sTeX: Semantic Markup in TeX/LaTeX

Michael Kohlhase
Jacobs University, Bremen
http://kwarc.info/kohlhase

May 7, 2008

Abstract

We present a collection of TeX macro packages that allow to markup TeX/LaTeX documents semantically without leaving the document format, essentially turning TeX/LaTeX into a document format for mathematical knowledge management (MKM).

Contents

The sTeX Collection

1 Introduction

The last few years have seen the emergence of various content-oriented XML-based, content-oriented markup languages for mathematics on the web, e.g. OPENMATH [BCC+04], Content-MATHML [ABC+03], or our own OMDOC [Koh06]. These representation languages for mathematics, that make the structure of the mathematical knowledge in a document explicit enough that machines can operate on it. Other examples of content-oriented formats for mathematics include the various logic-based languages found in automated reasoning tools (see [RV01] for an overview), program specification languages (see e.g. [Ber89]).

The promise if these content-oriented approaches is that various tasks involved in "doing mathematics" (e.g. search, navigation, cross-referencing, quality control, user-adaptive presentation, proving, simulation) can be machine-supported, and thus the working mathematician is relieved to do what humans can still do infinitely better than machines: The creative part of mathematics — inventing interesting mathematical objects, conjecturing about their properties and coming up with creative ideas for proving these conjectures. However, before these promises can be delivered upon (there is even a conference series [MKM07] studying "Mathematical Knowledge Management (MKM)"), large bodies of mathematical knowledge have to be converted into content form.

Even though MATHML is viewed by most as the coming standard for representing mathematics on the web and in scientific publications, it has not not fully taken off in practice. One of the reasons for that may be that the technical communities that need high-quality methods for publishing mathematics already have an established method which yields excellent results: the TEX/LATEX system: and a large part of mathematical knowledge is prepared in the form of TEX/LATEX documents.

TEX [Knu84] is a document presentation format that combines complex page-description primitives with a powerful macro-expansion facility, which is utilized in LATEX (essentially a set of TEX macro packages, see [Lam94]) to achieve more content-oriented markup that can be adapted to particular tastes via specialized document styles. It is safe to say that LATEX largely restricts content markup to the document structure[1], and graphics, leaving the user with the presentational TEX primitives for mathematical formulae. Therefore, even though LATEX goes a great step into the direction of an MKM format, it is not, as it lacks infrastructure for marking up the functional structure of formulae and mathematical statements, and their dependence on and contribution to the mathematical context.

1.1 The XML vs. TEX/LATEX Formats and Workflows

MATHML is an XML-based markup format for mathematical formulae, it is standardized by the World Wide Web Consortium in [ABC+03], and is supported

[1]supplying macros e.g. for sections, paragraphs, theorems, definitions, etc.

by the major browsers. The MathML format comes in two integrated components: presentation MathML and content MathML. The former provides a comprehensive set of layout primitives for presenting the visual appearance of mathematical formulae, and the second one the functional/logical structure of the conveyed mathematical objects. For all practical concerns, presentation MathML is equivalent to the math mode of TEX. The text mode facilitates of TEX (and the multitude of LATEX classes) are relegated to other Xml formats, which embed MathML.

The programming language constructs of TEX (i.e. the macro definition facilities[2]) are relegated to the Xml programming languages that can be used to develop language extensions. transformation language xslt [Dea99, Kay00] or proper Xml-enabled The Xml-based syntax and the separation of the presentational-, functional- and programming/extensibility concerns in MathML has some distinct advantages over the integrated approach in TEX/LATEX on the services side: MathML gives us better

- integration with web-based publishing,
- accessibility to disabled persons, e.g. (well-written) MathML contains enough structural information to supports screen readers.
- reusability, searchabiliby and integration with mathematical software systems (e.g. copy-and-paste to computer algebra systems), and
- validation and plausibility checking.

On the other hand, TEX/LATEX/s adaptable syntax and tightly integrated programming features within has distinct advantages on the authoring side:

- The TEX/LATEX syntax is much more compact than MathML (see the difference in Figure 1 and Equation 1), and if needed, the community develops LATEX packages that supply new functionality in with a succinct and intuitive syntax.
- The user can define ad-hoc abbreviations and bind them to new control sequences to structure the source code.
- The TEX/LATEX community has a vast collection of language extensions and best practice examples for every conceivable publication purpose and an established and very active developer community that supports these.
- There is a host of software systems centered around the TEX/LATEX language that make authoring content easier: many editors have special modes for LATEX, there are spelling/style/grammar checkers, transformers to other markup formats, etc.

[2]We count the parser manipulation facilities of TEX, e.g. category code changes into the programming facilities as well, these are of course impossible for MathML, since it is bound to Xml syntax.

In other words, the technical community is is heavily invested in the whole workflow, and technical know-how about the format permeates the community. Since all of this would need to be re-established for a MathML-based workflow, the technical community is slow to take up MathML over TeX/LaTeX, even in light of the advantages detailed above.

1.2 A LaTeX-based Workflow for Xml-based Mathematical Documents

An elegant way of sidestepping most of the problems inherent in transitioning from a LaTeX-based to an Xml-based workflow is to combine both and take advantage of the respective advantages.

The key ingredient in this approach is a system that can transform TeXLaTeX documents to their corresponding Xml-based counterparts. That way, Xml-documents can be authored and prototyped in the LaTeX workflow, and transformed to Xml for publication and added-value services, combining the two workflows.

There are various attempts to solve the TeX/LaTeX to Xml transformation problem; the most mature is probably Bruce Miller's LaTeXML system [Mil07]. It consists of two parts: a re-implementation of the TeX analyzer with all of it's intricacies, and a extensible Xml emitter (the component that assembles the output of the parser). Since the LaTeX style files are (ultimately) programmed in TeX, the TeX analyzer can handle all TeX extensions, including all of LaTeX. Thus the LaTeXML parser can handle all of TeX/LaTeX, if the emitter is extensible, which is guaranteed by the LaTeXML binding language: To transform a TeX/LaTeX document to a given Xml format, all TeX extensions[3] must have "LaTeXML bindings"binding, i.e. a directive to the LaTeXML emitter that specifies the target representation in Xml.

2 The Packages of the sTeX Collection

In the following, we will shortly preview the packages and classes in the sTeX collection. They all provide part of the solution of representing semantic structure in the TeX/LaTeX workflow. We will group them by the conceptual level they address[1] EdNote(1)

2.1 Content Markup of Mathematical Formulae in TeX/LaTeX

The first two packages are concerned basically with the math mode in TeX, i.e. mathematical formulae. The underlying problem is that run-of-the-mill TeX/LaTeX only specifies the presentation (i.e. what formulae look like) and not

[3]i.e. all macros, environments, and syntax extensions used int the source document

[1]EdNote: come up with a nice overview figure here!

their content (their functional structure). Unfortunately, there are no good methods (yet) to infer the latter from the former, but there are ways to get presentation from content.

Consider for instance the following “standard notations”[4] for binomial coefficients: $\binom{n}{k}$, ${}_nC^k$, $\mathcal{C}^n_k$, and $\mathcal{C}^k_n$ all mean the same thing: $\frac{n!}{k!(n-k)!}$. This shows that we cannot hope to reliably recover the functional structure (in our case the fact that the expression is constructed by applying the binomial function to the arguments n and k) from the presentation alone.

The solution to this problem is to dump the extra work on the author (after all she knows what she is talking about) and give them the chance to specify the intended structure. The markup infrastructure supplied by the sTEX collection lets the author do this without changing[5] the visual appearance, so that the LaTeX workflow is not disrupted. . We speak of semantic preloading for this process and call our collection of macro packages sTEX (Semantic TeX). For instance, we can now write

`\CSumlLimits{k}1\infty{\Cexp{x}k}` instead of the usual `\sum_{k=1}^\infty x^k` (1)

In the first form, we specify that you are applying a function (`CSumLimits` $\hat{=}$ Sum with Limits) to 4 arguments: (*i*) the bound variable k (that runs from) (*ii*) the number 1 (to) (*iii*) ∞ (to infinity summing the terms) (*iv*) `\Cexp{x}k` (i.e. x to the power k). In the second form, we merely specify hat LaTeX should draw a capital Sigma character (σ) with a lower index which is an equation $k = 1$ and an upper index ∞. Then it should place next to it an x with an upper index k.

Of course human readers (that understand the math) can infer the content structure from this presentation, but the LaTeXML converter (who does not understand the math) cannot, but we want to have the content MathML expression in Figure 1

```
<math xmlns="http://www.w3.org/1998/Math/MathML">
  <apply>
    <sum>
      <bvar><ci>k</ci></bvar>
      <lowlimit><cn>1</cn></lowlimit>
      <uplimit><infinit/></cn></uplimit>
      <apply><exp/><ci>x</ci><ci>k</ci></apply>
   </apply>
</math>
```

Example 1: Content MathML Form of $\sum_{k=1}^{\infty} x^k$

Obviously, a converter can infer this from the first LaTeX structure with the help of the curly braces that indicate the argument structure, but not from the

[4]The first one is standard e.g. in Germany and the US, the third one in France, and the last one in Russia

[5]However, semantic annotation will make the author more aware of the functional structure of the document and thus may in fact entice the author to use presentation in a more consistent way than she would usually have.

second (because it does not understand the math). The nice thing about the `cmathml` infrastructure is that you can still run LaTeX over the first form and get the same formula in the DVI file that you would have gotten from running it over the second form. That means, if the author is prepared to write the mathematical formulae a little differently in her LaTeX sources, then she can use them in XML and LaTeX at the same time.

2.1.1 `cmathml`: Encoding Content MathML in TeX/LaTeX

The `cmathml` package provides a set of macros that correspond to the K-14 fragment of mathematics (Kindergarten to undergraduate college level ($\hat{=}14^{th}$ grade)). We have already seen an example above in equation (1), where the content markup in TeX corresponds to a content MathML-expression (and can actually be translated to this by the LaTeXML system.) However, the content MathML vocabulary is fixed in the MathML specification and limited to the K-14 fragment; the notation of mathematics of course is much larger and extensible on the fly.

2.1.2 `presentation`: Flexible Presentation for Semantic Macros

The `presentation` package supplies an infrastructure that allows to specify the presentation of semantic macros, including preference-based bracket elision. This allows to markup the functional structure of mathematical formulae without having to lose high-quality human-oriented presentation in LaTeX. Moreover, the notation definitions can be used by MKM systems for added-value services, either directly from the sTeX sources, or after translation.

2.2 Mathematical Statements

2.2.1 `statements`: Extending Content Macros for Mathematical Notation

This package provides semantic markup facilities for mathematical statements like Theorems, Lemmata, Axioms, Definitions, etc. in sTeX files. This structure can be used by MKM systems for added-value services, either directly from the sTeX sources, or after translation.

2.2.2 `sproof`: Extending Content Macros for Mathematical Notation

This package supplies macros and environment that allow to annotate the structure of mathematical proofs in sTeX files. This structure can be used by MKM systems for added-value services, either directly from the sTeX sources, or after translation.

2.3 Context Markup for Mathematics

2.3.1 `modules`: Extending Content Macros for Mathematical Notation

This package supplies a definition mechanism for semantic macros and a non-standard scoping construct for them, which is oriented at the semantic depency relation rather than the document structure. This structure can be used by MKM systems for added-value services, either directly from the sTeX sources, or after translation.

2.4 Mathematical Document Classes

2.4.1 Connexions Modules

CNXLaTeX is a collection of LaTeX macros that allow to write Connexions modules without leaving the LaTeX workflow. Modules are authored in CNXLaTeX using only a text editor, transformed to PDF and proofread as usual. In particular, the LaTeX workflow is independent of having access to the Connexions system, which makes CNXLaTeX attractive for the initial version of single-author modules.

For publication, CNXLaTeX modules are transformed to CNXml via the LaTeXML translator and can be uploaded to the Connexions system.

2.4.2 OMDoc Documents

The `omdoc` package provides an infrastructure that allows to markup OMDoc documents in LaTeX. It provides `omdoc.cls`, a class with the and `omdocdoc.sty`

2.4.3 Slides and Presentations

We present a document class from which we can generate both course slides and course notes in a transparent way. Furthermore, we present a set of LaTeXML bindings for these, so that we can also generate OMDoc-based course materials, e.g. for inclusion in the ActiveMath system.

2.5 Conclusion

The sTeX collection provides a set of semantic macros that extends the familiar and time-tried LaTeX workflow in academics until the last step of Internet publication of the material. For instance, a Connexions module can be authored and maintained in LaTeX using a simple text editor, a process most academics in technical subjects are well familiar with. Only in a last publishing step (which is fully automatic) does it get transformed into the Xml world, which is unfamiliar to most academics.

Thus, sTeX can serve as a conceptual interface between the document author and MKM systems: Technically, the semantically preloaded LaTeX documents are transformed into the (usually Xml-based) MKM representation formats, but conceptually, the ability to semantically annotate the source document is sufficient.

The sTeX macro packages have been validated together with a case study [Koh05], where we semantically preload the course materials for a two-semester course in Computer Science at Jacobs University Bremen and transform them to the

OMDoc MKM format, so that they can be used in the ActiveMath system [MBA+01]. Another study of converting LaTeX materials for the Connexions project is under way.[2] EdNote(2)

2.6 Licensing, Download and Setup

[3] EdNote(3)

The sTeX packages and classes can be obtained as a self-documenting LaTeX packages: To obtain a package ⟨*package*⟩ download the files ⟨*package*⟩`.dtx` and ⟨*package*⟩`.ins` from

`https://svn.kwarc.info/repos/kwarc/projects/stex/sty/`⟨*package*⟩`/`

To extract the LaTeX package ⟨*package*⟩`.sty` and the LaTeXML bindings in ⟨*package*⟩`.ltxml`, run the LaTeX formatter on `cmathml.ins`, e.g. by typing `latex cmathml.ins` to a shell. To extract the documentation (the version of this document that goes with the extracted package) run the LaTeX formatter on `cmathml.dtx` e.g. by typing `latex` ⟨*package*⟩`.dtx` to a shell.

Usually, the sTeX distribution will also have the newest versions of the files ⟨*package*⟩`.sty`, ⟨*package*⟩`.ltxml`, and the documentation ⟨*package*⟩`.pdf` pregenerated for convenience, so they can be downloaded directly from the URL above.

To install the package, copy the file ⟨*package*⟩`.sty` somewhere, where TeX/LaTeX can find it and rebuild TeX's file name database. This is done by running the command `texhash` or `mktexlsr` (they are the same). In `MikTEX`, there is a menu option to do this.

3 Utilities

To simplify dealing with sTeX documents, we are providing a small collection of command line utilities, which we will describe here. For details and downloads go to `http://kwarc.info/projects/stex`.

`msplit` splits an sTeX file into smaller ones (one module per file)

`rf` computes the "reuse factor", i.e. how often sTeX modules are reused over a collection of documents

`sgraph` visualizes the module graph

`sms` computes the sTeX module signatures for a give sTeX file

`bms` proposes a sensible module structure for an un-annotated sTeX file

[2] EdNote: say some more
[3] EdNote: talk about licensing

cnx.dtx

4 Introduction

EdNote(4) The Connexions project is a[4]

The CNXml format — in particular the embeded content MathML — is hard to write by hand, so we provide a set of enviroments that allow to embed the CNXml document model into LaTeX.

5 The User Interface

This document is not a manual for the Connexions XML encoding, or a practical guide how to write Connexions modules. We only document the LaTeX bindings for CNXml and will presuppose experience with the format or familiarity with[5]. EdNote(5) Note that formatting CNXLaTeX documents with the LaTeX formatter does little to enforce the restrictions imposed by the CNXml document model. You will need to run the LaTeXML converter for that (it includes DTD validation) and any CNX-specific quality assurance tools after that. [6] EdNote(6)

The CNXLaTeX class makes heavy use of the `KeyVal` package, which is part of your LaTeX distribution. This allows to add optional information to LaTeX macros in the form of key-value pairs: A macro `\foo` that takes a KeyVal argument and a regular one, so a call might look like `\foo{bar}` (no KeyVal information given) or `\foo[key1=val1,...,keyn=valn]{bar}`, where `key1`,...,`keyn` are predefined keywords and values are LaTeX token sequences that do not contain comma characters (though they may contain blank characters). If a value needs to contain commas, then it must be enclosed in curly braces, as in `\foo[args={a,comma,separated,list}]`. Note that the order the key/value pairs appear in a KeyVal Argument is immaterial.

5.1 Document Structure

```
\documentclass{cnx}
\begin{document}
  \begin{cnxmodule}[name=Hello World,id=m4711]
    \begin{ccontent}
      \begin{cpara}[id=p01] Hello World\end{cpara}
    \end{ccontent}
  \end{cnxmodule}
\end{document}
```

Example 2: A Minimal CNXLaTeX Document

The first set of CNXLaTeX environments concern the top-level structure of the modules. The minimal Connexions document in LaTeX can be seen in Figure 2:

[4] EdNote: continue; copy from somewhere...
[5] EdNote: cite the relevant stuff here
[6] EdNote: talk about Content MathML and cmathml.sty somewhere

`cnxmodule` we still need the LaTeX document environment, then the `cnxmodule` environment contains the module-specific information as a KeyVal argument with the two keys: `id` for the module identifier supplied by the Connexions system) and `name` for the title of the module.

`ccontent` The `content`envionrment delineates the module content from the metadata (see Section 5.5). It is needed to make the conversion to CNXml simpler.

`c*section` CNXml knows three levels of sectioning, so the CNXLaTeX class supplies three as well: `csection`, `csubsection` and `csubsubsection`. In contrast to regular LaTeX, these are environments to keep the tight connection between the formats. These environments take an optional KeyVal argument with key `id` for the identifier and a regular argument for the title of the section (to be transformed into the CNXml `name` element).

`cpara, cnote` The lowest levels of the document structure are given by paragraphs and notes. The `cpara` and `cnote` environment take a KeyVal argument with the `id` key for identification, the latter also allows a `type` key for the note type (an unspecified string[7]). EdNote(7)

5.2 Mathematics

Mathematical formulae are integrated into text via the LaTeX math mode, i.e. wrapped in `$` characters or between `\(` and `\)` for inline mathematics and wrapped in `$$` or between `\[` and `\]` for display-style math. Note that CNXml expects Content MathML as the representation format for mathematical formulae, while run-of-the-mill LaTeX only specifies the presentation (i.e. the two-dimensional layout of formulae). The LaTeXML converter can usually figure out some of the content MathML from regular LaTeX, in other cases, the author has to specify it e.g. using the infrastructure supplied by the `cmathml` package.

`cequation` For numbered equations, CNXml supplies the `equation` element, for which CNXLaTeX provides the `cequation` environment. This environment takes a KeyVal argument with the `id` key for the (required) identifier.

5.3 Statements

CNXml provides special elements that make represnet various types of claims; we collectively call them statements.

`cexample` The `cexample` environment and `definition` elements take a KeyVal argument with key `id` for identification.

`crule, statement, proof` In CNXml, the `rule` element is used to represent a general assertion about the state of the world. The CNXLaTeX `rule`[8] environment is its CNXLaTeX counterpart. It takes a KeyVal attribute with the keys `id` for identification, `type` to specify the type of the assertion (e.g. "Theorem", "Lemma" or "Conjecture"), and `name`, if the assertion has a title. The body of the `crule` environment contains the statemnt of assertion in the `statement` environment and (optionally) a EdNote(8)

[7]EdNote: what are good values?
[8]EdNote: we have called this "crule", since "rule" is already used by TeX.

proof in the `proof` environment. Both take a KeyVal argument with an `id` key for identification.

```
\begin{crule}[id=prop1,type=Proposition]
   \begin{statement}[id=prop1s]
        Sample statement
   \end{statement}
   \begin{proof}[id=prop1p]
         Your favourite proof
   \end{proof}
\end{crule}
```

Example 3: A Basic crule Example

`definition, cmeaning` A definition defines a new technical term or concept for later use. The `definition` environment takes a KeyVal argument with the keys `id` for identification and `term` for the concept (definiendum) defined in this form. The definion text is given in the `cmeaning` environment[6], which takes a KeyVal argument with key `id` for identification. After the `cmeaning` environment, a `definition` can contain arbitrarily many `cexample`s.

```
\begin{definition}{term=term-to-be-defined, id=termi-def]
  \begin{cmeaning}[id=termi-meaning]
    {\term{Term-to-be-defined}} is defined as: Sample meaning
  \end{cmeaning}
\end{definition}
```

Example 4: A Basic `definition` and `cmeaning` Example

5.4 Connexions: Links and Cross-References

As the name CONNEXIONS already suggests, links and cross-references are very important for CONNEXIONS modules. CNXML provides three kinds of them. Module links, hyperlinks, and concept references.

`cnxn` Module links are speficied by the `\cnxn` macro, which takes a keyval argument with the keys `document`, `target`, and `strength`. The `document` key allows to specify the module identifier of the desired module in the repository, if it is empty, then the current module is intended. The `target` key allows to specify the document fragment. Its value is the respective identifier (given by its `id` attribute in CNXML or the `id` key of the corresponding environment in CNXLaTeX). Finally, the `strength` key allows to specify the relevance of the link.

The regular argument of the `\cnxn` macro is used to supply the link text.

`link` Hyperlinks can be specified by the `\link` macro in CNXLaTeX. It takes a

[6]we have called this `cmeaning`, sinc `menaning` is already taken by TeX

KeyVal argument with the key `src` to specify the URL of the link. The regular argument of the `\link` macro is used to supply the link text.

`term` The `\term` marcro can be used to specify the[9]

EdNote(9)

5.5 Metadata

Metadata is mostly managed by the system in CONNEXIONS, so we often do not need to care about it. On the other hand, it influences the system, so if we have work on the module extensively before converting it to CNXML, it may be worthwile specify some of the data in advance.

```
\begin{metadata}[version=2.19,
                  created=2000/07/21,revised=2004/08/17 22:07:27.213 GMT-5]
\begin{authorlist}
  \cnxauthor[id=miko,firstname=Michael,surname=Kohlhase,
              email=m.kohlhase@iu-bremen.de]
\end{authorlist}
\begin{keywordlist}\keyword{Hello}\end{keywordlist}
\begin{cnxabstract}
  A Minimal CNXLaTeX Document
\end{cnxabstract}
\end{metadata}
```

Example 5: Typical CNXLaTeX Metadata

`metadata` The `metadata` environment takes a KeyVal argument with the keys `version`, `created`, and `revised` with the obvious meanings. The latter keys take ISO 8601 norm representations for dates and times. Concretely, the format is `CCYY-MM-DDThh:mm:ss` where "`CC`" represents the century, "`YY`" the year, "`MM`" the month, and "`DD`" the day, preceded by an optional leading "`-`" sign to indicate a negative number. If the sign is omitted, "`+`" is assumed. The letter "`T`" is the date/time separator and "`hh`", "`mm`", "`ss`" represent hour, minutes, and seconds respectively.

`authorlist, maintainerlist` The lists of authors and maintainers can be specified in the `authorlist` and `maintainerlist` environments, which take no arguments.

`cnxauthor,maintainer` The entries on this lists are specified by the `\cnxauthor` and `\maintainer` macros. Which take a KeyVal argument specifying the individual. The `id` key is the identifier for the person, the `honorific`, `firstname`, `other`, `surname`, and `lineage` keys are used to specify the various name parts, and the `email` key is used to speficy the e-mail address of the person.

`keywordlist, keyword` The keywords are specified with a list of `keyword` macros, which take the respective keyword in their only argument, inside a `keyword` environment. Neither take any KeyVal arguments.

`cnxabstract` The abstract of a CONNEXIONS module is considered to be part of the meta-

[9] EDNOTE: continue, pending Chuck's investigation.

data. It is specified using the `cnxabstract` environment. It does not take any arguments.

5.6 Exercises

`cexercise, cproblem, csolution`

An exercise or problem in CONNEXIONS is specified by the `cexercise` environment, which takes an optional keyval argument with the keys `id` and `name`. It must contain a `cproblem` environment for the problem statement and a (possibly) empty set of `csolution` environments. Both of these take an optional keyval argument with the key `id`.

5.7 Graphics, etc.

`cfigure`

EdNote(10)

For graphics we will use the `cfigure`[10] macro, which provides a non-floating environment for including graphics into CNXML files. `cfigure` takes three arguments first an optional CNXML keys, then the keys of the `graphicx` package in a regular argument (leave that empty if you don't have any) and finally a path. So

```
\cfigure[id=foo,type=image/jpeg,caption=The first FOO]{width=7cm,height=2cm}{../images/f
```

EdNote(11)

Would include a graphic from the file at the path `../images/foo`, equip this image with a caption, and tell LaTeXML that[11] the original of the images has the MIME type `image/jpeg`.

[10]EDNOTE: probably better call it cgraphics
[11]EDNOTE: err, exactly what does it tell latexml?

cmathml.dtx

6 Introduction

This document describes the collection of semantic macros for content MathML and their LaTeXML bindings. These macros can be used to mark up mathematical formulae, exposing their functional/logical structure. This structure can be used by MKM systems for added-value services, either directly from the sTeX sources, or after translation. Even though it is part of the sTeX collection, it can be used independently. Note that this documentation of the package presupposes the discussion of the sTeX collection to be self-contained.

6.1 Encoding Content MathML in TeX/LaTeX

The `cmathml` packge presented here addresses part of transformation problem: representing mathematical formulae in the LaTeX workflow, so that content MathML representations can be derived from them. The underlying problem is that run-of-the-mill TeX/LaTeX only specifies the presentation (i.e. what formulae look like) and not their content (their functional structure). Unfortunately, there are no good methods (yet) to infer the latter from the former, but there are ways to get presentation from content.

The solution to this problem is to dump the extra work on the author (after all she knows what she is talking about) and give them the chance to specify the intended structure. The markup infrastructure supplied by the `cmathml` package lets the author do this without changing the visual appearance, so that the LaTeX workflow is not disrupted.

To use these `cmathml` macros in a LaTeX document, you will have to include the `cmathml` package using `\usepackage{cmathml}` somewhere in the document preamble. Then you can use the macros

```
$\Ceq{\Cexp{\Ctimes{\Cimaginaryi,\Cpi}},\Cuminus{\Ccn{1}}}$
```

which will result in $e^{i\pi} = -1$ when the document is formatted in LaTeX. If the document is converted to XML using the LaTeXML conversion tool, then the result will be content MathML representation:

```
<math xmlns="http://www.w3.org/1998/Math/MathML">
  <apply>
    <eq/>
    <apply>
      <exp/>
      <apply><times><imaginaryi/><pi/></times></apply>
    </apply>
    <apply><minus/><cn>1</cn></apply>
  </apply>
</math>
```

Example 6: Content MathML Form of $e^{i\pi} = -1$

6.2 Changing the TEX/LATEX Presentation

It is possible to change the default presentation (i.e. the result under LATEX formatting): The semantic macros only function as interface control sequences, which call an internal macro that does the actual presentation. Thus we simply have to redefine the internal macro to change the presentation. This is possible locally or globally in the following way:

```
\makeatletter
\gdef\CMathML@exp#1{exp(#1)}
\def\CMathML@pi{\varpi}
\makeatother
```

The first line is needed to lift the LATEX redefinition protection for internal macros (those that contain the character), and the last line restores it for the rest of the document. The second line has a *global* (i.e. the presentation will be changed from this point on to the end of the document.) redefinition of the presentation of the exponential function in the LATEX output. The third line has a *local* redefinition of the presentation (i.e. in the local group induced by LATEX's `begin`/`end` grouping or by TEX's grouping induced by curly braces). Note that the argument structure has to be respected by the presentation redefinitions. Given the redefinitions above, our equation above would come out as $exp(i\varpi) = -1$.

6.3 The Future: Heuristic Parsing

The current implementation of content MathML transformation from LATEX to MathML lays a heavy burden on the content author: the LATEX source must be semantically preloaded — the structure of the formulae must be fully annotated. In our example above, we had to write `\Ceq{A,B}` instead of the more conventional (and more legible) `A=B`.[12] EdNote(12)

The reason for this is that this keeps the transformation to content MathML very simple, predictable and robust at the expense of authoring convenience. The implementation described in this module should be considered as a first step and fallback solution only. Future versions of the LATEXML tool will feature more intelligent solutions for determining the implicit structure of more conventional mathematical notations (and LATEX representations), so that writing content MathML via LATEX will become less tedious.

However, such more advanced techniques usually rely on linguistic, structural, and semantic information about the mathematical objects and their preferred representations. They tend to be less predictable to casual users and may lead to semantically unexpected results.[13] EdNote(13)

[12]EdNote: come up with a good mixed example

[13]EdNote: talk about sTeX and extensibility in MathML/OpenMath/OMDoc

7 The User Interface

We will now tabulate the semantic macros for the Content MathML elements. We have divided them into modules based on the sectional structure of the MathML2 recommendation (2^{nd} edition). Before we go into the specific elements one-by-one, we will discuss some general properties of the `cmatml` macros and their LaTeXML bindings.

7.1 Generalities of the Encoding

The semantic macros provided by the `cmatml` package differ mainly in the way they treat their arguments. The simplest case are those for constants 7.12 that do not take any. Others take one, two, three, or even four arguments, which have to be TeX tokens or have to be wrapped in curly braces. For operators that are associative like addition the argument sequence is provided as a single TeX argument (wrapped in curly braces) that contains a comma-separated sequence of arguments (wrapped in curly braces where necessary).

`\Capply` The current setup of the `cmathml` infrastructure minimizes the need of specifying the MathML `apply` element, since the macros are all in applied form: As we have seen in the example in the Introduction 8, a macro call like `\Cexp{A}` corresponds to the application of the exponential function to some object, so the necessary `apply` elements in the MathML representation are implicit in the LaTeX formulation and are thus added by the transformation. Of course this only works, if the function is a content MathML element. Often, in mathematics we will have situations, where the function is a variable (or "arbitrary but fixed") function. Then the formula $f(x)$ represented as `$f(x)$` in TeX could (and sometimes will) be misunderstood by the Math parser as $f \cdot x$, i.e. a product of the number f with the number x, where x has brackets for some reason. In this case, we can
EdNote(14) disambiguate by using `\Capply{f}x`, which will also format as $f(x)$.[14]

By the same token, we do not need to represent the qualifier elements `condition` and `domainofapplication`[7], for binding operators. They are are folded into the special forms of the semantic macros for the binding operators below (the ones with the `Cond` and `DA` endings):

For operators that are associative, commutative, and idempotent (ACI i.e. bracketing, order, and multiplicity of arguments does not matter) MathML supplies the a special form of application as a binding operator (often called the corresponding "big operator)", which ranges over a whole set of arguments. For instance for the ACI operator $\cup$ for set uinon has the "big" operator for unions over collections of sets e.g. used in the power set $\bigcup_{S\subseteq T} S$ of a set T. In some cases, the "big" operators are provided independently by MathML, e.g. the ACI addition operator has the sum operator as a corresponding "big operator": $\sum_{x\in\mathbb{N}} x^i$ is the sum of the powers of x for all natural numbers. Where they are not, we will

[14]EdNote: what about n-ary functions?

[7]We do not support the `fn` element as it is deprecated in MathML2 and the `declare` and `sep` elements, since their semantic status is unclear (to the author, if you feel it is needed, please gripe to me).

supply extra macros in the `cmathml` package, e.g. the `\CUnion` macro as the big operator for `\Cunion`.

Finally, some of the binding operators have multiple content models flagged by the existence of various modifier elements. In these cases, we have provided different semantic macros for the different cases.

7.2 The Token Elements

The MathML token elements are very simple containers that wrap some presentation MathML text. The `csymbol` element is the extension element in MathML. It's content is the presentation of symbol, and it has a `definitionURL` attribute that allows to specify a URI that specifies the semantics of the symbol. This URL can be specified in an optional argument to teh `\Ccsymbol` macro, in accordance with usual mathematical practice, the `definitionURL` is not presented.

`\Ccn`
`\Cci`
`\Ccsymbol`

macro	args	Example	Result
`\Ccn`	token	`\Ccn{t}`	t
`\Cci`	token	`\Cci{t}`	t
`\Ccsymbol`	token, URI	`\Ccsymbol[http://w3.org]{t}`	t

Like the `\Ccsymbol` macro, all other macros in the `camthml` package take an optional argument[8] for the `definitionURL` attribute in the corresponding MathML element.

[8]This may change into a KeyVaL argument in future versionss of the `cmathml` package.

7.3 The Basic Content Elements

The basic elements comprise various pieces of the MathML infrastructure. Most of the semantic macros in this section are relatively uneventful.

`\Cinverse` `\Ccompose` `\Cident` `\Cdomain` `\Ccodomain` `\Cimage`

macro	args	Example	Result
`\Cinverse`	1	`\Cinverse{f}`	f^{-1}
`\Ccompose`	1	`\Ccompose{f,g,h}`	$f \circ g \circ h$
`\Cident`	0	`\Cident`	id
`\Cdomain`	1	`\Cdomain{f}`	$\mathrm{dom}(f)$
`\Ccodomain`	1	`\Ccodomain{f}`	$\mathrm{codom}(f)$
`\Cimage`	1	`\Cimage{f}`	$\mathbf{Im}(f)$

`\Clambda` `\ClambdaDA` `\Crestrict`

For the `lambda` element, we only have the `domainofapplication` element, so that we have three forms a λ-construct can have. The first one is the simple one where the first element is a bound variable. The second one restricts the appliccability of the bound variable via a `domainofapplication` element, while the third one does not have a bound variable, so it is just a function restriction operator.[15] EdNote(15)

macro	args	Example	Result
`\Clambda`	2	`\Clambda{x,y}{A}`	$\lambda(x, y, A)$
`\ClambdaDA`	3	`\ClambdaDA{x}{C}{A}`	$\lambda(x, y{:}\,C, A)$
`\Crestrict`	2	`\Crestrict{f}{S}`	$f\|_S$

`ccinterval` `cointerval` `ocinterval` `oointerval`

The `interval` constructor actually represents four types of intervals in MathML. Therefore we have four semantic macros, one for each combination of open and closed endings:

macro	args	Example	Result
`\Cccinterval`	2	`\Cccinterval{1}{2}`	$[1, 2]$
`\Ccointerval`	2	`\Ccointerval{1}{2}`	$[1, 2)$
`\Cocinterval`	2	`\Cocinterval{1}{2}`	$(1, 2]$
`\Coointerval`	2	`\Coointerval{1}{2}`	$(1, 2)$

`\Cpiecewise` `\Cpiece` `\Cotherwise`

The final set of semantic macros are concerned with piecewise definition of functions.

macro	args	Example	Result
`\Cpiecewise`	1	see below	see below
`\Cpiece`	2	`\Cpiece{A}{B}`	A if B
`\Cotherwise`	1	`\Cotherwise{B}`	1 *else*

For instance, we could define the abstract value function on the reals with the following markup

[15] EdNote: need ClambdaCond

Semantic Markup	Formatted

`\Ceq{\Cabs{x},`
`     \Cpiecewise{\Cpiece{\Cuminus{x}}{\Clt{x,0}}`
`                 \Cpiece{0}{\Ceq{x,0}}`
`                  \Cotherwise{x}}}`
	$\|x\| = \begin{cases} -x & \text{if } (x < 0) \\ 0 & \text{if } (x = 0) \\ x & else \end{cases}$

7.4 Elements for Arithmetic, Algebra, and Logic

This section introduces the infrastructure for the basic arithmetic operators. The first set is very simple

`\Cquotient` `\Cfactorial` `\Cdivide` `\Cminus` `\Cplus` `\Cpower` `\Crem` `\Ctimes` `\Croot`

macro	args	Example	Result
`\Cquotient`	2	`\Cquotient{1}{2}`	$\frac{1}{2}$
`\Cfactorial`	1	`\Cfactorial{7}`	7!
`\Cdivide`	2	`\Cdivide{1}{2}`	$1 \div 2$
`\Cminus`	2	`\Cminus{1}{2}`	$1 - 2$
`\Cplus`	1	`\Cplus{1}`	1
`\Cpower`	2	`\Cpower{x}{2}`	x^2
`\Crem`	2	`\Crem{7}{2}`	$7 \bmod 2$
`\Ctimes`	1	`\Ctimes{1,2,3,4}`	$1 \cdot 2 \cdot 3 \cdot 4$
`\Croot`	2	`\Croot{3}{2}`	$\sqrt[3]{2}$

The second batch below is sligtly more complicated, since they take a set of arguments. In the **`cmathml`** package, we treat them like associative operators, i.e. they act on a single argument that contains a sequence of comma-separated arguments[16]

EdNote(16)

`\Cmax` `\Cmin` `\Cgcd` `\Clcm`

macro	args	Example	Result
`\Cmax`	1	`\Cmax{1,3,6}`	$\max(1,3,6)$
`\Cmin`	1	`\Cmin{1,4,5}`	$\min(1,4,7)$
`\Cgcd`	1	`\Cgcd{7,3,5}`	$\gcd(7,3,5)$
`\Clcm`	1	`\Clcm{3,5,4}`	$\mathrm{lcm}(3,5,4)$

EdNote(17)

The operators for the logical connectives are associative as well[17]. Here, conjunction, (exclusive) disjunction are n-ary associative operators, therefore their semantic macro only has one TeX argument which contains a comma-separated list of subformulae.

`\Cand` `\Cor` `\Cxor` `\Cnot` `\Cimplies`

macro	args	Example	Result
`\Cand`	1	`\Cand{A,B,C}`	$A \wedge B \wedge C$
`\Cor`	1	`\Cor{A,B,C}`	$A \vee B \vee C$
`\Cxor`	1	`\Cxor{A,B,C}`	$A \oplus B \oplus C$
`\Cnot`	1	`\Cnot{A}`	$\neg A$
`\Cimplies`	2	`\Cimplies{A}{B}`	$A \Longrightarrow B$

`\CAndDA` `\CAndCond` `\COrDA` `\COrCond` `\CXorDA` `\CXorCond`

The following are the corresponding big operators, where appropriate.

[16] EdNote: implement this in the latexml side

[17] EdNote: maybe add some precedences here.

macro	args	Example	Result
`\CAndDA`	2	`\CAndDA\Cnaturalnumbers\phi`	$\bigwedge_{\mathbb{N}}\phi$
`\CAndCond`	3	`\CAndCond{x}{\Cgt{x}5}{\psi(x)}`	$\bigwedge_{x}x5$
`\COrDA`	2	`\COrDA\Cnaturalnumbers\phi`	$\bigvee_{\mathbb{N}}\phi$
`\COrCond`	3	`\COrCond{x}{\Cgt{x}5}{\psi(x)}`	$\bigvee_{x5}\psi(x)$
`\CXorDA`	2	`\CXorDA\Cnaturalnumbers\phi`	$\bigoplus_{\mathbb{N}}\phi$
`\CXorCond`	3	`\CXorCond{x}{\Cgt{x}5}{\psi(x)}`	$\bigoplus_{x5}\psi(x)$

`\Cforall` `\CforallCond` `\Cexists` `\CexistsCond`

The semantic macros for the quantifiers come in two forms: with- and without a condition qualifier. In a restricted quantification of the form $\forall x, C : A$, the bound variable x ranges over all values, such that C holds (x will usually occur in the condition C). In an unrestricted quantification of the form $\forall x : A$, the bound variable ranges over all possible values for x.

macro	args	Example	Result
`\Cforall`	2	`\Cforall{x,y}{A}`	$\forall x, y{:}\,A$
`\CforallCond`	3	`\CforallCond{x}{C}{A}`	$\forall x, C{:}\,A$
`\Cexists`	2	`\Cexists{x,y}{A}`	$\exists x, y{:}\,A$
`\CexistsCond`	3	`\CexistsCond{x}{C}{A}`	$\exists x, C{:}\,A$

`\Cabs` `\Cconjugate` `\Carg` `\Creal` `\Cimaginary` `\Cfloor` `\Cceiling`

The rest of the operators are very simple in structure.

macro	args	Example	Result
`\Cabs`	1	`\Cabs{x}`	$\lvert x\rvert$
`\Cconjugate`	1	`\Cconjugate{x}`	$\overline{x}$
`\Carg`	1	`\Carg{x}`	$\angle x$
`\Creal`	1	`\Creal{x}`	$\Re x$
`\Cimaginary`	1	`\Cimaginary{x}`	$\Im x$
`\Cfloor`	1	`\Cfloor{1.3}`	$\lfloor 1.3\rfloor$
`\Cceiling`	1	`\Cceiling{x}`	$\lceil x\rceil$

7.5 Relations

The relation symbols in MathML are mostly n-ary associative operators (taking a comma-separated list as an argument).

`\Ceq` `\Cneq` `\Cgt` `\Clt` `\Cgeq` `\Cleq` `\Cequivalent` `\Capprox` `\Cfactorof`

macro	args	Example	Result
`\Ceq`	1	`\CeqA,B,C`	$A = B = C$
`\Cneq`	2	`\Cneq{1}{2}`	$1 \neq 2$
`\Cgt`	1	`\Cgt{A,B,C}`	$A > B > C$
`\Clt`	1	`\Clt{A,B,C}`	$A < B < C$
`\Cgeq`	1	`\Cgeq{A,B,C}`	$A \geq B \geq C$
`\Cleq`	1	`\Cleq{A,B,C}`	$A \leq B \leq C$
`\Cequivalent`	1	`\Cequivalent{A,B,C}`	$A \equiv B \equiv C$
`\Capprox`	2	`\Capprox{1}{2}`	$1 \approx 1.1$
`\Cfactorof`	2	`\Cfactorof{7}{21}`	$7 \mid 21$

7.6 Elements for Calculus and Vector Calculus

The elements for calculus and vector calculus have the most varied forms.

The integrals come in four forms: the first one is just an indefinite integral over a function, the second one specifies the bound variables, upper and lower limits. The third one specifies a set instead of an interval, and finally the last specifies a bound variable that ranges over a set specified by a condition.

`\Cint`
`\CintLimits`
`\CintDA`
`\CintCond`

macro	args	Example	Result
`\Cint`	1	`\Cint{f}`	$\int f$
`\CintLimits`	4	`\CintLimits{x}{0}{\Cinfinit}{f(x)}`	$\int_0^\infty f(x)dx$
`\CintDA`	2	`\CintDA{\Creals}{f}`	$\int_{\mathbb{R}} f$
`\CintCond`	3	`\CintCond{x}{\Cin{x}{D}}{f(x)}`	$\int_{x\in D} f(x)dx$

`\Cdiff`
`\Cddiff`

The differentiation operators are used in the usual way: simple differentiation is represented by the `\Cdiff` macro which takes the function to be differentiated as an argument, differentiation with the *d*-notation is possible by the `\Cddiff`, which takes the bound varible[18] as the first argument and the function expression (in the bound variable) as a second argument. EdNote(18)

`\Cpartialdiff`

Partial Differentiation is specified by the `\Cpartialdiff` macro. It takes the overall degree as the first argument (to leave it out, just pass the empty argument). The second argument is the list of bound variables (with their degrees; see below), and the last the function expression (in these bound variables). To specify the respective degrees of differentiation on the variables, we use the `\Cdegree` macro, which takes two arguments (but no optional argument), the first one is the degree (a natural number) and the second one takes the variable. Note that the overall degree has to be the sum of the degrees of the bound variables.

`\Cdegree`

macro	args	Example	Result
`\Cdiff`	1	`\Cdiff{f}`	f'
`\Cddiff`	2	`\Cddiff{x}{f}`	$\frac{df(x)}{dx}$
`\Cpartialdiff`	3	`\Cpartialdiff{3}{x,y,z}{f(x,y)}`	$\frac{\partial^3}{\partial x,y,z} f(x,y)$
`\Cpartialdiff`	3	`\Cpartialdiff{7}` `{\Cdegree{2}{x},\Cdegree{4}{y},z}` `{f(x,y)}`	$\frac{\partial^7}{\partial 2^x,4^y,z} f(x,y)$

`\Climit`
`\ClimitCond`

For content MathML, there are two kinds of limit expressions: The simple one is specified by the `\Climit` macro, which takes three arguments: the bound variable, the target, and the limit expression. If we want to place additional conditions on the limit construction, then we use the `\ClimitCond` macro, which takes three arguments as well, the first one is a sequence of bound variables, the second one is the condition, and the third one is again the limit expression.

`\Ctendsto`
`\CtendstoAbove`

If we want to speak qualitatively about limit processes (e.g. in the condition of

\CtendstoBelow

a `\ClimitCond` expression), then can use the MathML `tendsto` element, which is represented by the `\Ctendsto` macro, wich takes two expressions arguments. In MathML, the `tendsto` element can be further specialized by an attribute to indicate the direction from which a limit is approached. In the `cmathml` package, we supply two additional (specialized) macros for that: `\CtendstoAbove` and `\CtendstoBelow`.

macro	args	Example	Result
`\Climit`	3	`\Climit{x}{0}{\Csin{x}}`	$\lim_{x\to 0}\sin(x)$
`\ClimitCond`	3	`\ClimitCond{x}{\Ctendsto{x}{0}}{\Ccos{x}}`	$\lim_{x\to 0}\cos(x)$
`\Ctendsto`	2	`\Ctendsto{f(x)}{2}`	$f(x)\to 2$
`\CtendstoAbove`	2	`\CtendstoAbove{x}{1}`	$x\searrow 1$
`\CtendstoBelow`	2	`\CtendstoBelow{x}{2}`	$x\nearrow 2$

\Cdivergence
\Cgrad
\Ccurl
\Claplacian

macro	args	Example	Result
`\Cdivergence`	1	`\Cdivergence{A}`	$\nabla\cdot A$
`\Cgrad`	1	`\Cgrad{\Phi}`	$\nabla\Phi$
`\Ccurl`	1	`\Ccurl{\Xi}`	$\nabla\times\Xi$
`\Claplacian`	1	`\Claplacian{A}`	$\nabla^2 A$

7.7 Sets and their Operations

\Cset
\Clist
\CsetDA
\CsetRes
\CsetCond

The `\Cset` macros is used as the simple finite set constructor, it takes one argument that is a comma-separated sequence of members of the set. `\CsetRes` allows to specify a set by restricting a set of variables, and `\CsetCond` is the general form of the set construction.[19]

EdNote(19)

macro	args	Example	Result
`\Cset`	1	`\Cset{1,2,3}`	$\{1,2,3\}$
`\CsetRes`	2	`\CsetRes{x}{\Cgt{x}5}`	$\{x\|x5\}$
`\CsetCond`	3	`\CsetCond{x}{\Cgt{x}5}{\Cpower{x}3}`	$\{x5\|x^3\}$
`\CsetDA`	3	`\CsetDA{x}{\Cgt{x}5}{S_x}}`	$\{x\in x5\|S_x\}$
`\Clist`	1	`\Clist{3,2,1}`	list$(3,2,1)$

\Cunion
\Cintersect
\Ccartesianproduct
\Csetdiff
\Ccard
\Cin
\Cnotin

macro	args	Example	Result
`\Cunion`	1	`\Cunion{S,T,L}`	$S\cup T\cup L$
`\Cintersect`	1	`\Cintersect{S,T,L}`	$S\cap T\cap L$
`\Ccartesianproduct`	1	`\Ccartesianproduct{A,B,C}`	$A\times B\times C$
`\Csetdiff`	2	`\Csetdiff{S}{L}`	$S\setminus L$
`\Ccard`	1	`\Ccard{\Cnaturalnumbers}`	$\#\mathbb{N}$
`\Cin`	2	`\Cin{a}{S}`	$a\in S$
`\Cnotin`	2	`\Cnotin{b}{S}`	$b\notin S$

[18]EdNote: really only one?
[19]EdNote: need to do this for lists as well? Probably

\CUnionDA
\CUnionCond
\CIntersectDA
\CIntersectCond
\CCartesianproductDA
\CCartesianproductCond

The following are the corresponding big operators for the first three binary ACI functions.

macro	args	Example	Result
`\CUnionDA`	2	`\CUnionDA\Cnaturalnumbers{S_i}`	$\bigwedge_{\mathbb{N}} S_i$
`\CUnionCond`	3	`\CUnionCond{x}{\Cgt{x}5}{S_x}}`	$\bigwedge_{x} x5$
`\CIntersectDA`	2	`\CIntersectDA\Cnaturalnumbers{S_i}`	$\bigvee_{\mathbb{N}} S_i$
`\CIntersectCond`	3	`\CIntersectCond{x}{\Cgt{x}5}{S_x}`	$\bigvee_{x5} S_x$
`\CCartesianproductDA`	2	`\CCartesianproductDA\Cnaturalnumbers{S_i}`	$\bigoplus_{\mathbb{N}} S_i$
`\CCartesianproductCond`	3	`\CCartesianproductCond{x}{\Cgt{x}5}{S_x}`	$\bigoplus_{x5} S_x$

\Csubset
\Cprsubset
\Cnotsubset
\Cnotprsubset

For the set containment relations, we are in a somewhat peculiar situation: content MathML only supplies the subset side of the reations and leaves out the superset relations. Of course they are not strictly needed, since they can be expressed in terms of the subset relation with reversed argument order. But for the `cmathml` package, the macros have a presentational side (for the LaTeX workflow) and a content side (for the LaTeXML converter) therefore we will need macros for both relations.

macro	args	Example	Result
`\Csubset`	1	`\Csubset{S,T,K}`	$S \subseteq T \subseteq K$
`\Cprsubset`	1	`\Cprsubset{S,T,K}`	$S \subset T \subset K$
`\Cnotsubset`	2	`\Cnotsubset{S}{K}`	$S \not\subseteq K$
`\Cnotprsubset`	2	`\Cnotprsubset{S}{L}`	$S \not\subset L$

\Csupset
\Cprsupset
\Cnotsupset
\Cnotprsupset

The following set of macros are presented in LaTeX as their name suggests, but upon transformation will generate content markup with the MathML elements (i.e. in terms of the subset relation).

macro	args	Example	Result
`\Csupset`	1	`\Csupset{S,T,K}`	$S \supseteq T \supseteq K$
`\Cprsupset`	1	`\Cprsupset{S,T,K}`	$S \supset T \supset K$
`\Cnotsupset`	2	`\Cnotsupset{S}{K}`	$S \not\supseteq K$
`\Cnotprsupset`	2	`\Cnotprsupset{S}{L}`	$S \not\supset L$

7.8 Sequences and Series

\CsumLimits
\CsumCond
\CsumDA
\CprodLimist
\CprodCond
\CprodDA

macro	args	Example	Result
`\CsumLimits`	4	`\CsumLimits{i}{0}{50}{x^i}`	$\sum_{i=0}^{50} x^i$
`\CsumCond`	3	`\CsumCond{i}{\Cintegers}{i}`	$\sum_{i\in\mathbb{Z}} i$
`\CsumDA`	2	`\CsumDA{\Cintegers}{f}`	$\sum_{\mathbb{Z}} f$
`\CprodLimits`	4	`\CprodLimits{i}{0}{20}{x^i}`	$\prod_{i=202^{20}x^i}$
`\CprodCond`	3	`\CprodCond{i}{\Cintegers}{i}`	$\prod_{i\in\mathbb{Z}} i$
`\CprodDA`	2	`\CprodDA{\Cintegers}{f}`	$\prod_{f}$

7.9 Elementary Classical Functions

\Csin
\Ccos
\Ctan
\Csec
\Ccsc
\Ccot

macro	args	Example	Result
`\Csin`	1	`\Csin{x}`	$\sin(x)$
`\Ccos`	1	`\Ccos{x}`	$\cos(x)$
`\Ctan`	1	`\Ctan{x}`	$\tan(x)$
`\Csec`	1	`\Csec{x}`	$\sec(x)$
`\Ccsc`	1	`\Ccsc{x}`	$\csc(x)$
`\Ccot`	1	`\Ccot{x}`	$\cot(x)$

\Csinh
\Ccosh
\Ctanh
\Csech
\Ccsch
\Ccoth

macro	args	Example	Result
`\Csinh`	1	`\Csinh{x}`	$\sinh(x)$
`\Ccosh`	1	`\Ccosh{x}`	$\cosh(x)$
`\Ctanh`	1	`\Ctanh{x}`	$\tanh(x)$
`\Csech`	1	`\Csech{x}`	$\mathrm{sech}(x)$
`\Ccsch`	1	`\Ccsch{x}`	$\mathrm{csch}(x)$
`\Ccoth`	1	`\Ccoth{x}`	$\coth(x)$

\Carcsin
\Carccos
\Carctan
\Carcsec
\Carccsc
\Carccot

macro	args	Example	Result
`\Carcsin`	1	`\Carcsin{x}`	$\arcsin(x)$
`\Carccos`	1	`\Carccos{x}`	$\arccos(x)$
`\Carctan`	1	`\Carctan{x}`	$\arctan(x)$
`\Carccosh`	1	`\Carccosh{x}`	$\mathrm{arccosh}(x)$
`\Carccot`	1	`\Carccot{x}`	$\mathrm{arccot}(x)$

\Carcsinh
\Carccosh
\Carctanh
\Carcsech
\Carccsch
\Carccoth

macro	args	Example	Result
`\Carccoth`	1	`\Carccoth{x}`	$\mathrm{arccoth}(x)$
`\Carccsc`	1	`\Carccsc{x}`	$\mathrm{arccsc}(x)$
`\Carcsinh`	1	`\Carcsinh{x}`	$\mathrm{arcsinh}(x)$
`\Carctanh`	1	`\Carctanh{x}`	$\mathrm{arctanh}(x)$
`\Cexp`	1	`\Cexp{x}`	$\exp(x)$
`\Cln`	1	`\Cln{x}`	$\ln(x)$
`\Clog`	2	`\Clog{5}{x}`	$\log_5(x)$

7.10 Statistics

The only semantic macro that is non-standard in this module is the one for the `moment` and `momentabout` elements in MathML. They are combined into the semantic macro `CmomentA`; its first argument is the degree, its second one the point in the distribution, the moment is taken about, and the third is the distribution.

\Cmean
\Csdev
\Cvar
\Cmedian
\Cmode
\Cmoment
\CmomentA

macro	args	Example	Result
`\Cmean`	1	`\Cmean{X}`	$\mathrm{mean}(X)$
`\Csdev`	1	`\Csdev{X}`	$\mathrm{std}(X)$
`\Cvar`	1	`\Cvar{X}`	$\mathrm{var}(X)$
`\Cmedian`	1	`\Cmedian{X}`	$\mathrm{median}(X)$
`\Cmode`	1	`\Cmode{X}`	$\mathrm{mode}(X)$
`\Cmoment`	3	`\Cmoment{3}{X}`	$\langle X^3\rangle$
`\CmomentA`	3	`\CmomentA{3}{p}{X}`	$\langle p^3\rangle X$

7.11 Linear Algebra

In these semantic macros, only the matrix constructor is unusual; instead of constructing a matrix from `matrixrow` elements like MathML does, the macro follows the TEX/LATEX tradition allows to give a matrix as an array. The first argument of the macro is the column specification (it will only be used for presentation purposes), and the second one the rows.

`\Cvector`
`\Cmatrix`
`\Cdeterminant`
`\Ctranspose`
`\Cselector`
`\Cvectorproduct`
`\Cscalarproduct`
`\Couterproduct`

macro	args	Example	Result
`\Cvector`	1	`\Cvector{1,2,3}`	$(1,2,3)$
`\Cmatrix`	2	`\Cmatrix{ll}{1 & 2\\ 3 & 4}`	$\begin{pmatrix} 1 & 2 \\ 3 & 4 \end{pmatrix}$
`\Cdeterminant`	1	`\Cdeterminant{A}`	$\|A\|$
`\Ctranspose`	1	`\Ctranspose{A}`	$A^\top$
`\Cselector`	2	`\Cselector{A}{2}`	A_2
`\Cvectproduct`	2	`\Cvectproduct{\phi}{\psi}`	$\phi\cdot\psi$
`\Cscalarproduct`	2	`\Cscalarproduct{\phi}{\psi}`	$\phi\psi$
`\Couterproduct`	2	`\Couterproduct{\phi}{\psi}`	$\phi\times\psi$

7.12 Constant and Symbol Elements

The semantic macros for the MathML constant and symbol elements are very simple, they do not take any arguments, and their name is just the MathML element name prefixed by a capital C.

`\Cintegers`
`\Creals`
`\Crationals`
`\Ccomplexes`
`\Cprimes`

macro	args	Example	Result
`\Cintegers`		`\Cintegers`	$\mathbb{Z}$
`\Creals`		`\Creals`	$\mathbb{R}$
`\Crationals`		`\Crationals`	$\mathbb{Q}$
`\Cnaturalnumbers`		`\Cnaturalnumbers`	$\mathbb{N}$
`\Ccomplexes`		`\Ccomplexes`	$\mathbb{C}$
`\Cprimes`		`\Cprimes`	$\mathbb{P}$

`\Cexponentiale`
`\Cimaginaryi`
`\Ctrue`
`\Cfalse`
`\Cemptyset`
`\Cpi`
`\Ceulergamma`
`\Cinfinit`

macro	args	Example	Result
`\Cexponemtiale`		`\Cexponemtiale`	e
`\Cimaginaryi`		`\Cimaginaryi`	i
`\Cnotanumber`		`\Cnotanumber`	NaN
`\Ctrue`		`\Ctrue`	true
`\Cfalse`		`\Cfalse`	false
`\Cemptyset`		`\Cemptyset`	$\emptyset$
`\Cpi`		`\Cpi`	π
`\Ceulergamma`		`\Ceulergamma`	γ
`\Cinfinit`		`\Cinfinit`	∞

7.13 Extensions

Content MathML does not (even though it claims to cover M-14 Math) symbols for all the common mathematical notions. The `cmathmlx` attempts to collect these and provide TEX/LATEX and LATEXML bindings.

`\Ccomplement`

macro	args	Example	Result
`\Ccomplement`	1	`\Ccomplement{\Cnaturalnumbers}`	$\mathbb{N}^c$

modules.dtx

8 Introduction

Following general practice in the TEX/LATEX community, we use the term "semantic macro" for a macro whose expansion stands for a mathematical object, and whose name takes up the name of the mathematical object. This can range from simple definitions like `\def\Reals{{\mathbb R}}` for indivicual mathematical objects to more complex (functional) ones object constructors like `\def\SmoothFunctionsOn#1{{\cal{C}}^\infty(#1)}`. Semantic macros are traditionally used to make TEX/LATEX code more portable. However, the TEX/LATEX scoping model (macro definitions are scoped either in the local group or until the rest of the document), does not mirror mathematical practice, where notations are scoped by mathematical environments like statements, theories, or such.

9 The User Interface

The main contributions of the `modules` package are the `module` environment, which allows for lexical scoping of semantic macros with inheritance and the `\symdef` macro for declaration of semantic macros that underly the `module` scoping.

9.1 Modules

`module` The `module` environment takes an optional `KeyVal` argument. Currently, only the `id` key is supported for specifying the identifier of a module (also called the module name).

`\importmodule` A module introduced by `\begin{module}[id=foo]` restricts the scope the semantic macros defined by the `\symdef` form to the end of this module given by the corresponding `\end{module}`, and to any other `module` environments that import them by a `\importmodule{foo}` directive. If the module `foo` contains `\importmodule` directives of its own, these are also exported to the importing module. Thus `\importmodule` induces the semantic inheritance relation and `usesqualified`[20] for macros imported with a prefix (this is used whenever we have conflicting names for macros inherited from different modules). EdNote(20)

9.2 Semantic Macros

`\symdef` A call to the `\symdef` macro has the general form

`\symdef[`⟨*keys*⟩`]{`⟨*cseq*⟩`}[`⟨*args*⟩`]{`⟨*definiens*⟩`}`

where ⟨*cseq*⟩ is a control sequence (the name of the semantic macro) ⟨*args*⟩ is a number between 0 and 9 for the number of arguments ⟨*definiens*⟩ is the token sequence used in macro expansion for ⟨*cseq*⟩. Finally ⟨*keys*⟩ is a keyword list that further specifies the semantic status of the defined macro.

A key `local` can be added to ⟨*keys*⟩ to specify that the symbol is local to the module and is invisible outside. The key-value pair `aliases=`⟨*symname*⟩

[20]EdNote: do an importqualified as well

specifies that the defined symbol ⟨*cseq*⟩ is a presentational variant of the symbol ⟨*symname*⟩.

Finallly, the keys `cmml`, `cattrs`, and `definitionURL` can be used to specify the Content-MathML encoding of the symbols. They key-value pair `cmml=`⟨*elt*⟩ specifies that the semantic macro corresponds to the Content-MathML element with the name ⟨*elt*⟩, `cattrs=`⟨*attrtring*⟩ its argument string and `definitionURL` allows to specify the `definitionURL` attribute on that element. The most common case will be a symbol definition of the following form:

`\symdef[cmml=csymbol,definitionURL=`⟨*URI*⟩`]{`⟨*cseq*⟩`}[`⟨*args*⟩`]{`⟨*definiens*⟩`}`

where ⟨*URI*⟩ is the URI pointing to the location of the Xml file generated from the current LaTeX file.

`\abbrdef` The `\abbrdef` macro is a variant of `\symdef` that is only different in semantics, not in presentation. An abbreviative macro is like a semantic macro, and underlies the same scoping and inheritance rules, but it is just an abbreviation that is meant to be expanded, it does not stand for an atomic mathematical object.

We will use a simple module for natural number arithmetics as a running example. It defines exponentiation and summation as new concepts while drawing on the basic operations like + and − from LaTeX. In our example, we will define a semantic macro for summation `\Sumfromto`, which will allow us to express an expression like $\sum i = 1^n x^i$ as `\Sumfromto{i}1n{2i-1}` (see Example 7 for an example). In this example we have also made use of a local semantic symbol for n, which is treated as an arbitrary (but fixed) symbol.

```
\begin{module}[id=arith]
  \symdef{Sumfromto}[4]{\sum_{#1=#2}^{#3}{#4}}
  \symdef[local]{arbitraryn}{n}
  What is the sum of the first $\arbitraryn$ odd numbers, i.e.
  $\Sumfromto{i}1\arbitraryn{2i-1}?$
\end{module}
```

is formatted by sTeX to

What is the sum of the first n odd numbers, i.e. $\sum_{i=1}^{n} 2i-1$?

Example 7: Semantic Markup in a `module` context

9.3 Dealing with multiple Files

The infrastructure presented above works well if we are dealing with small files or small collections of modules. In reality, collections of modules tend to grow, get reused, etc, making it much more difficult to keep everything in one file. This general trend towards increasing enthropy is aggravated by the fact that modules are very self-contiained objects that are ideal for re-used. Therefore in the absence of a content management system for LaTeX document (fragments), module collections tend to develop towards the "one module one file" rule, which leads to situations with lots and lots of little files.

Moreover, most mathematical documents are not self-contained, i.e. they do not build up the theory from scratch, but pre-suppose the knowledge (and notation) from other documents. In this case we want to make use of the semantic macros from these prerequisite documents without including their text into the current document. One way to do this would be to have LaTeX read the prerequisite documents without producing output. For efficiency reasons, sTeX chooses a different route. It comes with a utility `sms` (see Section 3) that exports the modules and macros defined inside them from a particular document and stores them inside `.sms` files. This way we can avoid overloading LaTeX with useless information, while retaining the important information which can then be imported in a more efficient way.

`\importmodule` For such situations, the `\importmodule` macro can be given an optional first argument that is a path to a file that contains a path to the module file, whose module definition (the `.sms` file) is read. Note that the `\importmodule` macro can be used to make module files truly self-contained. To arrive at a file-based content management system, it is good practice to reuse the module identifiers as module names and to prefix module files with corresponding `\importmodule` statements that pre-load the corresponing module files.

```
\begin{module}[id=foo]
\importmodule[../other/bar]{bar}
\importmodule[../mycolleaguesmodules]{baz}
\importmodule[../other/bar]{foobar}
  ...
\end{module}
```

Example 8: Self-contained Modules via `importmodule`

In Example 8, we have shown the typical setup of a module file. The `\importmodule` macro takes great care that files are only read once, as sTeX allows multiple inheritance and this setup would lead to an exponential (in the module inheritance depth) number of file loads.

Note that the recursive (depth-first) nature of the file loads induced by this setup is very natural, but can lead to problems with the depth of the file stack in the TeX formatte (it is usually set to something like 15). Therefore, it may be necessary to circumvent the recursive load pattern providing (logically spurious) `\importmodule` commands. Consider for instance module `bar` in Example 8, say that `bar` already has load deph 15, then we cannot naivedly import it in this way. If module `bar` depended say on a module `base` on the critical load path, then `\requiremodules` we could add a statement `\requiremodules{../base}` in the second line. This would load the modules from `../base.sms` in advance (uncritical, since it has load depth 10), so that it would not have to be re-loaded in the critical path of the module `foo`. Solving the load depth problem.

9.4 Including Externally Defined Semantic Macros

In some cases, we use an existing LaTeX macro package for typesetting objects that have a conventionalized mathematical meaning. In this case, the macros are "semantic" even though they have not been defined by a `\symdef`. This is no problem, if we are only interested in the LaTeX workflow. But if we want to e.g. transform them to OMDoc via LaTeXML, the LaTeXML bindings will need to contain references to an OMDoc theory that semantically correponds to the LaTeX package. In particular, this theory will have to be imported in the generated OMDoc file to make it OMDoc-valid.

`\requirepackage` To deal with this situation, the `modules` package provides the `\requirepackage` macro. It takes two arguments: a package name, and a URI of the corresponing OMDoc theory. In the LaTeX workflow this macro behaves like a `\usepackage` on the first argument, except that it can — and should — be used outside the LaTeX preamble. In the LaTeXML workflow, this loads the LaTeXML bindings of the package specified in the first argument and generates an appropriate `imports` element using the URI in the second argument.

statements.dtx

10 Introduction

The motivation for the `statemets` package is very similar to that for semantic macros in the `modules` package: We want to annotate the structural semantic properties of statements in the source, but present them as usual in the formatted documents. In contrast to the case for mathematical objects, the repertoire of mathematical statements and their structure is more or less fixed.

This structure can be used by MKM systems for added-value services, either directly from the sTeX sources, or after translation. Even though it is part of the sTeX collection, it can be used independently, like it's sister package `sproofs`.

sTeX is a version of TeX/LaTeX that allows to markup TeX/LaTeX documents semantically without leaving the document format, essentially turning TeX/LaTeX into a document format for mathematical knowledge management (MKM).

11 The User Interface

`assertion` All the statements are marked up as envioronments, that take a `KeyVal` argument that allows to annotate semantic information. For instance, instead of providing environments for "Theorem", "Lemma", "Proposition",... we have a single `assertion` environment that generalizes all of these, and takes a `type` key that allows to specify the "type". So instead of `\begin{Lemma}`we have to write `\begin{assertion}[type=Lemma]`(see Example 9 for an example).[21] EdNote(21)

```
\begin{assertion}[id=sum-over-odds,type=Lemma]
  $\sum_{i=1}^n{2i-1}=n^2$
\end{assertion}
```

will lead to the result

Lemma: $\sum_{i=1}^n 2i-1=n^2$

Example 9: Semantic Markup for a Lemma in a `module` context

Whether we will see the keyword "Lemma" will depend on the value of the optional `display` key. In all of the `assertion` environments, the presentation expectation is that the text will be presented in italic font. Generally, we distinguish two forms of statements:

block statements have explicit discourse markers that delimit their content in the surrounding text, e.g. the boldface word "**Theorem:**" as a start marker and a little line-end box as an end marker of a proof.

flow statements do not have explicit markers, they are interspersed with the surrounding text.

Since they have the same semantic status, they must both be marked up, but styled differently. We distinguis between these two presentational forms with the

[21] EdNote: talk about package options here! Draft mode,...

`display` key, which is allowed on all statement environments. If it has the value `block` (the default), then the statement will be presented in a paragraph of its own, have explicit discourse markers for its begin and end, possibly numbering, etc. If it has the value `flow`, then no extra presentation will be added[22] the semantic information is invisible to the reader. EdNote(22)

Another key that is present on all statement environments in the `id` key it allows to identify the statement with a name.

`axiom` The `axiom` environment is similar to `assertion`, but the content has a different ontological status: axioms are assumed without (formal) justification, whereas assertions are expeceted to be justified from other assertions, axioms or definitions.

`definition` `\definiendum` The `definition` environment is used for marking up mathematical definitions. Its peculiarity is that it defines (i.e. gives a meaning to) new mathematical concepts or objects. Theseare identified by the `definiendum` macro, which takes two arguments. The first one is the system name of the symbol defined (for reference via `\termin`), the second one is the text that is to be emphasized in the presentation. Note that the `\definiendum` macro can only be used inside the `definition` environment. If you find yourself in a situation where you want to use it outside, you will most likely want to wrap the apporpriate text fragment in a `\begin{definition}[display=flow]` ... and `\end{definition}`.[23] EdNote(23)

`\termin` If we have defined a concept with the `\definiendum` macro, then we can mark up other occurrences of the term as referring to this concept. Note that this process cannot be fully automatized yet, since that would need advanced lanauge technology to get around problems of disambiguation, inflection, and non-contiguous phrases[9]. Therefore, the `\termin` can be used to make this information explicit.

`simpleDef` The `simpleDef` environment is a statement environment for simple definitions, which introduce a new symbol that abbreviates another concept. The envioronment takes an argument for the new concept

`example` The `example` environment is a generic statement envionment, except that the `for` key should be given to specify the identifier what this is an example for. The `example` environment also expcets a `type` key to be specified, so that we know whether this is an example or a counterexample[24] EdNote(24)

`\defemph` The `\defemph` macro is a configuration hook that allows to specify the style of presentation of the definiendum. By default, it is set to `\bf` as a fallback, since we can be sure that this is always available. It can be customized by redefinition: For instance `\renewcommand{\defemph}[1]{\emph{#1}}`, changes the default behavior to italics.

`\termemph` The `\termenph` macro does the same for the style for `\termin`, it is empty by default. Note the term might carry an implicit hyperreference to the defining occurrance and that the presentation engine might mark this up, changing this behavior.

`\stDMemph` The `\stDMemph` macro does the same for the style for the markup of the dis-

[22] EdNote: in the flow case, the text should not be made italic; implement this!

[23] EdNote: need to leave hypertargets on the definiendum, so that we can crosslink

[9] We do have a program that helps annotate larger text collections spotting the easy cases; see http://kwarc.info/projects/stex and look for the program `termin`.

[24] EdNote: think about this some more

course markers like “Theorem”. If it is not defined, it is set to `\bf`; that allows to preset this in the class file.

sproof.dtx

12 Introduction

The sproof (semantic proofs) package supplies macros and environment that allow to annotate the structure of mathematical proofs in sTEX files. This structure can be used by MKM systems for added-value services, either directly from the sTEX sources, or after translation. Even though it is part of the sTEX collection, it can be used independently, like it's sister package statements.

sTEX is a version of TEX/LATEX that allows to markup TEX/LATEX documents semantically without leaving the document format, essentially turning TEX/LATEX into a document format for mathematical knowledge management (MKM).

EdNote(25) We will go over the general intuition by way of our running example (see Figure 10 for the source and Figure 11 for the formatted result).[25]

13 The User Interface

13.1 Proofs and Proof steps

sproof The proof environment is the main container for proofs. It takes an optional KeyVal argument that allows to specify the id (identifier) and for (for which assertion is this a proof) keys. The regular argument of the proof environment contains an introductory comment, that may be used to announce the proof style. The proof environment contains a sequence of \step, proofcomment, and pfcases environments that are used to markup the proof steps. The proof environment has a variant Proof, which does not use the proof end marker. This is convenient, if a proof ends in a case distinction, which brings it's own proof end marker with sProof it. The Proof environment is a variant of proof that does not mark the end of a proof with a little box; presumably, since one of the subproofs already has one and then a box supplied by the outer proof would generate an otherwise empty \sproofidea line. The \sproofidea macro allows to give a one-paragraph description of the proof idea.

spfstep Regular proof steps are marked up with the step environment, which takes an optional KeyVal argument for annotations. A proof step usually contains a local assertion (the text of the step) together with some kind of evidence that this can be derived from already established assertions.

Note that both \premise and \justarg can be used with an empty second argument to mark up premises and arguments that are not explicitly mentioned in the text.

13.2 Justifications

justification This evidence is marked up with the justification environment in the sproof

[25]EdNote: talk a bit more about proofs and their structure,... maybe copy from OMDoc spec.

```
\begin{sproof}[id=simple-proof,for=sum-over-odds]
   {We prove that $\sum_{i=1}^n{2i-1}=n^{2}$ by induction over $n$}
  \begin{spfcases}{For the induction we have to consider the following cases:}
   \begin{spfcase}{$n=1$}
    \begin{spfstep}[display=flow] then we compute $1=1^2$\end{spfstep}
   \end{spfcase}
   \begin{spfcase}{$n=2$}
      \begin{sproofcomment}[display=flow]
        This case is not really necessary, but we do it for the
        fun of it (and to get more intuition).
      \end{sproofcomment}
      \begin{spfstep}[display=flow] We compute $1+3=2^{2}=4$.\end{spfstep}
   \end{spfcase}
   \begin{spfcase}{$n>1$}
      \begin{spfstep}[type=assumption,id=ind-hyp]
        Now, we assume that the assertion is true for a certain $k\geq 1$,
        i.e. $\sum_{i=1}^k{(2i-1)}=k^{2}$.
      \end{spfstep}
      \begin{sproofcomment}
        We have to show that we can derive the assertion for $n=k+1$ from
        this assumption, i.e. $\sum_{i=1}^{k+1}{(2i-1)}=(k+1)^{2}$.
      \end{sproofcomment}
      \begin{spfstep}
        We obtain $\sum_{i=1}^{k+1}{2i-1}=\sum_{i=1}^k{2i-1}+2(k+1)-1$
        \begin{justification}[method=arith:split-sum]
          by splitting the sum.
        \end{justification}
      \end{spfstep}
      \begin{spfstep}
        Thus we have $\sum_{i=1}^{k+1}{(2i-1)}=k^2+2k+1$
        \begin{justification}[method=fertilize] by inductive hypothesis.\end{justification}
      \end{spfstep}
      \begin{spfstep}[type=conclusion]
        We can \begin{justification}[method=simplify]simplify\end{justification}
        the right-hand side to ${k+1}^2$, which proves the assertion.
      \end{spfstep}
   \end{spfcase}
    \begin{spfstep}[type=conclusion]
      We have considered all the cases, so we have proven the assertion.
    \end{spfstep}
  \end{spfcases}
\end{sproof}
```

Example 10: A very explicit proof, marked up semantically

Proof: We prove that $\sum_{i=1}^{n} 2i - 1 = n^2$ by induction over n

P.1 For the induction we have to consider the following cases:

P.1.1 $n = 1$: then we compute $1 = 1^2$ $\square$

P.1.2 $n = 2$: This case is not really necessary, but we do it for the fun of it (and to get more intuition). We compute $1 + 3 = 2^2 = 4$ $\square$

P.1.3 $n > 1$:

P.1.3.1 Now, we assume that the assertion is true for a certain $k \geq 1$, i.e. $\sum_{i=1}^{k} (2i - 1) = k^2$.

P.1.3.2 We have to show that we can derive the assertion for $n = k + 1$ from this assumption, i.e. $\sum_{i=1}^{k+1} (2i - 1) = (k + 1)^2$.

P.1.3.3 We obtain $\sum_{i=1}^{k+1} (2i - 1) = \sum_{i=1}^{k} (2i - 1) + 2(k + 1) - 1$ by splitting the sum

P.1.3.4 Thus we have $\sum_{i=1}^{k+1} (2i - 1) = k^2 + 2k + 1$ by inductive hypothesis.

P.1.3.5 We can simplify the right-hand side to $k + 1^2$, which proves the assertion. $\square$

P.1.4 We have considered all the cases, so we have proven the assertion. $\square$

Example 11: The formatted result of the proof in Figure 10 package. This environment totally invisible to the formatted result; it wraps the text in the proof step that corresponds to the evidence. The environment takes an optional `KeyVal` argument, which can have the `method` key, whose value is the name of a proof method (this will only need to mean something to the application that consumes the semantic annotations). Furthermore, the justification can contain "premises" (specifications to assertions that were used justify the step) and "arguments" (other information taken into account by the proof method).

`\premise` The `\premise` macro allows to mark up part of the text as reference to an assertion that is used in the argumentation. In the example in Figure 10 we have used the `\premise` macro to identify the inductive hypothesis.

`\justarg` The `\justarg` macro is very similar to `\premise` with the difference that it is used to mark up arguments to the proof method. Therefore the content of the first argument is interpreted as a mathematical object rather than as an identifier as in the case of `\premise`. In our example, we specified that the simplification should take place on the right hand side of the equation. Other examples include proof methods that instantiate. Here we would indicate the substituted object in a `\justarg` macro.

13.3 Proof Structure

`spfcases` The `pfcases` environment is used to mark up a proof by cases. This environment takes an optional `KeyVal` argument for semantic annotations and a second argument that allows to specify an introductory comment (just like in the `proof` environment).

`spfcase` The content of a `pfcases` environment are a sequence of case proofs marked up in the `pfcase` environment, which takes an optional `KeyVal` argument for semantic annotations. The second argument is used to specify the the description of the case under considertation. The content of a `pfcase` environment is the same as that of a `proof`, i.e. `steps`, `proofcomments`, and `pfcases` environments.

`sproofcomment` The `proofcomment` environment is much like a `step`, only that it does not have an obejct-level assertion of its own. Rather than asserting some fact that is relevant for the proof, it is used to explain where the proof is going, what we are attempting to to, or what we have achieved so far. As such, it cannot be the target of a `\premise`.

13.4 Proof End Markers

Traditionally, the end of a mathematical proof is marked with a little box at the end of the last line of the proof (if there is space and on the end of the next line if there isn't), like so: □

`\sproofend` The `sproof` package provides the `\sproofend` macro for this. If a different symbol for the proof end is to be used (e.g. *q.e.d*), then this can be obtained by
`\sProofEndSymbol` specifying it using the `\sProofEndSymbol` configuration macro (e.g. by specifying `\sProofEndSymbol{q.e.d}`).

Some of the proof structuring macros above will insert proof end symbols for sub-proofs, in most cases, this is desirable to make the proof structure explicit, but sometimes this wastes space (especially, if a proof ends in a case analysis which will supply its own proof end marker). Therefore, all proof environments have the `noproofend` `noproofend` keyword that suppresses the proof end markers for this element. It can be specified on its own, and does not need a value (if one is specified, that is completely ignored).

omdoc.dtx

14 Introduction

The `omdoc` package supplies macros and environment that allow to label document fragements and to reference them later in the same document or in other documents. In essence, this enhances the docuent-as-trees model to documents-as-directed-acyclic-graphs (DAG) model. This structure can be used by MKM systems for added-value services, either directly from the sTeX sources, or after translation. Currently, trans-document referencing provided by this package can conly be used in the sTeX collection.

sTeX is a version of TeX/LaTeX that allows to markup TeX/LaTeX documents semantically without leaving the document format, essentially turning TeX/LaTeX into a document format for mathematical knowledge management (MKM).

DAG models of documents allow to replace the "Copy and Paste" in the source document with a label-and-reference model where document are shared in the document source and the formatter does the copying during document formatting/presentation.[26][27][28]

EdNote(26)
EdNote(27)
EdNote(28)

15 The User Interface

15.1 Document Structure

`omgroup` The structure of the document is given by the `omgroup` environment just like in OMDoc.

15.2 Providing IDs for OMDoc Elements

Some of the OMDoc elements need IDs to function corrrectly. The general strategy here is to equip the sTeX macros with keys, so that the author can specify meaningful ones, but to let the transformation give default ones if the author did not.

15.3 Mathematical Text

`omtext` The `omtext` environment is used for any text fragment that has a contribution to a text that needs to be marked up. It can have a title, which can be specified via the `title` key. Often it is also helpful to annotate the `type` key. The standard relations from rhethorical structure theory `abstract`, `introduction`, `conclusion`, `thesis`, `comment`, `antithesis`, `elaboration`, `motivation`, `evidence`, `transition`, `note`, `annote` are recommended. Note that some of them are unary relations like `introduction`, which calls for a target. In this case, a target using the `for` key should be specified. The `transition` relation is special in that it is binary (a

[26]EdNote: talk about the advantages and give an example.
[27]EdNote: is there a way to load documents at URIs in LaTeX?
[28]EdNote: integrate with latexml's XMRef in the Math mode.

"transition between two statements"), so additionally, a source should be specified using the `from` key.[29] EdNote(29)

15.4 Structure Sharing

`\STRlabel` `\STRcopy` The `\STRlabel` macro takes two arguments: a label and the content and stores the the content for later use by `\STRcopy{label}`, which expands to the previously stored content.

`\STRsemantics` The `\STRlabel` macro has a variant `\STRsemantics`, where the label argument is optional, and which takes a third argument, which is ignored in LaTeX. This allows to specify the meaning of the content (whatever that may mean) in cases, where the source document is not formatted for presentation, but is transformed into some content markup format. [30] EdNote(30)

15.5 Phrase-Level Markup

`phrase` The `phrase` enviornment allows to mark up phrases with semantic information. It takes an optional `KeyVal` argument with the keys

[29] EdNote: describe the keys more fully
[30] EdNote: make an example

presentation.dtx

Contents

16 Introduction

The `presentation` package supplies an infrastructure that allows to specify the presentation of semantic macros, including preference-based bracket elision. This allows to markup the functional structure of mathematical formulae without having to lose high-quality human-oriented presentation in LaTeX. Moreover, the notation definitions can be used by MKM systems for added-value services, either directly from the sTeX sources, or after translation.

sTeX is a version of TeX/LaTeX that allows to markup TeX/LaTeX documents semantically without leaving the document format, essentially turning TeX/LaTeX into a document format for mathematical knowledge management (MKM).

The setup for semantic macros described in the sTeX `modules` package works well for simple mathematical functions: we make use of the macro application syntax in TeX to express function application. For a simple function called "foo", we would just declare `\symdef{foo}[1]{foo(#1)}` and have the concise and intuitive syntax `\foo{x}` for $foo(x)$. But mathematical notation is much more varied and interesting than just this.

17 The User Interface

In this package we will follow the sTeX approach and assume that there are four basic types of mathematical expressions: symbols, variables, applications and binders. Presentation of the variables is relatively straightforward, so we will not concern ourselves with that. The application of functions in mathematics is mostly presented in the form $f(a_1, \ldots, a_n)$, where f is the function and the a_i are the arguments. However, many commonly-used functions from this presentational scheme: for instance binomial coefficients: $\binom{n}{k}$, pairs: $\langle a, b\rangle$, sets: $\{x \in S \,|\, x^2 \neq 0\}$, or even simple addition: $3 + 5 + 7$. Note that in all these cases, the presentation is determined by the (functional) head of the expression, so we will bind the presentational infrastructure to the operator.

17.1 Mixfix Notations

For the presentation of ordinary operators, we will follow the approach used by the Isabelle theorem prover. There, the presentation of an n-ary function (i.e. one that takes n arguments) is specified as $\langle pre\rangle\langle arg_0\rangle\langle mid_1\rangle\cdots\langle mid_n\rangle\langle arg_n\rangle\langle post\rangle$, where the $\langle arg_i\rangle$ are the arguments and $\langle pre\rangle$, $\langle post\rangle$, and the $\langle mid_i\rangle$ are presentational material. For instance, in infix operators like the binary subset operator, $\langle pre\rangle$ and $\langle post\rangle$ are empty, and $\langle mid_1\rangle$ is $\subseteq$. For the ternary conditional operator in a programming language, we might have the presentation pattern `if`$\langle arg_1\rangle$`then`$\langle arg_2\rangle$`else`$\langle arg_3\rangle$`fi` that utilizes all presentation positions.

`\mixfix*` The `presentation` package provides mixfix declaration macros `\mixfixi`, `\mixfixii`, and `\mixfixiii` for unary, binary, and ternary functions. This covers most of the cases, larger arities would need a different argument pattern.[10] The

[10] If you really need larger arities, contact the author!

call pattern of these macros is just the presentation pattern above. In general, the mixfix declaration of arity i has $2n + 1$ arguments, where the even-numbered ones are for the arguments of the functions and the odd-numbered ones are for presentation material. For instance, to define a semantic macro for the subset relation and the conditional, we would use the markup in Figure 12.

```
\symdef{sseteq}[2]{\mixfixii{}{#1}{\subseteq}{#2}{}}
\symdef{sseteq}[2]{\infix\subseteq{#1}{#2}}
\symdef{ite}[2]{\mixfixiii{{\tt{if}}\;}{#1}
                          {\;{\tt{then}}\;}{#2}
                          {\;{\tt{else}}\;}{#3}{\;{\tt{fi}}}}
```

source	presentation
`\sseteq{S}T`	$(S \subseteq T)$
`\ite{x<0}{-x}x`	$\texttt{if}\,x < 0\,\texttt{then}\, -x\,\texttt{else}\,x\,\texttt{fi}$

Example 12: Declaration of mixfix operators

`\prefix` For certain common cases, the **`presentation`** package provides shortcuts for the mixfix declarations. The `\prefix` macro allows to specify a prefix presentation for a function (the usual presentation in mathematics). Note that it is better to specify `\symdef{uminus}[1]{\prefix{-}{#1}}` than just `\symdef{uminus}[1]{-#1}`, since we can specify the bracketing behavior in the former (see Section 17.3).

`\postfix` The `\postfix` macro is similar, only that the function is presented after the argument as for e.g. the factorial function: 5! stands for the result of applying the factorial function to the number 5. Note that the function is still the first argument to the `\postfix` macro: we would specify the presentation for the factorial function with `\symdef{factorial}[1]{\postfix{!}{#1}}`.

`\infix` Finally, we provide the `\infix` macro for binary operators that are written between their arguments (see Figure 12).

17.2 n-ary Associative Operators

Take for instance the operator for set union: formally, it is a binary function on sets that is associative (i.e. $(S_1 \cup S_2) \cup S_3 = S_1 \cup (S_2 \cup S_3)$), therefore the brackets are often elided, and we write $S_1 \cup S_2 \cup S_3$ instead (once we have proven associativity). Some authors even go so far to introduce set union as a n-ary operator, i.e. a function that takes an arbitrary (positive) number of arguments. We will call such operators **n-ary associative**.

EdNote(31) Specifying the presentation[31] of n-ary associative operators in `\symdef` forms is not straightforward, so we provide some infrastructure for that. As we cannot predict the number of arguments for n-ary operators, we have to give them all at once, if we want to maintain our use of TEX macro application to specify

[31] EdNote: introduce the notion of presentation above

function application. So a semantic macro for an n-ary operator will be applied as `\nunion{`$\langle a_1\rangle$`,...,`$\langle a_n\rangle$`}`, where the sequence of n logical arguments $\langle a_i\rangle$ are supplied as one TEX argument which contains a comma-separated list. We provide variants of the mixfix declarations presented in section 17.1 which deal with associative arguments. For instance, the variant `\mixfixa` allows to specify n-ary associative operators. `\mixfixa{`$\langle pre\rangle$`}{`$\langle arg\rangle$`}{`$\langle post\rangle$`}{`$\langle op\rangle$`}` specifies a presentation, where $\langle arg\rangle$ is the associative argument and $\langle op\rangle$ is the corresponding operator that is mapped over the argument list; as above, $\langle pre\rangle$, $\langle post\rangle$, are prefix and postfix presentational material. For instance, the finite set constructor could be constructed as

`\mixfixa`

```
\newcommand{\fset}[1]{\mixfixa[p=0]{\{}{#1}{\}}{,}}
```

`\assoc`

The `\assoc` macro is a convenient abbreviation of a `\mixfixa` that can be used in cases, where $\langle pre\rangle$ and $\langle post\rangle$ are empty (i.e. in the majority of cases). It takes two arguments: the presentation of a binary operator, and a comma-separated list of arguments, it replaces the commas in the second argument with the operator in the first one. For instance `\assoc\cup{S_1,S_2,S_3}` will be formatted to $S_1 \cup S_2 \cup S_3$. Thus we can use `\def\nunion#1{\assoc\cup{#1}}` or even `\def\nunion{\assoc\cup}`, to define the n-ary operator for set union in TEX. For the definition of a semantic macro in sTEX, we use the second form, since we are more conscious of the right number of arguments and would declare `\symdef{nunion}[1]{\assoc\cup{#1}}`.[32]

EdNote(32)

`\prefixa` `\postfixa`

These macros `\prefix` and `\postfix` have n-ary variants `\prefixa` and `\postfixa` that take an arbitrary number of arguments (mathematically; syntactically grouped into one TEX argument). These take an extra separator argument.[33]

EdNote(33)

`\mixfixia` `\mixfixai` `\mixfixaa`

The `\mixfixii` macro has variants `\mixfixia`, `\mixfixai`, and `\mixfixaa`, which allow to make one or two arguments in a binary function associative[11]. A use case for the second macro is an nary function type operator `\fntype`, which can be defined via

```
\def\fntype#1#2{\mixfixai{}{#1}\rightarrow{#2}{}\times}
```

and which will format `\fntype{\alpha,\beta,\gamma}\delta` as $\alpha\times\beta\times\gamma\to\delta$.

17.3 Precedence-Based Bracket Elision

With the infrastructure supplied by the `\assoc` macro we could now try to combine set union and set intersection in one formula. Then, writing

```
\nunion{\ninters{a,b},\ninters{c,d}}        (2)
```

would yield $((a\cap b)\cup(c\cap d))$, and not $a\cap b\cup c\cap d$ as we would like, since $\cap$ binds stronger than $\cup$. Dropping outer brackets in the presentations of the presentation

[32]EdNote: think about big operators for ACI functions

[33]EdNote: think of a good example!

[11]If you really need larger arities with associative arguments, contact the package author!

of the operators will not help in general: it would give the desired form for (2) but $a \cap b \cup c \cap d$ for (3), where we would have liked $(a \cup b) \cap (c \cup d)$

```
\ninters{\nunion{a,b},\nunion{c,d}}                    (3)
```

In mathematics, brackets are elided, whenever the author anticipates that the reader can understand the formula without them, and would be overwhelmed with them. To achieve this, there are set of common conventions that govern bracket elision. The most common is to assign precedences to all operators, and elide brackets, if the precedence of the operator is lower than that of the context it is presented in. In our example above, we would assign $\cap$ a lower precedence than $\cup$ (and both a lower precedence than the initial precedence). To compute the presentation of (3) we start out with the `\ninters`, elide its brackets (since the precedence n of $\cup$ is lower than the initial precedence i), and set the context precedence for the arguments to n. When we present the arguments, we present the brackets, since the precedence of `nunion` is lower than the context precedence n.

This algorithm, which we call **precedence-based bracket elision** goes a long way towards approximating mathematical practice. Note that full bracket elision in mathematical practice is a reader-oriented process, it cannot be fully mechanical, e.g. in $(a \cap b \cap c \cap d \cap e \cap f \cap g) \cup h$ we better put the brackets around the septary intersection to help the reader even thoug they could have been elided by our algorithm. Therefore, the author has to retain full control over bracketing in a bracket elision architecture (otherwise it would become impossible to explain the concept of associativity).[34]. EdNote(34)

Precedence	Operators	Comment
200	+,-	unary
200	^	exponentiation
400	$*, \wedge, \cap$	multiplicative
500	$+, -, \vee, \cup$	additive
600	$/$	fraction
700	$=, \neq, \leq, <, >, \geq$	relation

Figure 1: Common Operator Precedences

In sTeX we supply an optional keyval arguments to the mixfix declarations and
`p` their abbreviations that allow to specify precedences: The key `p` key is used to
`pi` specify the **operator precedence**, and the keys `p`$\langle i \rangle$ can be used to specify the
`pii` **argument precedences**. The latter will set the precedence level while process-
`piii` ing the arguments, while the operator precedence invokes brackets, if it is larger than the current precedence level — which is set by the appropriate argument precedence by the dominating operators or the outer precedence.

[34]EdNote: think about how to implement that

If none of the precedences is specified, then the defaults are assumed. The operator precedence is set to the default operator precedence, which defaults to 1000 and can be set by `\setDefaultPrecedence{`⟨*prec*⟩`}` where ⟨*prec*⟩ is an integer. The argument precedences default to the operator precedence.

`\setDefaultPrecedence`

Figure 1 gives an overview over commonly used precedences. Note that most operators have precedences lower than the default precedence of 1000, otherwise the brackets would not be elided. For our examples above, we would define

```
\newcommand{\nunion}[1]{\assoc[p=500]{\cup}{#1}}
\newcommand{\ninters}[1]{\assoc[p=400]{\cap}{#1}}
```

to get the desired behavior.

Note that the presentation macros uses round brackets for grouping by default. We can specify other brackets via two more keywords: `lbrack` and `rbrack`. Just as above, we can also reset the default brackets with `\setDefaultLeftBracket{`⟨*lb*⟩`}`and `\setDefaultRightBracket{`⟨*rb*⟩`}` where ⟨*lb*⟩ and ⟨*rb*⟩ expand to the desired brackets. Note that formula parts that look like brackets usually are not. For instance, we should not define the finite set constructor via

`lbrack`
`rbrack`
`\setDefaultLeftBracket`
`\setDefaultRightBracket`

```
\newcommand{\fset}[1]{\assoc[lbrack=\{,rbrack=\}]{,}{#1}}
```

where the curly braces are used as brackets, but as presented in section 17.2 even though both would format `\fset{a,b,c}` as $\{a,b,c\}$. In the encoding here, an operator with suitably high operator precedence would be able to make the brackets disappear.

17.4 Flexible Elision

There are several situations in which it is desirable to display only some parts of the presentation:

- We have alreday seen the case of redundant brackets above
- Arguments that are strictly necessary are omitted to simplify the notation, and the reader is trusted to fill them in from the context.
- Arguments are omitted because they have default values. For example $\log_{10} x$ is often written as $\log x$.
- Arguments whose values can be inferred from the other arguments are usually omitted. For example, matrix multiplication formally takes five arguments, namely the dimensions of the multiplied matrices and the matrices themselves, but only the latter two are displayed.

Typically, these elisions are confusing for readers who are getting acquainted with a topic, but become more and more helpful as the reader advances. For experienced readers more is elided to focus on relevant material, for beginners representations are more explicit. In the process of writing a mathematical document for traditional (print) media, an author has to decide on the intended audience and design the level of elision (which need not be constant over the document though). With electronic media we have new possibilities: we can make elisions flexible. The author still chooses the elision level for the initial presentation, but the reader can adapt it to her level of competence and comfort, making details more or less explicit.

\elide To provide this functionality, the `presentation` package provides the `\elide` macro allows to asociate a text with an integer **visibility level** and group them into **elision groups**. High levels mean high elidability.

Elision can take various forms in print and digital media. In static media like traditional print on paper or the PostScript format, we have to fix the elision level, and can decide at presentation time which elidable tokens will be printed and which will not. In this case, the presentation algorithm will take visibility thresholds T_g for every elidability group g as a user parameter and then elide (i.e. not print) all tokens in visibility group g with level $l > T_g$. We specify this threshold for via the `\setelevel` macro. For instance in the example below, we have a two type annotations `par` for type parameters and `typ` for type annotations themselves.

\setelevel

```
$\mathbf{I}\elide{par}{500}{^\alpha}\elide{typ}{100}{_{\alpha\to\alpha}}
    :=\lambda{X\elide{ty}{500}{_\alpha}}.X$
```

The visibility levels in the example encode how redundant the author thinks the elided parts of the formula are: low values show high redundancy. In our example the intuition is that the type paraemter on the **I** cominator and the type annotation on the bound variable X in the λ expression are of the same obviousness to the reader. So in a document that contains `\setegroup{typ}{1000}` and `\setegroup{an}{1000}` will show $\mathbf{I} := \lambda X.X$ eliding all redundant information. If we have both values at 400, then we will see $\mathbf{I}^\alpha := \lambda X_\alpha.X$ and only if the threshold for `typ` dips below 100, then we see the full information: $\mathbf{I}^\alpha_{\alpha\to\alpha} := \lambda X_\alpha.X$.

In an output format that is capable of interactively changing its appearance, e.g. dynamic XHTML+MathML (i.e. XHTML with embedded Presentation MathML formulas, which can be manipulated via JavaScript in browsers), an application can export the information about elision groups and levels to the target format, and can then dynamically change the visibility thresholds by user interaction. Here the visibility threshold would also be used, but here it only determines the default rendering; a user can then fine-tune the document dynamically to reveal elided material to support understanding or to elide more to increase conciseness.

The price the author has to pay for this enhanced user experience is that she has to specify elided parts of a formula that would have been left out in conventional LaTeX. Some of this can be alleviated by good coding practices. Let us consider the log base case. This is elided in mathematics, since the reader is expected to pick it up from context. Using semantic macros, we can mimic this behavior: defining two semantic macros: `\logC` which picks up the log base from the context via the `\logbase` macro and `\logB` which takes it as a (first) argument.

```
\provideEdefault{logbase}{10}
\symdef{logB}[2]{\prefix{\mathrm{log}\elide{base}{100}{_{#1}}}{#2}}
\abbrdef{logC}[1]{\logB{\fromEcontext{logbase}}{#1}}
```

`\provideEdefault` `\fromEcontext` `setEdefault` Here we use the `\provideEdefault` macor to initialize a LaTeX token register for the `logbase` default, which we can pick up from the elision context using `\fromEcontext` in the definition of `\logC`. Thus `\logC{x}` would render as $\log_{10}(x)$ with a threshold of 50 for `base` and as $\log_2$, if the local TeX group e.g. given by the `assertion` environment contains a `\setEdefault{logbase}{2}`.

17.5 Hyperlinking

EdNote(35) [35]

17.6 Variable Names

EdNote(36) [36]

`\vname` `\vname` identifies a token sequence as a name, and provides an ASCII (XML-compatible) identifier for it. The optional argument is the identifier, and the second one the LaTeX representation. The identifier can also be used with `\vnameref` for copy and paste.[37] EdNote(37)

[35]EdNote: describe what we want to do here
[36]EdNote: what is the problem?
[37]EdNote: does this really work

References

[ABC+03] Ron Ausbrooks, Stephen Buswell, David Carlisle, Stéphane Dalmas, Stan Devitt, Angel Diaz, Max Froumentin, Roger Hunter, Patrick Ion, Michael Kohlhase, Robert Miner, Nico Poppelier, Bruce Smith, Neil Soiffer, Robert Sutor, and Stephen Watt. Mathematical Markup Language (MathML) version 2.0 (second edition). W3C recommendation, World Wide Web Consortium, 2003. Available at http://www.w3.org/TR/MathML2.

[BCC+04] Stephen Buswell, Olga Caprotti, David P. Carlisle, Michael C. Dewar, Marc Gaetano, and Michael Kohlhase. The Open Math standard, version 2.0. Technical report, The Open Math Society, 2004. http://www.openmath.org/standard/om20.

[Ber89] J. A. Bergstra. *Algebraic specification.* ACM Press, 1989.

[Dea99] Stephen Deach. Extensible stylesheet language (xsl) specification. W3c working draft, W3C, 1999. Available at http://www.w3.org/TR/WD-xsl.

[Kay00] Michael Kay. *XSLT Programmers Reference.* Wrox, 2000.

[Knu84] Donald E. Knuth. *The TEXbook.* Addison Wesley, 1984.

[Koh05] Michael Kohlhase. Semantic markup for TEX/LATEX. Manuscript, available at http://kwarc.info/software/stex, 2005.

[Koh06] Michael Kohlhase. OMDoc – an open markup format for mathematical documents [version 1.2], 2006.

[Lam94] Leslie Lamport. *LaTeX: A Document Preparation System, 2/e.* Addison Wesley, 1994.

[MBA+01] E. Melis, J. Buedenbender, E. Andres, Adrian Frischauf, G. Goguadze, P. Libbrecht, M. Pollet, and C. Ullrich. The activemath learning environment. *Artificial Intelligence and Education*, 12(4), 2001.

[Mil07] Bruce Miller. `LaTeXML`: A LATEX to xml converter. Web Manual at http://dlmf.nist.gov/LaTeXML/, seen September2007.

[MKM07] Meetings and Conferences on Mathematical Knowledge Management. Project homepage at http://www.mkm-ig.org/meetings/, seen March 2007.

[RV01] Alan Robinson and Andrei Voronkov, editors. *Handbook of Automated Reasoning*, volume I and II. Elsevier Science and MIT Press, 2001.