Experimental unicode mathematical typesetting: The unicode-math package

Will Robertson

2006/02/20 v0.01

Contents

1 Introduction

This document describes the unicode-math package, which is an *experimental* implementation of a macro to unicode glyph encoding for mathematical characters. Its intended use is for XeTeX, although it is conjectured that small effect needs to be spent to create a cross-format package that would also work with Omega.

As of XeTeX v. 0.995, maths characters can be accessed in unicode ranges. Now, a proper method must be invented for real unicode maths support. Before any code is written, I'm writing a specification in order to work out what is required. Fairly significant pieces of the NFSS may have to be re-written, and I'm a little unsure where to start.

2 Specification

This section will turn into 'User Interface' in time, presumably.

In the ideal case, a single unicode font will contain all maths glyphs we need. Barbara Beeton's STIX table provides the mapping between unicode maths glyphs and macro names (all 3298 — or however many — of them!). A single command

\setmathsfont[⟨*font features*⟩]{⟨*font name*⟩}

would implement this for every every symbol and alphabetic variant. That means `x` to x, `\xi` to ξ, `\leq` to $\leq$, etc., `\mathcal{H}` to $\mathcal{H}$ and so on, all for unicode glyphs within a single font.

Furthermore, this package should deal well with unicode characters for maths input, as well. This includes using literal Greek letters in formulae, resolving to upright or italic depending on preference.

Finally, maths versions must also be provided for. While I guess version selection in LaTeX will remain the same, the specification for choosing the version fonts will probably be an optional argument:

\setmathsfont[Version=Bold,⟨*font features*⟩]{⟨*font name*⟩}

All instances of 'maths' in command names will be aliased to 'math' for our American (or abbreviatory-minded) friends. Instances above of

[⟨*font features*⟩]{⟨*font name*⟩}

follow from my `fontspec` package, and therefore any additional ⟨*font features*⟩ specific to maths fonts will hook into `fontspec`'s methods.

2.1 Using multiple fonts

Let's face it; there will probably be few cases where a single unicode maths font suffices. The upcoming STIX font comes to mind as a notable exception. It will therefore be necessary to delegate specific unicode ranges of glyphs to separate fonts. This syntax will also hook into the `fontspec` font feature processing:

\setmathsfont[Range=⟨*unicode range*⟩,⟨*font features*⟩]{⟨*font name*⟩}

where ⟨*unicode range*⟩ is a comma-separated list of unicode slots and ranges such as `{27D0-27EB, 27FF, 295B-297F}`. Furthermore, preset names ranges could be used, such as `MiscMathSymbolsA`, with such ranges based on unicode chunks. The amount of optimisation required here to achieve acceptable performance has yet to be determined. Techniques such as saving out unicode subsets based on ⟨*unicode range*⟩ data to be `\input` in the next LaTeX run are a possibility, but at this stage, performance without such measures seems acceptable.

2.2 Script and scriptscript fonts/features

Cambria Math uses OpenType font features to activate smaller optical sizes for scriptsize and scriptscriptsize symbols (the B and C, respectively, in A_{B_C}.

Other fonts will no doubt use entirely separate fonts. Both of these options must be taken into account. I hope this will be mostly automatic from the users' points of view. The `+ssty` feature can be detected and applied automatically, and appropriate optical size information embedded in the fonts will ensure this latter case. Fine tuning should be possible automatically with `fontspec` options. We might have to wait until MnMath, for example, before we really know.

File I

The unicode-math package

This is the package.

```
\ProvidesPackage{unicode-math}
  [2006/02/20 v0.01 Unicode maths definitions]
```

3 Things we need

Packages

```
\RequirePackage{fontspec}
```

Counters and conditionals

```
\newcounter{um@fam}
\newif\if@um@fontspec@feature
```

Shortcuts

```
\newcommand\um@PackageError[2]{\PackageError{unicode-math}{#1}{#2}}
\newcommand\um@PackageWarning[1]{\PackageWarning{unicode-math}{#1}}
\newcommand\um@PackageInfo[1]{\PackageInfo{unicode-math}{#1}}
```

3.1 Programming macros

\um@Loop
\um@Break

See Kees van der Laan's various articles on TEX programming:

```
\def\um@Loop#1\um@Pool{#1\um@Loop#1\um@Pool}
\def\um@Break#1\um@Pool{}
```

\um@FOR

A simple 'for' loop implemented with the above. Takes a (predefined) counter \csname and increments it between two integers, iterating as we go.

```
\long\def\um@FOR #1 = [#2:#3] #4{%
  {\csname#1\endcsname =#2\relax
   \um@Loop #4%
     \expandafter\advance\csname#1\endcsname\@ne
     \expandafter\ifnum\csname#1\endcsname>#3\relax
       \expandafter\um@Break
     \fi
   \um@Pool}}
```

---

g/h/i/j/k/l/m/

```
\newcount\@ii
\um@FOR @ii = [7:13] {\@alph\@ii/}
```

3.2 Overcoming `\@onlypreamble`

TODO: This will be refined later! Sort out which macros actually have to be removed from the `\@preamblecmds` token list.

```
\def\@preamblecmds{}
```

4 Fundamentals

4.1 Enlarging the number of maths families

To start with, we've got a power of two as many `\fam`s as before. So (from `ltfssbas.dtx`) we want to redefine

```
\def\new@mathgroup{\alloc@8\mathgroup\chardef\@cclvi}
\let\newfam\new@mathgroup
```

Up to math fam 25 of 255.

```
\um@FOR @tempcnta = [1:20]
  {\expandafter\newfam
     \csname mt\@alph\@tempcnta\endcsname}
Up to math fam \the\mtt\ of 255.
```

This is sufficient for LaTeX's `\DeclareSymbolFont`-type commands to be able to define 256 named maths fonts. Now we need a new `\DeclareMathSymbol`.

4.2 `\DeclareMathSymbol` for unicode ranges

This is mostly an adaptation from LaTeX's definition.

`\DeclareUnicodeMathSymbol`

`#1` : Symbol, *e.g.*, `\alpha` or `a`
`#2` : Type, *e.g.*, `\mathalpha`
`#3` : Math font name, *e.g.*, `operators`
`#4` : Slot, *e.g.*, `"221E`

```
\def\DeclareUnicodeMathSymbol#1#2#3#4{%
```

First ensure the math font (*e.g.*, `operators`) exists:

```
  \expandafter\in@\csname sym#3\expandafter\endcsname
    \expandafter{\group@list}%
  \ifin@
```

No longer need here to perform the obfuscated hex conversion, since `\XeTeXmathchar` (and friends) has a more simplified input than TeX's `\mathchar`.

```
    \begingroup
```

The symbol to be defined can be either a command (`\alpha`) or a character (`a`). Branch for the former:

```
      \if\relax\noexpand#1% is command?
      \edef\reserved@a{\noexpand\in@{\string\XeTeXmathchar}{\meaning#1}}%
        \reserved@a
```

If the symbol command definition contains `\XeTeXmathchar`, then we can provide the info that a previous symbol definition is being overwritten:

```
        \ifin@
          \expandafter\um@set@mathsymbol
             \csname sym#3\endcsname#1#2{#4}%
          \@font@info{Redeclaring math symbol \string#1}%
```

Otherwise, overwrite it if the symbol command definition contains plain old `\mathchar`:

```
        \else
         %\edef\reserved@a{\noexpand\in@{\string\mathchar}{\meaning#1}}%
          %\reserved@a
          %\ifin@
          %  \expandafter\set@xmathsymbol
          %     \csname sym#3\endcsname#1#2{#4}%
```

Otherwise, throw an error if the command name is already taken by a non-symbol definition:

```
          %\else
            %\expandafter\ifx
            %\csname\expandafter\@gobble\string#1\endcsname
            %\relax
              \expandafter\um@set@mathsymbol
                 \csname sym#3\endcsname#1#2{#4}%
            %\else
            %  \@latex@error{Command `\string#1' already defined}\@eha
            %\fi
          %\fi
        \fi
```

And if the symbol input is a character:

```
      \else
        \expandafter\um@set@mathchar
          \csname sym#3\endcsname#1#2{#4}%
      \fi
    \endgroup
```

Everything previous was skipped if the maths font doesn't exist in the first place:

```
  \else
    \@latex@error{Symbol font `#3' is not defined}\@eha
  \fi}
```

The final macros that actually define the maths symbol with XeTeX primitives.

`\um@set@mathsymbol`

`#1` : Symbol font number
`#2` : Symbol macro, *e.g.*, `\alpha`
`#3` : Type, *e.g.*, `\mathalpha`
`#4` : Slot, *e.g.*, `"221E`
If the symbol definition is for a macro. There are a bunch of tests to perform to process the various characters.

```
\def\um@set@mathsymbol#1#2#3#4{%
  \iftrue%\unless\ifx#3\mathalpha
```

Operators First test if the character requires a `\nolimits` suffix. This is controlled by the `\um@nolimits` macro, which contains a commalist of such characters. If so, define the mathchar \⟨*cs*⟩`op` (where `#2` is \⟨*cs*⟩) and define \⟨*cs*⟩ as the wrapper around this control sequence.

```
\expandafter\in@\expandafter#2\expandafter{\um@nolimits}%
\ifin@
  \expandafter\global\expandafter\XeTeXmathchardef
    \csname\expandafter\@gobble\string#2 op\endcsname
    ="\mathchar@type#3 #1 #4\relax
  \gdef#2{\csname\expandafter\@gobble\string#2 op\endcsname\nolimits}%
\else
```

Radicals

```
  \expandafter\in@\expandafter#2\expandafter{\um@radicals,}%
  \ifin@
    \gdef#2{\XeTeXradical#1 #4\relax}%
  \else
```

Delimiters

```
    \ifx\mathopen#3\relax
      \gdef#2{\XeTeXdelimiter "\mathchar@type#3 #1 #4}%
      \global\XeTeXdelcode#4=#1 #4\relax
    \else
      \ifx\mathclose#3\relax
        \gdef#2{\XeTeXdelimiter "\mathchar@type#3 #1 #4}%
        \global\XeTeXdelcode#4=#1 #4\relax
      \else
```

And finally, the general case. We define both the macro and the unicode mathcode; this only works for 16-bit unicode scalar values, however. TODO: make all higher plane maths characters math-active so that spacing works for literal unicode input.

```
        \global\XeTeXmathchardef#2="\mathchar@type#3 #1 #4\relax
        \ifnum#4<"FFFF
          \global\XeTeXmathcode#4="\mathchar@type#3 #1 #4\relax
        \fi
      \fi
    \fi
  \fi
\fi}
```

`\um@set@mathchar`

`#1` : Symbol font number
`#2` : Symbol, *e.g.*, `\alpha` or `a`
`#3` : Type, *e.g.*, `\mathalpha`
`#4` : Slot, *e.g.*, `"221E`
Or if it's for a character:

```
\def\um@set@mathchar#1#2#3#4{%
  \global\XeTeXmathcode`#2="\mathchar@type#3 #1 #4\relax}
```

```
\zf@fontspec{}{Cambria Math}
\let\glb@currsize\relax
\DeclareSymbolFont{test}{EU1}{\zf@family}{m}{n}
\DeclareUnicodeMathSymbol{\infinity}{\mathord}{test}{"221E}
$\infinity$
```

\DeclareUnicodeMathCode [For later] or if it's for a character code: (just a wrapper around the primitive)

```
\def\DeclareUnicodeMathCode#1#2#3#4{%
  \global\XeTeXmathcode#1=
    "\mathchar@type#2 \csname sym#3\endcsname #4\relax}
```

```
\zf@fontspec{}{Cambria Math}
\let\glb@currsize\relax
\DeclareSymbolFont{test2}{EU1}{\zf@family}{m}{n}
\DeclareUnicodeMathCode{65}{\mathalpha}{test2}{119860}
$A$
```

4.3 User interface to \DeclareSymbolFont

\setmathfont [#1]: font features
#2 : font name

```
\newcommand\setmathfont[2][]{%
%  \begingroup
```

Init

- Erase any conception LaTeX has of previously defined math symbol fonts; this allows \DeclareSymbolFont at any point in the document.
- To start with, assume we're defining every math symbol character.
- Bump up the um@fam counter to assign a new maths symbol font.
- Tell fontspec that maths font features are actually allowed.
- Grab the current size information (is this robust enough? Maybe it should be preceded by \normalsize…).

```
  \let\glb@currsize\relax
  \let\um@char@range\@empty
  \stepcounter{um@fam}%
  \@um@fontspec@featuretrue
  \csname S@\f@size\endcsname
```

Now when the list of unicode symbols is input, we want a suitable definition of its internal macro. By default, we want to define every single math char.

Use fontspec to select a font to use. The macro \S@⟨*size*⟩ contains the definitions of the sizes used for maths letters, subscripts and subsubscripts in \tf@size, \sf@size, and \ssf@size, respectively.

Probably in the future we want options to change the hard-coded fontspec maths-related features.

```
    \zf@fontspec{
      Mapping=math-italic,
      Script=Math, SizeFeatures={
        {Size=\tf@size-},
        {Size=\sf@size-\tf@size,ScriptStyle={}},
        {Size=-\sf@size,ScriptScriptStyle={}}},
      #1}{#2}%
```

Probably want to check there that we're not creating multiple symbol fonts with the same NFSS declaration. On that note, fontspec doesn't seem to be keeping track of that, either :((check that out!)

```
    \DeclareSymbolFont{um@fam\theum@fam}
      {\encodingdefault}{\zf@family}{\mddefault}{\updefault}%
  \ifx\um@char@range\@empty
```

Try other alphabets:

```
    \zf@fontspec{
      Script=Math, SizeFeatures={
        {Size=-\sf@size,ScriptScriptStyle={}},
        {Size=\sf@size-\tf@size,ScriptStyle={}},
        {Size=\tf@size-}},
      #1}{#2}%
   \SetMathAlphabet\mathrm{normal}{\encodingdefault}{\zf@family}{\mddefault}{\updefault}%
    \zf@fontspec{
      Mapping=math-bold,
      Script=Math, SizeFeatures={
        {Size=-\sf@size,ScriptScriptStyle={}},
        {Size=\sf@size-\tf@size,ScriptStyle={}},
        {Size=\tf@size-}},
      #1}{#2}%
   \SetMathAlphabet\mathbf{normal}{\encodingdefault}{\zf@family}{\mddefault}{\updefault}%
    \um@text@input{um@fam\theum@fam}%
    \um@PackageWarning{Defining the default maths font as `#2'}%
    \def\UnicodeMathSymbol##1##2##3##4{%
      \DeclareUnicodeMathSymbol
        {##2}{##3}{um@fam\theum@fam}{##1}}%
  \else
```

If the Range font feature has been used, then only a subset of the unicode glyphs are to be defined. See section **??** for the code that enables this.

```
    \def\UnicodeMathSymbol##1##2##3##4{%
      \um@parse@term{##1}{##2}{##3}{%
     \um@PackageWarning{Defining \string##2 as mathchar ##1 from font `#2'}%
        \DeclareUnicodeMathSymbol
          {##2}{##3}{um@fam\theum@fam}{##1}}}%
  \fi
```

And now we input every single maths char. See File **??** for the source to unicode-math.tex.

```
  \input uniadd.tex
```