whose names are of the form mmt@ followed by section number and a formula relative to the section.

4.4 MMTTeX

Next MMTTeX reads this file and produces the file JOB.tex.mmt.sty, where JOB is the main LaTeX file. For this document, assuming all paths are set up correctly, this is simply the command

```
mmttex example.tex
```

Any type-checking errors are printed to standard output.

Example For this document, the resulting file is:

```
\RequireMMTPackage{MMT/examples/export/presentation/mmt-sty/content
    /http..cds.omdoc.org/examples/$F$O$L$E$Q$Nat$Ded}
\newcommand{\Group@@op}[2]{#1\mmt@symref{file:/C:/frabe/MMT-devel/
    src/latex-mmt/latex/example.tex?Group?op}{\circ }#2}
\newcommand{\Group@@unit}{\mmt@symref{file:/C:/frabe/MMT-devel/src/
    latex-mmt/latex/example.tex?Group?unit}{e}}
\newcommand{\Group@@inv}[1]{#1\mmt@symref{file:/C:/frabe/MMT-devel/
    src/latex-mmt/latex/example.tex?Group?inv}{'}}
\newcommand{\Group@@division}[2]{\frac{#1}{#2}}
\newcommand{\Group@@commutator}[2]{\mmt@symref{file:/C:/frabe/MMT-
    devel/src/latex-mmt/latex/example.tex?Group?commutator}{<}#1\
    mmt@symref{file:/C:/frabe/MMT-devel/src/latex-mmt/latex/example
    .tex?Group?commutator}{,}#2\mmt@symref{file:/C:/frabe/MMT-devel
    /src/latex-mmt/latex/example.tex?Group?commutator}{>}}
\expandafter\def\csname mmt@2.4.0\endcsname{\mmt@group{\
    LambdaPi@@lambda{{\mmt@vardecl[term]{x}{\FOL@@term}{}},{\
    mmt@vardecl[term]{y}{\FOL@@term}{}}}{\mmt@group{\Group@@inv{\
    mmt@group{\Group@@op{\mmt@group{\Group@@inv{\mmt@group{\
    Group@@op{\mmt@group{\Group@@op{x}{y}}}{x}}}}}{y}}}}}}}
\expandafter\def\csname mmt@2.4.1\endcsname{\mmt@group{\
    LambdaPi@@lambda{{\mmt@vardecl[term]{x}{\FOL@@term}{}},{\
    mmt@vardecl[term]{y}{\FOL@@term}{}}}{\mmt@group{\
    Group@@commutator{x}{y}}}}}
\newcommand{\Group@CONTEXTUOne@@a}{a}
\expandafter\def\csname mmt@2.4.2\endcsname{\mmt@group{\Group@@inv
    {\Group@CONTEXTUOne@@a}}}
```

```
\newcommand{\Group@CONTEXTUOne@CONTEXTUTwo@@b}{b}
\expandafter\def\csname mmt@2.4.3\endcsname{\mmt@group{\Group@@inv
    {\mmt@group{\Group@@op{\mmt@group{\Group@@inv{\mmt@group{\
    Group@@op{\mmt@group{\Group@@op{\Group@CONTEXTUOne@@a}{\
    Group@CONTEXTUOne@CONTEXTUTwo@@b}}}{\Group@CONTEXTUOne@@a
    }}}}}}{\Group@CONTEXTUOne@CONTEXTUTwo@@b}}}}}
\expandafter\def\csname mmt@2.4.4\endcsname{\Group@CONTEXTUOne@@a}
\expandafter\def\csname mmt@2.4.5\endcsname{\
    Group@CONTEXTUOne@CONTEXTUTwo@@b}
\RequireMMTPackage{MMT/examples/export/presentation/mmt-sty/content
    /http..cds.omdoc.org/examples/$F$O$L$E$Q$Nat$Ded}
\RequireMMTPackage{MMT/examples/export/presentation/mmt-sty/content
    /http..cds.omdoc.org/examples/$Nat}
\newcommand{\GroupTwo@@power}[2]{{#1}^{#2}}
\RequireMMTPackage{MMT/examples/export/presentation/mmt-sty/content
    /http..cds.omdoc.org/examples/$F$O$L$E$Q$Nat$Ded}
\expandafter\def\csname mmt@3.0.0\endcsname{\mmt@group{\
    LambdaPi@@lambda{{\mmt@vardecl[nat]{n}{\Nat@@nat}{}},{\
    mmt@vardecl[term]{x}{\FOL@@term}{}},{\mmt@vardecl[term]{y}{\
    FOL@@term}{}}}{\mmt@group{\Group@@division{\mmt@group{\
    GroupTwo@@power{\mmt@group{\Group@@inv{x}}}{n}}}{y}}}}}
```

Explanations With some effort, one can see the following:

- All theories from the background result in RequireMMTPackage commands.
- Every constants results in a macro definition, whose name is derived from the name of the containing theory and the constants. Types and definitions are ignored, and the notation of the constant is used to define the macro.
- Every MMT-processed formula results in a macro definition using the constant name. The definiens of the macro is the type-checked form of the object rendered using a set of macros that are defined by the mmttex package.

These special macros include

- Every delimiter is wrapped in mmt@symref and carries the MMT URI of the MMT constant that constructed the term. The macro definition uses this to create a cross-reference to the place where the constant was declared.
- mmt@group indicates the exact structure of the syntax tree, and mmt@vardecl is used for variable declarations.
- mmt@vardecl is used for variable declarations. If the type of the variable was inferred, it is additionally passed in MMT surface syntax, and the mmt@vardecl macro uses this to add the inferred as a tooltip on the variable.

4.5 Second LaTeX Pass

In the second LaTeX pass, mmttex.sty detects the presence of the generated style file. Now every time it encounters an MMT-processed formula, the macro produced in the previous step is executed, thus inserting the appropriate LaTeX object.