`cmath.sty`: An Infrastructure for building Inline Content Math in sTeX[*]

Michael Kohlhase & Deyan Ginev
Jacobs University, Bremen
`http://kwarc.info/kohlhase`

January 13, 2013

Abstract

The `cmath` package is a central 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 supplies an infrastructure that allows to build content math expresions (strict content MathML or OpenMath objects) in the text. This is needed whenever the head symbols of expressions are variables and can thus not be treated via the `\symdef` mechanism in sTeX.

[*]Version v0.1 (last revised 2012/09/23)

Contents

1 Introduction

sTeX allows to build content math expressions via the `\symdef` mechanism [**KohAmb:smmssl:ctan**] if their heads are constants. For instance, if we have defined `\symdef{lt}[2]{#1<#2}` in the module `relation1`, then an invocation of `\lt3a` will be transformed to

```
<OMA>
  <OMS cd="relation1" name="lt"/>
  <OMI>3</OMI>
  <OMV name="a"/>
</OMA>
```

If the head of the expression (i.e. the function symbol in this case) is a variable, then we cannot resort to a `\symdef`, since that would define the functional equivalent of a logical constant. Sometimes, LaTeXML can figure out that when we write $f(a,b)$ that f is a function (especially, if we declare them to be via the `functions=` key in the dominating statement environment [**Kohlhase:smmtf:ctan**]). But sometimes, we want to be explicit, especially for n-ary functions and in the presence of elided elements in argument sequences. A related problem is markup for complex variable names, such as x_{left} or ST^*.

The `cmath` package supplies the LaTeX bindings that allow us to achieve this.

2 The User Interface

2.1 Variable Names

In mathematics we often use complex variable names like x', g_n, f^1, $\widetilde{\phi}_i^j$ or even foo; for presentation-oriented LaTeX, this is not a problem, but if we want to generate content markup, we must show explicitly that those are complex identifiers (otherwise the variable name foo might be mistaken for the product $f\cdot o\cdot o$). In careful mathematical typesetting, `$sin$` is distinguished from `$\sin$`, but we cannot rely on this effect for variable names.

`\vname` `\vname` identifies a token sequence as a name, and allows the user to provide 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 `\vname` with `\vnref` for referencing. So, if we have used `\vnname[xi]{x_i}`, then we can later use `\vnref{xi}` as a short name for `\vname{x_i}`. Note that in output formats that are capable of generating structure sharing, `\vnref{xi}` would be EdN:1 represented as a cross-reference.[1]

Since indexed variable names make a significant special case of complex identi- `\livar` fiers, we provides the macros `\livar` that allows to mark up variables with lower indices. If `\livar` is given an optional first argument, this is taken as a name. `\livar` Thus `\livar[foo]{x}1` is "short" for `\vname[foo]{x_1}`. The macros `\livar`,

[1] EdNote: DG: Do we know whether using the same name in two vname invocations, would refer to two instances of the same variable? Presumably so, since the names are the same? We should make this explicit in the text. A different variable would e.g. have a name "xi2", but the same body

`\nappa{f}{a_1,a_2,a_3}`	$f(a_1, a_2, a_3)$
`\nappe{f}{a_1}{a_n}`	$f(a_1, \ldots, a_n)$
`\symdef{eph}[1]{e_{#1}^{\varphi(#1)}}\nappf{g}\eph14`	$g(e_1^{\varphi(1)}, \ldots, e_4^{\varphi(4)})$
`\nappli{f}a1n`	$f(a_1, \ldots, a_n)$

Figure 1: Application Macros

`\ulivar` `\primvar` `\pprimvar` serve the analogous purpose for variables with upper indices, and `\ulivar` for upper and lower indices. Finally, `\primvar` and `\pprimvar` do the same for variables with primes and double primes (triple primes are bad style).

2.2 Applications

`\nappa` `\nappf` To construct a content math application of the form $f(a_1, \ldots, a_n)$ with concrete arrugments a_i (i.e. without elisions), then we can use the `\nappa` maro. If we have elisions in the arguments, then we have to interpret the arguments as a sequence of argument constructors applied to the respective positional indexes. We can mark up this situation with the `\nappf` macro: `\nappf{`⟨*fun*⟩`}{`⟨*const*⟩`}{`⟨*first*⟩`}{`⟨*last*⟩`}` where ⟨*const*⟩ is a macro for the constructor is presented as ⟨*fun*⟩(⟨*const*⟩⟨*first*⟩, . . . , ⟨*const*⟩⟨*last*⟩); see Figure 1 for a concrete example, and Figure 1.[2]

EdN:2

`\nappe` For a simple elision in the argument list, we can use `\nappe` macro: `\nappe{`⟨*fun*⟩`}{`⟨*firstarg*⟩`}{`⟨*lastarg*⟩ will be formatted as ⟨*fun*⟩(⟨*firstarg*⟩, . . . , ⟨*lastarg*⟩). Note that this is quite unsemantic (we have to guess the sequence), so the use of `\nappe` is discouraged.

2.3 Binders

3 Limitations

In this section we document known limitations. If you want to help alleviate them, please feel free to contact the package author. Some of them are currently discussed in the sTeX TRAC [**sTeX:online**].

1. none reported yet

4 The Implementation

The `cmath` package generates to files: the LaTeX package (all the code between ⟨*package⟩ and ⟨/package⟩) and the LaTeXML bindings (between ⟨*ltxml⟩ and ⟨/ltxml⟩). We keep the corresponding code fragments together, since the documentation applies to both of them and to prevent them from getting out of sync.

For LaTeXML, we initialize the package inclusions.

```
1 ⟨*ltxml⟩
```

[2] EdNote: MK@MK: we need a meta-cd `cmath` with the respective notation definition here. It is very frustrating that we cannot even really write down the axiomatization of

```
\symdef{eph}[1]{e_{#1}^{\phi(#1)}}
\nappf{g}\eph14
```

currently generates

```
<OMA>
  <OMS cd="cmath" name="apply-from-to"/>
  <OMV name="g"/>
  <OMBIND>
    <OMS cd="fns1" name="lambda"/>
    <OMBVAR><OMV name="x"/></OMBVAR>
    <OMA><OMS cd="???" name="eph"/><OMV name="x"/></OMA>
  </OMBIND>
  <OMI>1</OMI>
  <OMI>4</OMI>
</OMA>
```

Example 1: Application Macros

```
# -*- CPERL -*-
package LaTeXML::Package::Pool;
use strict;
use LaTeXML::Package;
⟨/ltxml⟩
```

4.1 Package Options

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).[3] EdN:3

```
⟨*package⟩
\ProcessOptions
⟨/package⟩
```

4.2 Variable Names

`\vname` a name macro; the first optional argument is an identifier ⟨*id*⟩, this is standard for LaTeX, but for LaTeXML, we want to generate attributes `xml:id="cvar.`⟨*id*⟩`"` and `name="`⟨*id*⟩`"`. However, if no id was given in we default them to `xml:id="cvar.`⟨*count*⟩`"` and `name="name.cvar.`⟨*count*⟩`"`.

```
⟨*package⟩
\newcommand\vname[2][]{#2%
\def\@opt{#1}%
\ifx\@opt\@empty\else\expandafter\gdef\csname MOD@name@#1\endcsname{#2}\fi}
```

[3] EdNote: we have no options at the moment

```
⟨/package⟩
⟨*ltxml⟩
# return: unique ID for variable
sub cvar_id {
  my ($id) = @_;
  $id = ToString($id);
  if (!$id) {
    $id=LookupValue('cvar_id') || 0;
    AssignValue('cvar_id', $id + 1, 'global'); }
  "cvar.$id"; }#$
DefConstructor('\vname[]{}',
  "<ltx:XMWrap role='ID' xml:id='&cvar_id(#1)'>#2</ltx:XMWrap>",
  requireMath=>1);
⟨/ltxml⟩
```

\vnref

```
⟨*package⟩
\def\vnref#1{\csname MOD@name@#1\endcsname}
⟨/package⟩
⟨*ltxml⟩
# \vnref{<reference>}
DefMacro('\vnref{}','\@XMRef{cvar.#1}');

⟨/ltxml⟩
```

EdN:4 [4]

\uivar constructors for variables.

```
⟨*package⟩
\newcommand\primvar[2][]{\vname[#1]{#2^\prime}}
\newcommand\pprimvar[2][]{\vname[#1]{#2^{\prime\prime}}}
\newcommand\uivar[3][]{\vname[#1]{{#2}^{#3}}}
\newcommand\livar[3][]{\vname[#1]{{#2}_{#3}}}
\newcommand\ulivar[4][]{\vname[#1]{{#2}^{#3}_{#4}}}
⟨/package⟩
⟨*ltxml⟩
# variants for declaring variables
DefMacro('\uivar[]{}{}',    '\vname[#1]{{#2}^{#3}}');
DefMacro('\livar[]{}{}',    '\vname[#1]{{#2}_{#3}}');
DefMacro('\ulivar[]{}{}{}', '\vname[#1]{{#2}^{#3}_{#4}}');
DefMacro('\primvar[]{}',    '\vname[#1]{#2^\prime}');
DefMacro('\pprimvar[]{}',   '\vname[#1]{#2^{\prime\prime}}');

⟨/ltxml⟩
```

4.3 Applications

\napp*

[4] EdNote: the following macros are just ideas, they need to be implemented and documented

```
⟨*package⟩
\newcommand\nappa[2]{#1(#2)}
\newcommand\nappe[3]{\nappa{#1}{#2,\ldots,#3}}
\newcommand\nappf[4]{\nappe{#1}{#2{#3}}{#2{#4}}}
\newcommand\nappli[4]{\nappe{#1}{#2_{#3}}{#2_{#4}}}
\newcommand\nappui[4]{\nappe{#1}{#2^{#3}}{#2^{#4}}}
⟨/package⟩
⟨*ltxml⟩
# \nappa{<function>}{<(const)(,\1)*>}
# @#1(#2)
DefConstructor('\nappa{}{}',
  "<ltx:XMApp>"
    ."<ltx:XMTok meaning='#1' />"
    ."<ltx:XMArg>#2</ltx:XMArg>"
  ."</ltx:XMApp>");

# \@napp@seq{<function>}{start <const>}{end <const>}
# @#1(@sequence(#2,sequencefromto,#3))
DefConstructor('\@napp@seq{}{}{}',
  "<ltx:XMApp>"
  ."<ltx:XMTok meaning='#1' />"
  ."<ltx:XMArg>"
    ."<ltx:XMApp>"
      ."<ltx:XMTok meaning='sequence' />"
      ."<ltx:XMArg>#2</ltx:XMArg>"
      ."<ltx:XMArg><ltx:XMTok meaning='sequencefromto' /></ltx:XMArg>"
      ."<ltx:XMArg>#3</ltx:XMArg>"
    ."</ltx:XMApp>"
  ."</ltx:XMArg>"
  ."</ltx:XMApp>");

DefMacro('\nappe{}{}{}',    '\@napp@seq{#1}{#2}{#3}');
DefMacro('\nappf{}{}{}{}',  '\@napp@seq{#1}{#2{#3}}{#2{#4}}');
DefMacro('\nappli{}{}{}{}', '\@napp@seq{#1}{#2_{#3}}{#2_{#4}}');
DefMacro('\nappui{}{}{}{}', '\@napp@seq{#1}{#2^{#3}}{#2^{#4}}');

⟨/ltxml⟩
```

4.4 Binders

4.5 Finale

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

```
⟨ltxml⟩1;
```