

smultiling.sty: Multilinguality Support for \LaTeX

Michael Kohlhase, Deyan Ginev
Jacobs University, Bremen
<http://kwarc.info/kohlhase>

May 14, 2014

Abstract

The `smultiling` package is part of the \LaTeX collection, a version of $\text{\TeX}/\text{\LaTeX}$ that allows to markup $\text{\TeX}/\text{\LaTeX}$ documents semantically without leaving the document format, essentially turning $\text{\TeX}/\text{\LaTeX}$ into a document format for mathematical knowledge management (MKM).

The `smultiling` package adds multilinguality support for \LaTeX , the idea is that multilingual modules in \LaTeX consist of a module signature together with multiple language bindings that inherit symbols from it, which also account for cross-language coordination.

Contents

1	Introduction	2
1.1	\LaTeX Module Signatures	2
2	The User Interface	2
2.1	Multilingual Modules	2
2.2	Multilingual Definitions and Crossreferencing Terms	3
2.3	Multilingual Views	4
3	Implementation	6
3.1	Class Options	6
3.2	Handling Languages	6
3.3	Signatures	8
3.4	Language Bindings	9
3.5	Multilingual Statements and Terms	12
3.6	Finale	12

1 Introduction

We have been using \TeX as the encoding for the Semantic Multilingual Glossary of Mathematics (SMGloM; see [Gin+14]). The SMGloM data model has been taxing the representational capabilities of \TeX with respect to multilingual support and verbalization definitions; see [Koh14], which we assume as background reading for this note.

1.1 \TeX Module Signatures

(monolingual) \TeX had the intuition that the symbol definitions (`\symdef` and `\symvariant`) are interspersed with the text and we generate \TeX module signatures (SMS `*.sms` files) from the \TeX files. The SMS duplicate “formal” information from the “narrative” \TeX files. In the SMGloM, we extend this idea by making the the SMS primary objects that contain the language-independent part of the formal structure conveyed by the \TeX documents and there may be multiple narrative “language bindings” that are translations of each other – and as we do not want to duplicate the formal parts, those are inherited from the SMS rather than written down in the language binding itself. So instead of the traditional monolingual markup in Figure 1, we we now advocate the divided style in Figure 2.

```
\begin{module}[id=foo]
\symdef{bar}{BAR}
\begin{definition}[for=bar]
  A \defiii{big}{array}{raster} ( $\bar{\phantom{x}}$ ) is a\ldots, it is much bigger
  than a \defiii[sar]{small}{array}{raster}.
\end{definition}
\end{module}
```

Example 1: A module with definition in monolingual \TeX

We retain the old `module` environment as an intermediate stage. It is still useful for monolingual texts. Note that for files with a module, we still have to extract `*.sms` files. It is not completely clear yet, how to adapt the workflows. We clearly need a `lmh` or editor command that transfers an old-style module into a new-style signature/binding combo to prepare it for multilingual treatment.

2 The User Interface

The `smultiling` package accepts all options of the `babel.sty` and just passes them on to it. The options specify which languages can be used in the \TeX language bindings.

2.1 Multilingual Modules

`modsig` There the `modsig` environment works exactly like the old `module` environment,

```

\usepackage[english,ngerman]{multiling}
\begin{modsig}{foo}
\symdef{bar}{BAR}
\symi{sar}
\end{modsig}

\begin{modnl}[creators=miko,primary]{foo}{en}
\begin{definition}
  A \defiii[bar]{big}{array}{raster} ( $\bar{}$ ) is a\ldots, it is much bigger
  than a \defiii[sar]{small}{array}{raster}.
\end{definition}
\end{modnl}

\begin{modnl}[creators=miko]{foo}{de}
\begin{definition}
  Ein \defiii[bar]{gro"ses}{Feld}{Raster} ( $\bar{}$ ) ist ein\ldots, es
  ist viel gr"o"ser als ein \defiii[sar]{kleines}{Feld}{Raster}.
\end{definition}
\end{modnl}

```

Example 2: Multilingual \TeX for Figure 1.

only that the `id` attribute has moved into the required argument – anonymous module signatures do not make sense.

`modnl` The `modnl` environment takes two arguments the first is the name of the module signature it provides language bindings for and the second the ISO 639 language specifier of the content language. We add the `primary` key `modnl`, which can specify the primary language binding (the one the others translate from; and which serves as the reference in case of translation conflicts).¹

`\symi` There is another difference in the multilingual encoding: All symbols are introduced in the module signature, either by a `\symdef` or the new `\symi` macro. `\symi{<name>}` takes a symbol name `<name>` as an argument and reserves that name. The variant `\symi*{<name>}` declares `<name>` to be a primary symbol; see [Koh14] for a discussion. \TeX provides variants `\symii` and `\symiii` – and their starred versions – for multi-part names.

2.2 Multilingual Definitions and Crossreferencing Terms

We do not need a new infrastructure for defining mathematical concepts, only the realization that symbols are language-independent. So we can use symbols for the coordination of corresponding verbalizations. As the example in Figure 2 already shows, we can just specify the symbol name in the optional argument of the `\defi` macro to establish that the language bindings provide different verbalizations of the same symbol.

For multilingual term references the situation is more complex: For single-word verbalizations we could use `\atrefi` for language bindings. Say we

¹EdNOTE: @DG: This needs to be implemented in LaTeXML

have introduced a symbol `foo` in English by `\defi{foo}` and in German by `\defi[foo]{Foo}`. Then we can indeed reference it via `\trefi{foo}` and `\atrefi{Foo}{foo}`. But on the one hand this blurs the distinction between translation and “linguistic variants” and on the other hand does not scale to multi-word compounds as `bar` in Figure 2, which we would have to reference as `\atrefiii{gro"ses Feld Raster}{bar}`. To avoid this, the `smultiling` package provides the new macros `\mtrefi`, `\mtrefii`, and `\mtrefiii` for multilingual references. Using this, we can reference `bar` as `\mtrefiii[?bar]{gro"ses}{Feld}{Raster}`, where we use the (up to three) mandatory arguments to segment the lexical constituents.

The first argument is syntactically optional to keep the parallelity to `*def*` `*tref*` it specifies the symbol via its name $\langle name \rangle$ and module name $\langle mod \rangle$ in a MMT URI $\langle mod \rangle ? \langle name \rangle$. Note that MMT URIs can be relative:

1. `foo?bar` denotes the symbol `bar` from module `foo`
2. `foo` the module `foo` (the symbol name is induced from the remaining arguments of `\mtref*`)
3. `?bar` specifies symbol `bar` from the current module

Note that the number suffix `i/ii/iii` indicates the number of words in the actual language binding, not in the symbol name as in `\atref*`.

2.3 Multilingual Views

Views receive a similar treatment as modules in the `smultiling` package. A multilingual view consists of a view signature marked up with the `viewsig` environment. This takes three required arguments: a view name, the source module, and the target module. The optional first argument is for metadata (display, title, creators, and contributors) and load information (frompath, fromrepos, topath, and torepos).²

```
\begin{viewsig}[creators=miko,]{norm-metric}{metric-space}{norm}
  \vassign{base-set}{base-set}
  \vassign{metric}{\funcdot{x,y}{\norm{x-y}}}
\end{viewsig}
```

Views have language bindings just as modules do, in our case, we have

```
\begin{gviewnl}[creators=miko]{norm-metric}{en}{norm}{metric-space}
  \obligation{metric-space}{obl.norm-metric.en}
  \begin{assertion}[type=obligation,id=obl.norm-metric.en]
    $\defeq{d(x,y)}{\norm{x-y}}$ is a \trefii[metric-space]{distance}{function}
  \end{assertion}
  \begin{sproof}[for=obl.norm-metric.en]
    {we prove the three conditions for a distance function:}
    ...
  \end{sproof}
```

²EdNOTE: MK: that does not work yet, what we describe here is `mhviewig`; we need to refactor further.

`\end{gviewnl}`

3 Implementation

Technically, the `smultiling` package is essentially a wrapper around the `babel` package but allows specification of languages by their ISO 639 language codes.

3.1 Class Options

To initialize the `smultiling` class, we pass on all options to `babel.cls` and record which languages are loaded by defining `\smul@⟨language⟩@loaded` macros.³

The `langfiles` option specifies that for a module `⟨mod⟩`, the module signature file has the name `⟨mod⟩.tex` and the language bindings of language with the ISO 639 language specifier `⟨lang⟩` have the file name `⟨mod⟩.⟨lang⟩.tex`.⁴

```

1 ⟨*sty⟩
2 \newif\if@langfiles\@langfilesfalse
3 \DeclareOption{langfiles}{\@langfilestrue}
4 \DeclareOption*{\PassOptionsToPackage{\CurrentOption}{babel}}
5 \@namedef{smul@⟨CurrentOption⟩@loaded}{yes}
6 \ProcessOptions
7 ⟨/sty⟩
8 ⟨*ltxml⟩
9 # -*- CPERL -*-
10 package LaTeXML::Package::Pool;
11 use strict;
12 use LaTeXML::Package;
13 DeclareOption('langfiles',sub {AssignValue('smultiling_langfiles',1,'global')});
14 DeclareOption(undef,sub {PassOptions('babel','sty',ToString(Digest(T_CS('⟨CurrentOption⟩')))); });
15 ProcessOptions();
16 ⟨/ltxml⟩

```

We load `babel.sty`

```

17 ⟨*sty⟩
18 \RequirePackage{etoolbox}
19 \RequirePackage{babel}
20 \RequirePackage{modules}
21 ⟨/sty⟩
22 ⟨*ltxml⟩
23 RequirePackage('babel');
24 RequirePackage('modules');
25 ⟨/ltxml⟩

```

3.2 Handling Languages

`\smg@select@language` This macro selects one of the registered languages by its language code by setting the internal `\smg@lang` macro to the argument and then runs the actual selection

³EDNOTE: @DG: We also want to do that in `LaTeXML`

⁴EDNOTE: implement other schemes, e.g. the onefile scheme.

code in `\smg@select@lang`. This internal code register is only initialized there, the code is generated by the `\smg@register@language` macro below.

```

26 <ltxml>RawTeX('
27 <*sty | ltxml>
28 \newcommand\smg@select@lang{}
29 \newcommand\smg@select@language[1]{\def\smg@lang{#1}\smg@select@lang}
```

`\smg@register@language` `\smg@register@language{<lang>}{<babel>}` registers the `babel` language name `<babel>` with its ISO 639 language code `<lang>` by extending the `\smg@select@language` macro.

```

30 \newcommand\smg@register@language[2]%
31 {\@ifundefined{smul@#1@loaded}{\appto\smg@select@lang%
32 {\expandafter\ifstrequal\expandafter\smg@lang{#1}{\selectlanguage{#2}}{}}}}
```

Now we register a couple of languages for which we have `babel` support. Maybe we have to extend this list with others. But then we have to extend the mechanisms.

```

33 \smg@register@language{af}{afrikaans}
34 \smg@register@language{de}{ngerman}
35 \smg@register@language{fr}{french}%
36 \smg@register@language{he}{hebrew}
37 \smg@register@language{hu}{hungarian}
38 \smg@register@language{id}{indonesian}
39 \smg@register@language{ms}{malay}
40 \smg@register@language{nn}{nynorsk}
41 \smg@register@language{pt}{portuguese}
42 \smg@register@language{ru}{russian}
43 \smg@register@language{uk}{ukrainian}
44 \smg@register@language{en}{english}
45 \smg@register@language{es}{spanish}
46 \smg@register@language{sq}{albanian}
47 \smg@register@language{bg}{bulgarian}
48 \smg@register@language{ca}{catalan}
49 \smg@register@language{hr}{croatian}
50 \smg@register@language{cs}{czech}
51 \smg@register@language{da}{danish}
52 \smg@register@language{nl}{dutch}
53 \smg@register@language{eo}{esperanto}
54 \smg@register@language{et}{estonian}
55 \smg@register@language{fi}{finnish}
56 \smg@register@language{ka}{georgian}
57 \smg@register@language{el}{greek}
58 \smg@register@language{is}{icelandic}
59 \smg@register@language{it}{italian}
60 \smg@register@language{la}{latin}
61 \smg@register@language{no}{norsk}
62 \smg@register@language{pl}{polish}
63 \smg@register@language{sr}{serbian}
64 \smg@register@language{sk}{slovak}
65 \smg@register@language{sl}{slovenian}
```

```

66 \smg@register@language{sv}{swedish}
67 \smg@register@language{th}{thai}
68 \smg@register@language{tr}{turkish}
69 \smg@register@language{vi}{vietnamese}
70 \smg@register@language{cy}{welsh}
71 \smg@register@language{hi}{hindi}

```

3.3 Signatures

modsig The `modsig` environment is just a layer over the `module` environment. We also redefine macros that may occur in module signatures so that they do not create markup.

```

72 \newenvironment{modsig}[2][]{%
73 \def\@test{#1}\ifx\@test\@empty\begin{module}[id=#2]\else\begin{module}[id=#2,#1]\fi}
74 {\end{module}}

```

viewsig The `viewsig` environment is just a layer over the `view` environment with the keys suitably adapted.

```

75 \newenvironment{viewsig}[4][]{\def\@test{#1}\ifx\@test\@empty%
76 \begin{view}[id=#2,ext=tex]{#3}{#4}\else\begin{view}[id=#2,#1,ext=tex]{#3}{#4}\fi}
77 {\end{view}}

```

mhviewsig The `mhviewsig` environment is just a layer over the `mhview` environment with the keys suitably adapted.

```

78 \newenvironment{mhviewsig}[4][]{\def\@test{#1}\ifx\@test\@empty%
79 \begin{mhview}[id=#2,ext=tex]{#3}{#4}\else\begin{mhview}[id=#2,#1,ext=tex]{#3}{#4}\fi}
80 {\end{mhview}}
81 \langle*sty | ltxml\rangle
82 \langle ltxml\rangle';

```

\@sym* has a starred form for primary symbols.

```

83 \langle*sty\rangle
84 \newcommand\symi{\@ifstar\@symi@star\@symi}
85 \newcommand\@symi[1]{\if@importing\else Symbol: \textsf{#1}\fi}
86 \newcommand\@symi@star[1]{\if@importing\else Primary Symbol: \textsf{#1}\fi}
87 \newcommand\symii{\@ifstar\@symii@star\@symii}
88 \newcommand\@symii[2]{\if@importing\else Symbol: \textsf{#1-#2}\fi}
89 \newcommand\@symii@star[2]{\if@importing\else Primary Symbol: \textsf{#1-#2}\fi}
90 \newcommand\symiii{\@ifstar\@symiii@star\@symiii}
91 \newcommand\@symiii[3]{\if@importing\else Symbol: \textsf{#1-#2-#3}\fi}
92 \newcommand\@symiii@star[3]{\if@importing\else Primary Symbol: \textsf{#1-#2-#3}\fi}
93 \langle/sty\rangle
94 \langle ltxml\rangle
95 DefConstructor('\symi OptionalMatch:* {}',
96   "<omdoc:symbol ?#1(role='primary')(role='secondary') name='#2'/>");
97 DefConstructor('\symii OptionalMatch:* {} {}',
98   "<omdoc:symbol ?#1(role='primary')(role='secondary') name='#2-#3'/>");
99 DefConstructor('\symiii OptionalMatch:* {} {} {}',
100   "<omdoc:symbol ?#1(role='primary')(role='secondary') name='#2-#3-#4'/>");

```



```
101 </ltxml>
```

3.4 Language Bindings

modnl:*

```
102 <*sty>
103 \addmetakey{modnl}{load}
104 \addmetakey*{modnl}{title}
105 \addmetakey*{modnl}{creators}
106 \addmetakey*{modnl}{contributors}
107 \addmetakey{primary}{contributors}[yes]
108 </sty>
109 <*ltxml>
110 DefKeyVal('modnl','title','Semiverbatim');
111 DefKeyVal('modnl','load','Semiverbatim');
112 DefKeyVal('modnl','creators','Semiverbatim');
113 DefKeyVal('modnl','contributors','Semiverbatim');
114 DefKeyVal('modnl','primary','Semiverbatim');
115 </ltxml>
```

modnl The modnl environment is just a layer over the module environment and the \importmodule macro with the keys and language suitably adapted.

```
116 <*sty>
117 \newenvironment{modnl}[3][\metasetkeys{modnl}{#1}%
118 \smg@select@language{#3}%
119 \def\@test{#1}\ifx\@test\@empty\begin{module}[id=#2.#3]\else\begin{module}[id=#2.#3,#1]\fi%
120 \if@langfiles\importmodule[load=#2,ext=tex]{#2}\else
121 \ifx\modnl@load\@empty\importmodule{#2}\else\importmodule[ext=tex,load=\modnl@load]{#2}\fi%
122 \fi}
123 {\end{module}}
124 </sty>
125 <*ltxml>
126 DefEnvironment('{modnl} OptionalKeyVals:modnl {}{}',
127     '?#excluded()(<omdoc:theory '
128     . 'xml:id="#2.#3">'
129     . ')?&defined(&GetKeyVal{#1,"creators"})(<dc:creator>&GetKeyVal{#1,"creators"}</dc:cr
130     . ')?&defined(&GetKeyVal{#1,"title"})(<dc:title>&GetKeyVal{#1,"title"}</dc:title>())'
131     . ')?&defined(&GetKeyVal{#1,"contributors"})(<dc:contributor>&GetKeyVal{#1,"contribut
132     . '<omdoc:imports from="#?&GetKeyVal{#1,"load"}(&canonical_omdoc_path(&GetKeyVal{#1,"
133     . '#body'
134     . '</omdoc:theory>))',
135 afterDigestBegin=>sub {
136     my ($stomach, $whatsit) = @_;
137     my $keyval = $whatsit->getArg(1);
138     my $signature = ToString($whatsit->getArg(2));
139     my $language = ToString($whatsit->getArg(3));
140     if ($keyval) {
141         # If we're not given load, AND the langfiles option is in effect,
```

```

142     # default to #2
143     if ((! $keyval->getValue('load')) && (LookupValue('smultiling_langfiles')) {
144         $keyval->setValue('load',$signature); }
145     # Always load a TeX file
146     $keyval->setValue('ext','tex');
147     $keyval->setValue('id',"signature.$language"); }
148     module_afterDigestBegin(@_);
149     importmoduleI(@_);
150     return; },
151     afterDigest =>\&module_afterDigest );
152 </ltxml>%$

mhmodnl:*
153 <*sty>
154 \addmetakey{mhmodnl}{repos}
155 \addmetakey{mhmodnl}{path}
156 \addmetakey*{mhmodnl}{title}
157 \addmetakey*{mhmodnl}{creators}
158 \addmetakey*{mhmodnl}{contributors}
159 \addmetakey{primary}{contributors}[yes]
160 </sty>
161 <*ltxml>
162 DefKeyVal('mhmodnl','title','Semiverbatim');
163 DefKeyVal('mhmodnl','repos','Semiverbatim');
164 DefKeyVal('mhmodnl','path','Semiverbatim');
165 DefKeyVal('mhmodnl','creators','Semiverbatim');
166 DefKeyVal('mhmodnl','contributors','Semiverbatim');
167 DefKeyVal('mhmodnl','primary','Semiverbatim');
168 </ltxml>

mhmodnl The mhmodnl environment is just a layer over the module environment and the
\importmhmodule macro with the keys and language suitably adapted.
169 <*sty>
170 \newenvironment{mhmodnl}[3][\metasetkeys{mhmodnl}{#1}%
171 \smg@select@language{#3}%
172 \def\@test{#1}\ifx\@test\@empty\begin{module}[id=#2.#3]\else\begin{module}[id=#2.#3,#1]\fi%
173 \edef\@repos{\ifx\mhmodnl@repos\@empty\mh@currentrepos\else\mhmodnl@repos}
174 \if@langfiles\importmhmodule[repos=\@repos,load=#2,ext=tex]{#2}\else
175 \ifx\mhmodnl@load\@empty\importmodule{#2}\else\importmodule[ext=tex,load=\mhmodnl@load]{#2}\fi%
176 \fi}
177 {\end{module}}
178 </sty>
179 <*ltxml>
180 DefEnvironment('{mhmodnl} OptionalKeyVals:mhmodnl {}{}',
181     '?#excluded()(<omdoc:theory '
182     . 'xml:id="#2.#3" >'
183     . ')?&defined(&GetKeyVal(#1,"creators"))(<dc:creator>&GetKeyVal(#1,"creators")</dc:cr
184     . ')?&defined(&GetKeyVal(#1,"title"))(<dc:title>&GetKeyVal(#1,"title")</dc:title>())'
185     . ')?&defined(&GetKeyVal(#1,"contributors"))(<dc:contributor>&GetKeyVal(#1,"contribut
186     . '<omdoc:imports from="#?&GetKeyVal(#1,"load")(&canonical_omdoc_path(&GetKeyVal(#1,"

```

```

187         . '#body'
188         . '</omdoc:theory>)',
189   afterDigestBegin=>sub {
190     my ($stomach, $whatsit) = @_;
191     my $keyval = $whatsit->getArg(1);
192     my $signature = ToString($whatsit->getArg(2));
193     my $language = ToString($whatsit->getArg(3));
194     my $repos = ToString(GetKeyVal($keyval, 'torepos'));
195     my $current_repos = LookupValue('current_repos');
196     if (!$repos) { $repos = $current_repos; }
197     my $defpaths = LookupValue('defpath');
198     my $load_path = ($$defpaths{MathHub}).$repos.'/source/'.$signature;
199
200     if ($keyval) {
201       # If we're not given load, AND the langfiles option is in effect,
202       # default to #2
203       if ((! $keyval->getValue('path')) && (LookupValue('smultiling_langfiles')))) {
204         $keyval->setValue('load', $load_path); }
205       # Always load a TeX file
206       $keyval->setValue('ext', 'tex');
207       $keyval->setValue('id', "$signature.$language"); }
208     module_afterDigestBegin(@_);
209     importmoduleI(@_);
210     return; },
211   afterDigest=>sub {
212     module_afterDigest(@_); });
213 </ltxml>%$

```

EdN:5

viewnl The **viewnl** environment is just a layer over the **viewsketch** environment with the keys and language suitably adapted.⁵

```

214 <ltxml>RawTeX('
215 <*sty | ltxml>
216 \newenvironment{viewnl}[5][\def\@test{#1}\ifx\@test\@empty%
217 \begin{viewsketch}[id=#2.#3,ext=tex]{#4}{#5}\else%
218 \begin{viewsketch}[id=#2.#3,#1,ext=tex]{#4}{#5}\fi%
219 \smg@select@language{#3}}
220 {\end{viewsketch}}

```

EdN:6

mhviewnl The **mhviewnl** environment is just a layer over the **mhviewsketch** environment with the keys and language suitably adapted.⁶

```

221 \newenvironment{mhviewnl}[5][\def\@test{#1}\ifx\@test\@empty%
222 \begin{mhviewsketch}[id=#2.#3,ext=tex]{#4}{#5}\else%
223 \begin{mhviewsketch}[id=#2.#3,#1,ext=tex]{#4}{#5}\fi%
224 \smg@select@language{#3}}
225 {\end{mhviewsketch}}

```

⁵EDNOTE: MK: we have to do something about the if@langfiles situation here. But this is non-trivial, since we do not know the current path, to which we could append `.(lang)!`

⁶EDNOTE: MK: we have to do something about the if@langfiles situation here. But this is non-trivial, since we do not know the current path, to which we could append `.(lang)!`

```

226 </sty | ltxml>
227 <ltxml>');

```

3.5 Multilingual Statements and Terms

`\mtref*` we first first define an auxiliary conditional `\@instring` that checks if `?` is in the first argument. `\mtrefi` uses it, if there is one, it just calls `\termref`, otherwise it calls `\@mtrefi`, which assembles the `\termref` after splitting at the `?`.

```

228 <ltxml>RawTeX('
229 <*sty | ltxml>
230 \def\@instring#1#2{TT\fi\begin{group}\edef\x{\end{group}\noexpand\in@{#1}{#2}}\x\ifin@}
231 \newcommand\mtrefi[2][\if\@instring{?}{#1}\@mtref #1\relax{#2}\else\termref[cd=#1]{#2}\fi}
232 \def\@mtref#1#2\relax{\termref[cd=#1,name=#2]}
233 \newcommand\mtrefii[3][\mtrefi[#1]{#2 #3}]
234 \newcommand\mtrefiii[4][\mtrefi[#1]{#2 #3 #4}]
235 </sty | ltxml>
236 <ltxml>');

```

3.6 Finale

Finally, we need to terminate the file with a success mark for perl.

```

237 <ltxml>1;

```

References

- [Gin+14] Deyan Ginev et al. “The SMGLoM Project and System”. 2014. URL: <http://kwarc.info/kohlhase/submit/cicm14-smglom-system.pdf>.
- [Koh14] Michael Kohlhase. “A Data Model and Encoding for a Semantic, Multilingual Glossary of Mathematics”. In: *Intelligent Computer Mathematics*. Conferences on Intelligent Computer Mathematics. (Coimbra, Portugal, July 7–11, 2014). Ed. by Stephan Watt et al. Lecture Notes in Computer Science. accepted. Springer, 2014. URL: <http://kwarc.info/kohlhase/submit/cicm14-smglom-datamd1.pdf>. Forthcoming.