— SMGloM Blue Note* —
SMGloM Primer

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

December 6, 2015

Abstract

The SMGloM is a semantic, multilingual terminology base for mathematics. It provides a lexical resource for document processing and understanding. This note is a primer – i.e. an introductory hands-on manual – for SMGloM authors. It discusses the anatomy and structure of SMGloM modules, how to edit and manage them in the local and web workflows, and how to debug the content with the various tools supporting SMGloM development.

*Inspired by the "blue book" in Alan Bundy's group at the University of Edinburgh, SMGloM blue notes, are documents used for fixing and discussing ϵ-baked ideas in projects by the SMGloM group (see http://mathhub.info/help/SMGloM). Unless specified otherwise, they are for project-internal discussions only. Please only distribute outside the SMGloM group after consultation with the author.

Contents

1 Introduction

The SMGloM is a semantic, multilingual terminology for mathematics. It provides a lexical resource for document processing and understanding. System, Management, and organizational aspects of the SMGloM have been described from a systems or data model perspective elsewhere [Koh13; Koh14b; KJ14; Koh14a] (consult them for details and further discussion). In this primer, we want to give a hands-on introduction for the SMGloM author and curator.

2 SMGloM Concepts

Before we can understand the workflow details and how-tos, we first need to get an overview over the system and the terminology used in SMGloM

Terms are words and compound words that in specific contexts are given specific meanings. In SMGloM we fix terms as **lexemes** which summarize the various inflectional variants of a word or compound word into a single unit of lexical meaning[1]. Lexemes are usually referenced by their **lemma** (or **citation form**) – a particular form of a lexeme that is chosen by convention to represent a canonical form of a lexeme.

A **terminology** or **termbase** is a special ontology that organizes terms and their definitions by terminological relations and/or the inherent structure of the underlying domain. Despite its name SMGloM is a terminology for mathematics, which is distinct from a **glossary**, – i.e. a list of technical/non-standard terms with short definitions ordered alphabetically or in the chronology of the document it illustrates. A glossary can be generated from a terminology automatically. But so can other lexical resources, e.g. a **translation dictionary** – i.e. a specialized dictionary used to translate words or phrases from one language to another: a translation dictionary usually consists of tuples of words from multiple languages that are translations of each other – information present in a multilingual terminology.

Terminological relations are semantic relations between terms. SMGloM uses the following ones: *i*) Two terms are **synonymous**, if they have the same meaning, i.e. they are interchangeable in a context without changing the truth value of the proposition in which they are embedded. *ii*) Term Y is a **hypernym** of term X if every X is a (kind of) Y; the converse relation of hypernymy is called **hyponymy**. *iii*) Term Y is a **meronym** of term X if Y is a part of X; the converse relation of meronomy is called **holonymy**, *iv*) Two terms are **homonyms** if they have the same pronunciation and spelling (but different meanings). *v*) Two terms are **antonyms**, if they have opposite meanings: one is the antithesis of the other.

The SMGloM terminology is organized as a set of **terminology modules**, which contain a set of **symbols** – i.e. language-independent identifiers for the mathematical concepts the terms denote, their **verbalizations** (language-dependent common names) and the corresponding definitions and **notations** (symbolic representations for formulae), and the **terminological relations** to other glossary entries. Note that symbols, notations, and terminological relations are language-independent, while definitions and verbalizations are language dependent, so we structure a terminology module into a **module signature** and a set of **language bindings**. All of these are written in STEX [sTeX; Koh08], a semantic variant of

[1] In SMGloM we do not cover grammatical information about the lexemes yet. That is usually represented in a **lexicon**, a listing of the lexemes of a given language together with a grammar a system of rules which allow for the combination of those words into meaningful sentences.

LaTeX that is optimized towards content markup.

3 The Anatomy of an SMGloM Module

We will start with language bindings and work our way towards the module signature using a terminology module that defines the concept of divisibility as an example.

3.1 Language Bindings

In Listing 1 we present the English language binding for divisibility; which we assume to be in a file divisor.en.tex. We generally use the convention for the file name ⟪mod⟫.⟪lang⟫.tex, where ⟪mod⟫ is the module name and ⟪lang⟫ is the ISO 639 language specifier – e.g. en for English, de for German, fr for French, ro for Romanian, xh for Chinese.

Listing 1: The English Language Binding for the Module on "divisibility"

```
1  \begin{modnl}[creators=miko]{divisor}{en}
2    \begin{definition}[id=divisor.en.def]
3      Let $n$ and $m$ be \trefii[nats]{natural}{number}s, then we say that $n$ is a
4      \defi{divisor} of $m$ (or that $n$ \defi[divisor]{divides} $m$) and write
5      $\divides{n}m$, iff there is a \trefii[nats]{natural}{number} $k$, such that $m=nk$.
6    \end{definition}
7
8    \begin{definition}
9      If $n$ is a \trefi{divisor} of $m$, then we call $m$ a \defi{multiple} of $n$.
10   \end{definition}
11 \end{modnl}
```

Let us look at the markup one-by-one: The modnl environment marks up STeX language bindings. SMGloM language binding files should only contain single modnl environments. The environment takes two mandatory arguments and an optional one. The optional argument – in [] – allows key/value pairs for metadata – here the creators key was used to specify the (nickname of the) author[2] . The first mandatory argument – in {} – is the **glossary module name** – it should coincide with the base of the file name – and the second argument the ISO 639 language specifier.

The content of the modnl is the content of the SMGloM module encoded in STeX, usually a single **definition** environment that gives the definition of the concepts of the module. Multiple definitions as in our example are allowed if the module specifies closely related concepts where a separation into multiple modules would be inconvenient.

The optional argument of the **definition** environment has an id key whose value is an identifier. The content of the **definition** is natural mathematical language with special infrastructure for defining and referencing mathematical terms.

3.2 Defining Technical Terms

The definiendum – i.e. the term defined in the definition – is marked up with a **defi** macro. In Listing 1 (line 4) we see **defi**{divisor}, which does two things: it *i*) marks up the phrase "divisor" in the text as the definiendum, and *ii*) associates it with a symbol[1] divisor for EdN:1

[2] If the key value is a list, then it must be encapsulated in curly braces, e.g. as in Listing 3.

[1] EdNote: MK: remember to state restrictions on symbol names. (no Math).

referencing. Intuitively, the symbol is the language-independent (platonic) name of the mathematical concept, where as the phrase singled out in the definiendum its language-specific realization in natural language (we call it a **verbalization** of the respective symbol). If we want to associate a definiendum with a different symbol, we can give that in the optional argument of defi. We have made use of this in Listing 1 (line 4) to associate an alternative verbalization "divides" with the symbol divisor[2]. In this sense, the symbol divisor coordinates the phrases "divisor" and "divides" as synonymous technical terms. EdN:2

The same mechanism is used to coordinate verbalizations in other languages; see for instance Listing 2, where we use **defi**[divisor]{Teiler} to specify that the German term "Teiler" is a verbalization of the symbol divisor and therefore a translation of the English term "divisor".

The **defi** macro has variants **defii** and **defiii** for two- and three-word compounds. For instance, a "special linear group" would be marked up as **defiii**{special}{linear}{group}; this corresponds to the symbol name special−linear−group.

In all the cases above, the definiendum phrase was already the citation form[3] If the text of the definition forces us to use a term in non-citation form, then we can do this via the variant **adefi**, where **adefi**[⟪sym⟫]{⟪phrase⟫}{⟪lemma⟫} will associate the term ⟪lemma⟫ in citation form with the symbol ⟪sym⟫ and display ⟪phrase⟫ for it in the glossary module. Note that **adefi** has variants **adefii** and **adefiii** for two/three-word lemmata.

3.3 Referencing Technical Terms

Terms defined other modules can be referenced by their module and symbol names. For instance, in line 3 of Listing 1 we can see **trefii**[nats]{natural}{number}, which references the symbol natural−number from the nats module and formats the phrase "natural number". Term references have the same systematicity for multi-word compounds and alternative phrases (in the macros **trefi**, **trefii**, **trefiii**, **atrefi**, **atrefii**, and **atrefiii**). If the symbol referenced is from the current module, the optional argument of the \∗tref∗ can be left out.

Listing 2: The German Language Binding for the Module on "divisibility"

```
1  \begin{modnl}[creator=ako]{divisor}{de}
2    \begin{definition}[id=divisor.en.def]
3      Sind $n$ und $m$ \mtrefii[nats?natural−number]{nat"urliche}{Zahl}en, so nennen wir $n$
4      einen \defi{divisor} oder \defi[divisor]{Teiler} von $m$ (und schreiben
5      $\divides{n}m$), falls es eine \mtrefii[nats?natural−number]{nat"urliche}{Zahl} $k$
6      gibt, so dass $m=nk$.
7    \end{definition}
8
9    \begin{definition}
10     Ist $n$ ein \mtrefi[?divisor]{Teiler} von $m$, so sagen wir auch dass $n$ $m$
11     \defi[divisor]{teilt} und nennen $m$ ein \defi[multiple]{Vielfaches} von $n$.
12   \end{definition}
13 \end{modnl}
```

In Listing 2 we have another referencing macro: **mtrefi**, which is designed for use in non-primary (i.e. usually non-English) language bindings. There we have the problem that

[2] EdNote: MK: it is not totally clear to me that we should have two verbalizations of the same symbol with completely different grammatical categories, here verb and noun phrase. If we decided to allow this this is a very good example and should be discussed in more detail here.

[3] The **citation form** or **lemma** of a technical term is that inflection you would expect to find cited in a dictionary.

the phrase to be formatted usually different from the symbol name, but maybe a multiword compound. **mtrefi** uses the mandatory arguments for the word components and allows to specify module and name of the referenced symbol in the optional first argument. The general form is **mtrefi**[⟪mod⟫?⟪sym⟫]{⟪phrase⟫}, where *i*) ⟪mod⟫ is the module name – this may be left empty for the "current module" (see Listing 2 line 10 for an example), *ii*) ⟪sym⟫ is the symbol name, and *iii*) ⟪phrase⟫ is the phrase to be formatted. Note that we have variants **mtrefii** and **mtrefiii** for two/tree-word compounds.

Sometimes, the \∗tref∗ mechanism is too heavyweight to be practical, especially when starting off with a new topic in the glossary. Then the terminology modules we need to reference do not exist yet. To get started, we can use the \term macro to get started. \term{⟪phrase⟫} marks up ⟪phrase⟫ as a term "to be referenced, when the target exists". It should be thought of and used as a Wiki-style dangling link that is re-visited to complete the markup. \term takes an optional module name: \term[mod]{⟪phrase⟫} is used in case the module ⟪mod⟫ already exists, but does not have a symbol for ⟪phrase⟫.

3.4 The Module Signature

The language bindings are accompanied by a module signature that imports the modules of terms referenced in the language bindings, declares all the symbols in the module, and adds notation definitions.

Listing 3: The Module Signature for "divisibility"

```
1 \begin{modsig}[creators={miko,ako}]{divisor}
2   \gimport{nats}
3   \symi{multiple}
4   \symdef[name=divisor]{dividesOp}{|}
5   \symdef[name=divisor]{divides}[2]{#1\dividesOp{#2}}
6   \symtest{divisor}{\divides{3}{6}}
7 \end{modsig}
8 \end{modnl}
```

Listing 3 shows the module signature for the "divisibility" module. The modsig environment takes an optional key/value argument for metadata, and the module name in the mandatory argument. In line 2 we see the \gimport macro that imports another module, here the nats module which supplies the natural−number symbol referenced in the language bindings. Note that all referenced symbols must be imported in the respective module signature, but also note that module import is recursive if imported modules have \gimports themselves.

In line 3 of Listing 3 we see a **symbol declaration** via the \symi macro: this declares the symbol with name multiple. Note that it is a glossary invariant that the names of all symbols defined in language bindings must be declared in the module signature, and symbols declared in the module signature must be defined in all language bindings. The order of symbol declarations is immaterial.

3.5 Semantic Macros and Notation Definitions

In lines 4 and 5 we see two instances of a the **symdef** macro that does triple duty:
symdef[⟪keys⟫]{⟪cs⟫}[⟪args⟫]{⟪notation⟫}

1. supplies a semantic macro \⟪cs⟫, where ⟪cs⟫ is a (new) control sequence – aka. "macro name". Applied to ⟪args⟫ arguments $a_1, \ldots a_n$, the semantic macro \⟪cs⟫ expands

to ⟪notation⟫. In our example, we have two semantic macros: \dividesOp without arguments, which expands to a vertical bar | and \divides with two arguments, such that \divides{a}b expands to $a|b$.

2. declares a symbol with name ⟪cs⟫ – unless ⟪keys⟫ has a key name=⟪name⟫, in which case it declares a symbol with name ⟪name⟫

3. associates with the declared symbol a notation definition as specified in ⟪notation⟫.

The new semantic macro \⟪cs⟫ can be used in all language bindings of the the current module signature and all modules that import it. In our example in Listing 3 we have used the name "divides" in the semantic macro \divides mainly for didactic reasons – macro name and symbol names can be different. Note that we already used the semantic macro \divides in the language bindings in Listings 1 and 2 above to specify the default notation for the divisibility relation. Note furthermore that we have defined a separate semantic macro \dividesOp, so that we can write down the divisibility relation |itself, rather than only the applied form. Both semantic macros correspond to the same symbol: divisor of course.

We can use the **symvariant** macro to specify alternative notations; see for instance the case for binomial coefficients in Listing 4, where we have a primary notation in the **symdef** in line 1 and a **symvariant** that adds an alternative notation in line 2.

Listing 4: Alternative Notation Definitions

```
1 \symdef[name=binomial-coefficient]{binom}[2]{\left(#1\atop #2\right)}
2 \symtest{binomcoeff}{\binomcoeff{n}k}
3 \symvariant{binom}[2]{c}{\mathcal{C}^{#1}_{#2}}
4 \symtest[variant=c]{binomcoeff}{\binomcoeff[c]{n}k}
```

The call pattern **symvariant**{⟪cs⟫}[⟪args⟫]{⟪var⟫}{⟪notation⟫} is similar to that of **symdef**: ⟪cs⟫, ⟪args⟫, and ⟪notation⟫ are identical, the key/value argument is missing as **symvariant** inherits all metadata – here the name – from the primary notation, whose control sequence is specified in ⟪cs⟫. The main difference is the variant specifier ⟪var⟫, which identifies the notation variant. The variant specifier can be used in an optional argument to the semantic macro: In the example from Listing 4 \binom{n}k expands to $\binom{n}{k}$ and \binom[c]{n}k results in $\mathcal{C}^n_k$.

Finally, we can use **symtest** to test the newly defined macros: **symtest**[⟪keys⟫]{⟪cs⟫}{⟪test⟫} will generate a phrase describing the test. For instance, the **symtest** macros in the listing above will generate the test output:

> Symbol binomcoeff with semantic macro \binomcoeff: used e.g. in $\binom{n}{k}$
>
> Symbol binomcoeff (variant c) with semantic macro \binomcoeff[c]: used e.g. in $\mathcal{C}^n_k$.

3.6 Specifying Presentation Declaratively

The notations in the semantics macros are somewhat unrealistic, since they use TEX primitives directly to express the layout of the basic mathematical operators. In SMGloM we usually use the more sophisticated facilities of the presentation package [**Kohlhase:ipsmsl:svn**] from STEX instead.[3]

EdN:3

[3] EdNote: MK: need to talk about the presentation package.

4 Authoring and Managing SMGloM content in MathHub

As we have seen in the last section SMGloM modules are sTEX files of a specific form. While they can in principle be written in an arbitrary text editor, LaTeX editing support helps considerably in practice. But the main support for SMGloM authoring and management comes with the MathHub system [MH; Ian+14] that hosts the SMGloM and provides Git repositories, a glossary view, a web editing workflow, and local working copy support via lmh. In the following, we will discuss the web workflow (ultimately for casual users) and the lmh-based workflows (for power-users).

The web workflows are provided online by the MathHub portal at `http://mathhub.info`, we will give the specific REST URLs for the workflows. The lmh tool is a python script, that supports multi-repository workflows and SMGloM-specific actions.

Currently not all tasks are supported in both forms, therefore mixed workflows may be necessary, therefore we will first cover obtaining a local working copy of the SMGloM content.

4.1 Obtaining a Local Working Copy of SMGloM

We assume a working installation of lmh (see [LMH] for instructions and workflow details) in $HOME$/localmh. As the MS Windows port is still immature, we recommend Linux or Mac OS X and restrict our exposition to those. We can install the SMGloM content with

```
cd $HOME/localmh/MathHub
lmh install smglom/*
lmh install ⟨account⟩/*
```

The registered SMGloM content is in `$HOME/localmh/MathHub/smglom` and the private repositories in `$HOME/localmh/MathHub/`⟨account⟩. To generate all unversioned files we should also issue

```
lmh gen --alltex --localpaths
```

Now, the local working copy with SMGloM and private content is set up and ready to go. Note that the lmh gen command should be re-issued in any subdirectory of the local working copy where changes have been made.

4.2 Managing SMGloM Repositories

For inclusion into SMGloM, and management support in MathHub, SMGloM modules must be committed to a GIT [GIT] repository on the GitLab [GL] repository of MathHub. The SMGloM content is distributed over multiple repositories in the SMGloM repository group (see `http://gl.mathhub.info/groups/smglom`). Each repository groups a set of modules manages access by restricting write access to a set of accounts. Repositories can but need not be thematically oriented; see [Koh14b, Section 2.2] for a description of the setup.

For the following we will assume that the user has a MathHub account with name ⟨account⟩, is signed on (so that the REST URIs are accessible), and has deployed an ssh key via `http://gl.mathhub.info/profile/keys`. Then we can create a private SMGloM repository ⟨repos⟩ by issuing

```
cd $HOME/localmh/MathHub/⟨account⟩
lmh init -t smglom ⟨repos⟩
```

This will generate a subdirectory ⟪repos⟫ in `$HOME/localmh/MathHub/`⟪account⟫ with the subdirectories META−INF for lmh and repository metadata, ∗.msl for MMT, lib for STEX metadata and files, and source for the SMGloM modules. For the following we assume that our working directory is

```
1 $HOME/localmh/MathHub/⟪account⟫/⟪repos⟫/source
```

Note that the usual GIT commands (git status, git pull, git commit, git push work as expected), and a versioned pull/commit/push workflow can and should be followed for glossary development.

Alternatively, an SMGloM module can be edited in the web workflow under `http://mathhub.info/`⟪account⟫`/`⟪repos⟫`/source/`⟪module⟫`/.omdoc` – just change to the "edit" tab, this gives a joint editing window for all components of the module.

4.3 Forking an SMGloM Repository

[4]

EdN:4

4.4 Pre-Translations

Once the first (we call it the **primary**) language binding for a terminology module has been written, it is much easier to create another. In fact lmh supports this via the lmh translate command: lmh translate ⟪source⟫ ⟪lang⟫ translates a language binding ⟪source⟫ into one of the language specified in the ISO 639 specifier ⟪lang⟫, taking into account all the lexical information in SMGloM.[5][6]

EdN:5
EdN:6

5 Fixing SMGloM Errors and Quality Control

The STEX encoding of SMGloM modules contains markup at multiple levels. This ranges form content markup for formulae provided by the semantic macros – where errors in the function/argument structure may occur – to symbol declarations, imports, and term references – where the semantic links may dangle or the import structure may become cyclic. The two editing workflows described above allow complementary quality control measures: On the local working copy, we can run pdflatex from the local LATEX installation for proofreading for mathematical correctness and verifying the function/argument structure of content formulae as well as the imports declarations. But pdflatex processing does not take semantic linking into account. That can be testing by generating OMDOC [Koh06] via LATEXML [LTX] and then loading it into the MMT API [Rab13; MMT].

5.1 The Local PDFLATEX Workflow

Glossary modules can be tested with lmh. lmh pdf ⟪comp⟫ will run the pdflatex program of your system's LATEX installation over the module component (the module signature or a language binding) using the SMGloM preamble file provided by your repository. Note that the file localpaths.tex needs to be up-to-date (it can be re-generated by lmh gen −−localpaths).

[4] EDNOTE: MK: does not work yet, needs to be implemented and described.
[5] EDNOTE: MK: Need to test this, to see whether *trefi* are translated correctly.
[6] EDNOTE: Make a simple example and discuss that.

Formatting a module component with lmh pdf is a good way to flush out sTeX bugs and get a PDF for proofreading. lmh pdf without specifying a glossary component file (recursively) re-formats all components in the current directory that have changed since the last transformation lmh pdf −f forces re-formatting on all files.

Alternatively, all glossary modules in the repository can be formatted into a single PDF file by running pdflatex over the file all.tex, which can be generated by lmh gen −−alltex. This provides a good visual overview over the modules in the current repository.

5.2 Local Translation LaTeXML

The SMGloM components can be transformed into OMDoc via LaTeXML; this reveals a slightly different set of errors than the pdflatex workflow; and the resulting OMDoc files are more machine-processable. Given a complete lmh installation a glossary component ⟪comp⟫ in the local working copy can be transformed from the command line by issuing lmh omdoc ⟪comp⟫. This generates the log file ⟪comp⟫.ltxlog and (if successful) the result file ⟪comp⟫.omdoc. lmh omdoc without specifying a glossary component file (recursively) re-translates all components in the current directory that have changed since the last transformation lmh omdoc −f forces re-translation on all files. grep Error ∗.ltxlog gives an overview over all the errors incurred in this process.

5.3 Local Mmt

SMGloM components in OMDoc can be processed by the Mmt API to semantically instrumented XHTML form that can be browsed a MathML-enabled browser. As this transformation exercises many of the semantic features, generating XHTML via lmh xhtml ⟪comp⟫[7] EdN:7
reveals many of the semantic errors in the sTeX source of the components.[8] EdN:8

5.4 Errors in the Web Workflow

The web interface of MathHub combines the LaTeXML transformation and Mmt presentation into one (automatic) process when a glossary component is committed or previewed. Correspondingly, the errors reported by the two processes involved are reported in a central location: `http://mathhub.info/mh/common-errors`. Particular errors are aggregated by error text (see Figure 1); the blue "Show All" gives access to all occurrences, which can be , and the red button allows users with sufficient rights to re-run all components that have this error – this usually needs an edit or update (see Section 5.5).[9] Warnings are treated analogously, EdN:9
but are colored in yellow.[10] EdN:10

5.5 Keeping Up-To-Date

Note that the presentation processes discussed here depend on many factors to be current: *i*) the SMGloM sources in the working copy *ii*) the sTeX and *iii*) LaTeXML distributions, and *iv*) the Mmt API. All of these have to be kept up to date to get the newest state of the

[7] EdNote: MK: this currently does not work for me; maybe soon.
[8] EdNote: MK: needs to be described more, when it works again.
[9] EdNote: MK@MH: does this also re-crawl the nodes? I think it should.
[10] EdNote: MK@MI: we should show the error annotations in a figure here and discuss it. Could you at least make a nice screenshot?

Figure 1: An Error reported by LaTeXML

glossary components and glossary. In the local working copy, they can be updated with a lmh update at `$HOME/localmh/MathHub`.

Note that the MathHub system keeps its own working copy internally,[11] so the same updating requirements apply to that as well. As users do not have access to the MathHub server, the system gives users with sufficient rights an web front end to lmh on the server at `http://mathhub.info/mh/administrate_mathhub`. Note that re-running the whole of SMGloM puts quite a lot of computational load on the system, so this should be used sparingly. Note that to update the SMGloM glossary (see 6.1), the generated nodes must be re-crawled which can be done (in batches of 30) via `http://mathhub.info/mh/crawl-nodes`. EdN:11

5.6 Communicating about Errors/Issues

When looking at the terminology modules[12] or the generated glossary (see Section 6.1), we often find issues that still need to be resolved. There are two ways to address these. If we have write access to the sTeX sources, we can insert an \ednote{⟪AI⟫: ⟪message⟫} in the source (see [Koh15] for details about the ed package). This formats as a special footnote into the PDF; the convention is to sign the message with the author's initials ⟪AI⟫. EdN:12

In the web interface, we cal also use the "localized commenting" feature of MathHub; a right-click will[13][14] EdN:13

General issues with the content of a repository can be raised via its associated issue tracker, which can be found at `http://gl.mathhub.info/`⟪group⟫`/`⟪repos⟫`/issues/`. This can also be used for project planning for this repository. The issue tracker is also a good place for content requests. EdN:14

6 SMGloM Applications

There are various ways the SMGloM modules can be used. For an overview of potential applications see [Koh13], we will go over the ones that are already realized in the system here, since they can also be used for quality control.

6.1 The Glossary Front-End

The glossary generated from the SMGloM modules is hosted on MathHub at `http://mathhub.info/mh/glossary`. It consists of alphabetically ordered glossary entries such as the one in Figure 2[15]. The blue label "de" gives access to the German language bindings EdN:15

[11] EdNote: MK: eventually, we need to update this discussion to MathHub Workers when they are deployed
[12] EdNote: MK@MK: use this word instead of glossary modules
[13] EdNote: MK: this does not work any more for me; describe further, when it does again. Make screenshots.
[14] EdNote: Do we have any notification feature here?
[15] EdNote: MK: update, when bsetst notation works

for this mathematical concept (other language bindings would generate other links in the glossary).

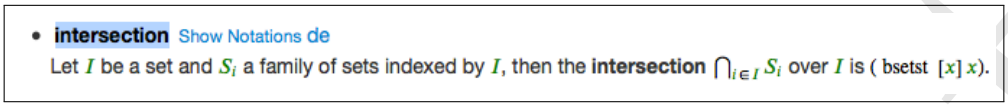

Figure 2: A Generated Glossary Entry

The "Show Notations" link reveals the notation definitions for the concept in the glossary entry in the STEX source form and the MathML presentation (see Figure 3) for a version of the glossary entry from Figure ?? with notations expanded.

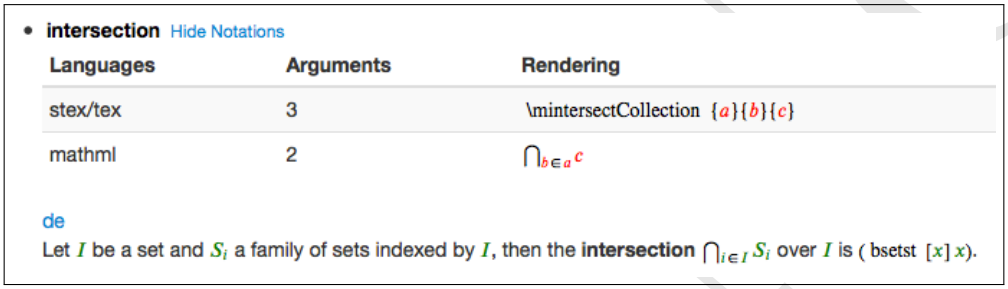

Figure 3: A Generated Glossary Entry With Notations

The generated glossary is a useful tool for getting an overview over the SMGloM contents. We can just use the in-page search feature of the browser to look for concept names (e.g. "Cauchy Sequence"). If that is found, we can look up the notations and (what is more important still) the semantic macros and their call patterns (see the STEX line in Figure 3).

7 Conclusion

We have presented an introductory hands-on manual for novice SMGloM authors. We have discussed the anatomy and structure of SMGloM modules, how to edit and manage them in the local and web workflows, and how to debug the content with the various tools supporting SMGloM development.

Note that the primer is relatively young and describes a moving target (the SMGloM system), so it may be a good idea to update it from [Koh14c] before re-reading it. Correspondingly, we are very interested in feedback, corrections, and documentation wishes. Please send these directly to the author.

References

[GIT] *Git – Fast Version Control System*. seen September 2007. URL: `http://git-scm.com/` (visited on 07/14/2014).

[GL] *GitLab*. URL: `http://gitlab.org` (visited on 02/24/2014).

[Ian+14] Mihnea Iancu et al. "System Description: MathHub.info". In: *Intelligent Computer Mathematics*. Conferences on Intelligent Computer Mathematics. (Coimbra, Portugal, July 7–11, 2014). Ed. by Stephan Watt et al. LNCS 8543. Springer, 2014, pp. 431–434. URL: http://kwarc.info/kohlhase/submit/cicm14-mathhub.pdf.

[KJ14] Michael Kohlhase and Constantin Jucovschi. "Editing Workflows in SMGloM". SMGloM Blue Note. 2014. URL: http://gl.kwarc.info/smglom/blue/raw/master/editing/note.pdf.

[Koh06] Michael Kohlhase. OMDoc – *An open markup format for mathematical documents [Version 1.2]*. LNAI 4180. Springer Verlag, Aug. 2006. URL: http://omdoc.org/pubs/omdoc1.2.pdf.

[Koh08] Michael Kohlhase. "Using LaTeX as a Semantic Markup Format". In: *Mathematics in Computer Science* 2.2 (2008), pp. 279–304. URL: https://svn.kwarc.info/repos/stex/doc/mcs08/stex.pdf.

[Koh13] Michael Kohlhase. "SMGloM: a Semantic Multilingual Glossary System for Mathematics". SMGloM Blue Note. 2013. URL: http://gl.kwarc.info/smglom/blue/raw/master/smglom/note.pdf.

[Koh14a] Michael Kohlhase. "A Data Model and Encoding for SMGloM". SMGloM Blue Note. 2014. URL: http://gl.kwarc.info/smglom/blue/raw/master/datamdl/note.pdf.

[Koh14b] Michael Kohlhase. "Content Management in SMGloM". SMGloM Blue Note. 2014. URL: http://gl.kwarc.info/smglom/blue/raw/master/contmgt/note.pdf.

[Koh14c] Michael Kohlhase. "SMGloM Primer". SMGloM Blue Note. 2014. URL: http://gl.kwarc.info/smglom/manuals/raw/master/primer/primer.pdf.

[Koh15] Michael Kohlhase. *Editorial Notes for LaTeX*. Tech. rep. Comprehensive TeX Archive Network (CTAN), 2015.

[LMH] *Local Editing Workflows*. URL: http://mathhub.info/help/lmh-workflows.html (visited on 04/14/2014).

[LTX] Bruce Miller. *LaTeXML: A LaTeX to XML Converter*. URL: http://dlmf.nist.gov/LaTeXML/ (visited on 03/12/2013).

[MH] *MathHub.info: Active Mathematics*. URL: http://mathhub.info (visited on 01/28/2014).

[MMT] Florian Rabe. *The MMT System*. URL: https://svn.kwarc.info/repos/MMT/doc/html/ (visited on 07/16/2014).

[Rab13] Florian Rabe. "The MMT API: A Generic MKM System". In: *Intelligent Computer Mathematics*. Conferences on Intelligent Computer Mathematics. (Bath, UK, July 8–12, 2013). Ed. by Jacques Carette et al. Lecture Notes in Computer Science 7961. Springer, 2013, pp. 339–343. DOI: 10.1007/978-3-642-39320-4.

[sTeX] *KWARC/sTeX*. URL: https://github.com/KWARC/sTeX (visited on 05/15/2015).