Semantic Markup for Mathematical Statements*

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

May 7, 2008

Abstract

The `statements` packge is part of the sTEX collection, 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).

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.

*Version v0.9d (last revised 2007/09/09)

1 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).

2 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 1 for an example).[1]

EdNote(1)

```
\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 1: 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

[1] 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[2] the semantic information is invisible to the reader.

EdNote(2)

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` 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}`.[3]

`\definiendum`

EdNote(3)

`\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[1]. 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[4]

EdNote(4)

`\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-

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

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

[1] 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`.

[4] 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.

3 The Implementation

We declare some switches which will modify the behavior according to the package options. Generally, an option `xxx` will just set the appropriate switches to true (otherwise they stay false). First we have the general options

```
⟨*package⟩
\newif\ifst@env\st@envfalse
\newif\ifst@id\st@idfalse
\newif\ifst@display\st@displayfalse
\DeclareOption{id}{\st@idtrue\st@envtrue}
\DeclareOption{env}{\st@envtrue}
\DeclareOption{display}{\st@displaytrue\st@envtrue}
```

And then the options that are specific to the `statements` package.

```
\newif\ifstat@for\stat@forfalse
\newif\ifstat@from\stat@forfalse
\newif\ifstat@type\stat@typefalse
\newif\ifstat@title\stat@titlefalse
\newif\ifstat@continues\stat@continuesfalse
\DeclareOption{for}{\stat@fortrue\st@envtrue}
\DeclareOption{from}{\stat@fromtrue\st@envtrue}
\DeclareOption{type}{\stat@typetrue\st@envtrue}
\DeclareOption{title}{\stat@titletrue\st@envtrue}
\DeclareOption{continues}{\stat@continuestrue\st@envtrue}
```

`\stattrue` For convenience, we collect the switches into one.

```
\def\stattrue{\stat@fortrue\stat@fromtrue\stat@typetrue\stat@continuestrue}
```

Now, we define two collective options, which are equivalent to turning on all the other options.

```
\DeclareOption{draft}{\st@envtrue\st@idtrue\stat@fortrue\stat@fromtrue\typetrue\justtrue}
\DeclareOption{all}{\st@envtrue\stattrue\justtrue}
```

Finally, we need to declare the end of the option declaration section to LaTeX.

```
\ProcessOptions
⟨/package⟩
```

The next measure is to ensure that the `omdoc` package is loaded (in the right version). For LaTeXML, we also initialize the package inclusions.

```
⟨package⟩\RequirePackage{omdoc}[2007/09/09]
⟨*ltxml⟩
# -*- CPERL -*-
package LaTeXML::Package::Pool;
use strict;
use LaTeXML::Package;
RequirePackage('omdoc');
⟨/ltxml⟩
```

`\define@statement@env` We define a meta-macro that allows us to define several variants of statements. Upon beginning this environment, we first set the `KeyVal` attributes, then we decide whether to print the discourse marker based on the value of the `display` key, then (given the right Options were set), we show the semantic annotations, and finally initialize the environment using the appropriate macro. Upon ending the environment, we just run the respcective termination macro.

```
⟨*package⟩
\def\define@statement@env#1#2{
\newenvironment{#1}[1][]{\setkeys{stat}{##1}
\ifx\st@display\st@flow\else\stDMemph{#2}:\fi%
\@ifundefined{stat@title}{}%
{\space\ifx\st@display\st@flow\else(\fi\stDMemph{\stat@title}\ifx\st@display\st@flow:\else)\par
\ifst@env\show@stat@keys{#1}\fi\csname st@#1@initialize\endcsname}
{\csname st@#1@terminate\endcsname}}
⟨/package⟩
```

`assertion`

```
⟨*package⟩
\newenvironment{assertion}[1][]{\setkeys{stat}{#1}
\ifx\st@display\st@flow\else{\stDMemph{\stat@type}:}\fi%
\@ifundefined{stat@title}{~}{~(\stDMemph{\stat@title})\par}%
\ifst@env\show@stat@keys{#1}\fi\em}{}
⟨/package⟩
⟨*ltxml⟩
DefCMPEnvironment('{assertion} OptionalKeyVals:stat',
  "<omdoc:assertion ?&KeyVal(#1,'id')(xml:id='&KeyVal(#1,'id')')() type='&KeyVal(#1,'type')'>"
  . "?&KeyVal(#1,'title')(<omdoc:metadata><dc:title>&KeyVal(#1,'title')</dc:title></omdoc:metad
  . "<omdoc:CMP><omdoc:p>#body</omdoc:p></omdoc:CMP>"
  . "</omdoc:assertion>\n");
⟨/ltxml⟩
```

`simpleDef`

```
⟨*package⟩
\newenvironment{simpleDef}[1][]{\setkeys{stat}{#1}
\ifx\st@display\st@flow\else{\stDMemph{Definition}:}\fi%
\@ifundefined{stat@title}{~}{~(\stDMemph{\stat@title})\par}%
\ifst@env\show@stat@keys{#1}\fi}{}
⟨/package⟩
⟨*ltxml⟩
DefCMPEnvironment('{simpleDef} OptionalKeyVals:stat',
  "?&KeyVal(#1,'for')(<omdoc:symbol name='&KeyVal(#1,'for')'/>)()"
  . "<omdoc:definition type='simple' "
    . "?&KeyVal(#1,'id')(xml:id='&KeyVal(#1,'id').def')()"
    . "?&KeyVal(#1,'for')(for='&KeyVal(#1,'for')')()>"
    . "?&KeyVal(#1,'title')(<omdoc:metadata><dc:title>&KeyVal(#1,'title')</dc:title></omdoc:met
    . "<omdoc:CMP><omdoc:p>#body</omdoc:p></omdoc:CMP>"
  . "</omdoc:definition>\n");
⟨/ltxml⟩
```

PatternDef

```
⟨*package⟩
\newenvironment{PatternDef}[1][]{\setkeys{stat}{#1}
\ifx\st@display\st@flow\else{\stDMemph{Definition}:}\fi%
\@ifundefined{stat@title}{~}{~(\stDMemph{\stat@title})\par}%
\ifst@env\show@stat@keys{#1}\fi}{}
\newenvironment{PatternRule}[1]{#1$\colon=$}{}
\newenvironment{PatternCMP}{}{}
⟨/package⟩
⟨*ltxml⟩
DefCMPEnvironment('{PatternDef} OptionalKeyVals:stat',
  "?&KeyVal(#1,'for')(<omdoc:symbol name='&KeyVal(#1,'for')'/>)()"
  . "<omdoc:definition type='pattern' "
    . "?&KeyVal(#1,'id')(xml:id='&KeyVal(#1,'id').def')()"
    . "?&KeyVal(#1,'for')(for='&KeyVal(#1,'for')')()>"
    . "?&KeyVal(#1,'title')(<omdoc:metadata><dc:title>&KeyVal(#1,'title')</dc:title></omdoc:met
    . "#body"
  . "</omdoc:definition>\n");
DefEnvironment('{PatternRule}{}',
       "<omdoc:requation>#1 #body</omdoc:requation>");
DefEnvironment('{PatternCMP}{}',
       "<omdoc:CMP>#body</omdoc:CMP>");
⟨/ltxml⟩
```

RecDef

```
⟨*package⟩
\newenvironment{RecDef}[1][]{\setkeys{stat}{#1}
\ifx\st@display\st@flow\else{\stDMemph{Definition}:}\fi%
\@ifundefined{stat@title}{~}{~(\stDMemph{\stat@title})\par}%
\ifst@env\show@stat@keys{#1}\fi}{}
⟨/package⟩
⟨*ltxml⟩
DefEnvironment('{RecDef} OptionalKeyVals:stat',
  "?&KeyVal(#1,'for')(<omdoc:symbol name='&KeyVal(#1,'for')'/>)()"
  . "<omdoc:definition type='inductive' "
    . "?&KeyVal(#1,'id')(xml:id='&KeyVal(#1,'id').def')()"
    . "?&KeyVal(#1,'for')(for='&KeyVal(#1,'for')')()>"
    . "?&KeyVal(#1,'title')(<omdoc:metadata><dc:title>&KeyVal(#1,'title')</dc:title></omdoc:met
    . "<omdoc:CMP><omdoc:p>#body</omdoc:p></omdoc:CMP>"
  . "</omdoc:definition>\n");
⟨/ltxml⟩
```

example

```
⟨*package⟩
\def\st@example@initialize{}\def\st@example@terminate{}
\define@statement@env{example}{Example}
⟨/package⟩
⟨*ltxml⟩
DefCMPEnvironment('{example} OptionalKeyVals:stat',
```

```
        "<omdoc:example "
        . "?&KeyVal(#1,'id')(xml:id='&KeyVal(#1,'id')')() "
        . "for='&KeyVal(#1,'for')'>"
        . "<omdoc:CMP><omdoc:p>#body</omdoc:p></omdoc:CMP>"
      . "</omdoc:example>\n");
⟨/ltxml⟩
```

axiom

```
⟨*package⟩
\def\st@axiom@initialize{}\def\st@axiom@terminate{}
\define@statement@env{axiom}{Axiom}
⟨/package⟩
⟨*ltxml⟩
DefCMPEnvironment('{axiom} OptionalKeyVals:stat',
  "<omdoc:axiom ?&KeyVal(#1,'id')(xml:id='&KeyVal(#1,'id')')()>"
  . "?&KeyVal(#1,'title')(<omdoc:metadata><dc:title>&KeyVal(#1,'title')</dc:title></omdoc:metad
  . "<omdoc:CMP><omdoc:p>#body</omdoc:p></omdoc:CMP>"
  . "</omdoc:axiom>\n");
⟨/ltxml⟩
```

consymb

```
⟨*package⟩
\define@key{consymb}{type}{\def\consymb@type{#1}}
\define@key{consymb}{title}{\def\consymb@title{#1}}
\define@key{consymb}{name}{\def\consymb@name{#1}}
\def\consymb@type{Symbol}
\newenvironment{consymb}[1][]{\setkeys{consymb}{#1}
\ifx\st@display\st@flow\else{\stDMemph{\consymb@type} \consymb@name:}\fi%
\@ifundefined{consymb@title}{~}{~(\stDMemph{\consymb@title})\par}}{}
⟨/package⟩
⟨*ltxml⟩
DefEnvironment('{consymb} OptionalKeyVals:stat',
       "<omdoc:symbol ?&KeyVal(#1,'id')(xml:id='&KeyVal(#1,'id')')()"
             .              "name='&KeyVal(#1,'name')'>"
             .   "<omdoc:metadata>"
             .      "<dc:description>"
     .        "#body"
             .      "</dc:description>"
             .   "</omdoc:metadata>"
     . "</omdoc:symbol>\n");
⟨/ltxml⟩
```

\symtype

```
⟨package⟩\newcommand{\symtype}[2]{Type (#1): #2}
⟨ltxml⟩DefConstructor('\symtype{}{}',"<omdoc:type system='#1'>#2</omdoc:type>");
```

definition The `definition` environment itself is quite simple. The only interesting thing is that it locally defines the `definiendum` macro, which we do in the initialization macro.

\definiendum The \definiendum macro is very simple: at the moment we do not do anything with the keyword arguments.

```
⟨*package⟩
\def\st@definition@initialize{%
\newcommand{\notatiendum}[2][]{\notemph{##2}}
\newcommand{\definiendum}[2][]{\defemph{##2}}}
\def\st@definition@terminate{}
\define@statement@env{definition}{Definition}
⟨/package⟩
⟨*ltxml⟩
DefCMPEnvironment('{definition} OptionalKeyVals:stat', sub {
    my ($doc, $keyvals, %props) = @_;
    my @symbols = @{$props{defs} || []};
    foreach my $symb(@symbols) {
      $doc->openElement('omdoc:symbol', name=>$symb);
      $doc->closeElement('omdoc:symbol'); }
    my %attrs = ();
    my $id = $keyvals->getValue('id') if $keyvals;
    $attrs{'xml:id'} = $id if $id;
    my $ffor = $keyvals->getValue('for') if $keyvals;
    my $for = $ffor . join(' ',@symbols);
    $attrs{for} = $for if $for;
    $doc->openElement('omdoc:definition', %attrs);
    my $title = $keyvals->getValue('title') if $keyvals;
    if ($title) {
      $doc->openElement('omdoc:metadata');
      $doc->openElement('dc:title');
      $doc->absorb($title);
      $doc->closeElement('dc:title');
      $doc->closeElement('omdoc:metadata'); }
    $doc->openElement('omdoc:CMP');
    $doc->openElement('omdoc:p');
    $doc->absorb($props{body}) if $props{body};
    $doc->closeElement('omdoc:p');
    $doc->closeElement('omdoc:CMP');
    $doc->closeElement('omdoc:definition');
    return; },
  afterDigestBegin=>sub {
    my ($stomach, $whatsit) = @_;
    my @symbols = ();
    $whatsit->setProperty(defs=>\@symbols);
    AssignValue('defs', \@symbols, 'global');
    return; },
  afterDigest => sub {
    AssignValue('defs', undef, 'global');
    return; });

DefConstructor('\definiendum[]{}',
       "<omdoc:term role='definiendum' name='#1' cd='#theory'>#2</omdoc:term>",
```

```
199       afterDigest => sub {
200  my ($stomach, $whatsit) = @_;
201  my $addr = LookupValue('defs');
202  push(@$addr, $whatsit->getArg(1)->toString) if $addr;
203  $whatsit->setProperty(theory=>LookupValue('current_module'));
204  return; });
205 ⟨/ltxml⟩
```

We expand the LaTeXML bindings for `\defin`, `\twindef` and `\atwindef` into two instances one will be used for the definition and the other for indexing

```
206 ⟨*ltxml⟩
207 DefMacro('\defin{}', sub {
208    my @args = (T_BEGIN, $_[1]->unlist, T_END);
209    (T_CS('\definiendum'), T_OTHER('['), $_[1]->unlist, T_OTHER(']'), @args, T_CS('\defin@index'
210 DefMacro('\twindef{}{}', sub {
211    my @args = (T_BEGIN, $_[1]->unlist, T_END, T_BEGIN, $_[2]->unlist, T_END);
212    (T_CS('\twindef@def'), @args, T_CS('\twindef@index'), @args); });
213 DefMacro('\atwindef{}{}{}', sub {
214    my @args = (T_BEGIN, $_[1]->unlist, T_END, T_BEGIN, $_[2]->unlist, T_END,
215        T_BEGIN, $_[3]->unlist, T_END);
216    (T_CS('\atwindef@def'), @args, T_CS('\atwindef@index'), @args); });
217 DefConstructor('\twindef@def{}{}',
218       "<omdoc:term role='definiendum' name='#1-#2' cd='#theory'>#1 #2</omdoc:term>",
219       afterDigest => sub {
220  my ($stomach, $whatsit) = @_;
221  my $addr = LookupValue('defs');
222  push(@$addr, $whatsit->getArg(1)->toString.'-'.$whatsit->getArg(2)->toString) if $addr;
223  $whatsit->setProperty(theory=>LookupValue('current_module'));
224  return; },
225       alias => '');
226 DefConstructor('\atwindef@def{}{}{}',
227       "<omdoc:term role='definiendum' name='#1-#2-#3' cd='#theory'>#1 #2 #3</omdoc:term>",
228       afterDigest => sub {
229  my ($stomach, $whatsit) = @_;
230  my $addr = LookupValue('defs');
231  push(@$addr, $whatsit->getArg(1)->toString.'-'.$whatsit->getArg(2)->toString
232       .'-'.$whatsit->getArg(3)->toString) if $addr;
233  $whatsit->setProperty(theory=>LookupValue('current_module'));
234  return; },
235       alias => '');
236 ⟨/ltxml⟩
```

notation

```
237 ⟨*package⟩
238 \def\notemph#1{{\bf{#1}}}
239 \def\st@notation@initialize{\newcommand{\notatiendum}[2][]{\notemph{##2}}}
240 \def\st@notation@terminate{}
241 \define@statement@env{notation}{Notation}
242 ⟨/package⟩
```

```
⟨*ltxml⟩
DefCMPEnvironment('{notation} OptionalKeyVals:stat',
  "<omdoc:omtext type='notation' ?&KeyVal(#1,'id')(xml:id='&KeyVal(#1,'id').not')()>"
  . "?&KeyVal(#1,'title')(<omdoc:metadata><dc:title>&KeyVal(#1,'title')</dc:title></omdoc:metad
  . "<omdoc:CMP><omdoc:p>#body</omdoc:p></omdoc:CMP>"
  . "</omdoc:omtext>\n");
DefConstructor('\notatiendum OptionalKeyVals:notation {}',
               "<omdoc:phrase type='notation'>#1</omdoc:phrase>");
⟨/ltxml⟩
```

EdNote(5)

`\termin` The `termin` macro is very simple, it forgets the semantic annotations[5] and puts the

```
⟨*package⟩
\def\termin{\@ifnextchar[{\@termin}{\@termin[]}}
\def\@termin[#1]#2{{\termemph{\index*{#2}}}}
⟨/package⟩
```

Now we care about the configuration switches, they are set to sensible values, if they are not defined already. These are just configuration parameters, which should not appear in documents, therefore we do not provide LaTeXML bindings for them.

```
⟨*package⟩
\providecommand{\termemph}[1]{#1}
\providecommand{\defemph}[1]{{\bf{#1}}}
\providecommand{\stDMemph}[1]{{\bf{#1}}}
⟨/package⟩
```

3.1 Providing IDs for OMDoc Elements

To provide default identifiers, we tag all OMDoc elements that allow `xml:id` attributes by excuting the `numberIt` procedure from `omdoc.sty.ltxml`.

```
⟨*ltxml.sty⟩
Tag('omdoc:assertion',afterOpen=>\&numberIt);
Tag('omdoc:definition',afterOpen=>\&numberIt);
⟨/ltxml.sty⟩
```

3.2 Finale

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

```
⟨ltxml⟩1;
```

[5] EdNote: use those for hyperlinking in the future