Integrating Unicode and OpenType math in ConTEXt

Aditya Mahajan

Abstract

In this article, I briefly explain the approach taken by ConTEXt to integrate Unicode math with OpenType fonts.

1 A bit of history

Since around 2000, ConTEXt has supported Unicode math input. Under the `utf-8` input regime (obtained with `\enableregime[utf-8]`), you could type `$αβγ$` to get the Greek letters $\alpha\beta\gamma$. This was achieved by lot of macro jugglery; similar to the kind of macro jugglery needed for accents to work without having to type in the arcane plain TEX accent macros. Basically, the input regime ensured that inside math mode `α` got mapped to `\alpha`, and the rest was taken care of by the usual font mappings that mapped `\alpha` to the correct glyph in the correct font. However, these mappings were not clean. If you defined a macro, say `\MACRO`, that took one argument, then `\MACRO α` would not work; you had to type `\MACRO{α}`. This small detail of grouping UTF-8 input was a constant reminder that things were not so clean underneath.

LuaTEX was designed to handle input encoding cleanly. The engine only understands UTF-8 encoding and provides enough hooks to implement other encodings. ConTEXt MkIV assumes that the input is either UTF-8 or UTF-16. The input can then be directly mapped to the correct glyph locations in TrueType and OpenType fonts. However, handling of math input was much trickier, mainly because of the effort needed to support OpenType math. So, for some time, the handling of math fonts did not change: in math mode `α` was mapped to `\alpha` and traditional TEX font mapping ensured that `\alpha` was mapped to the correct glyph in the correct font.

Around the beginning of this year, ConTEXt MkIV completely moved to Unicode math. Thus, `0x1D6FC` (math italic small alpha) was mapped to the same position in an OpenType math font. If you type this Unicode character in math mode, the output is α. For convenience, typing `0x03B1` (Greek small letter alpha) in math mode gets mapped to `0x1D6FC`; as does the macro `\alpha`. All this is transparent to the user, except when he accidentally types `$\MACRO α$` and is pleasantly surprised not to get a nasty error message.

There are two steps involved in the integration of Unicode input with OpenType math fonts: (i) map the input characters or macros to the correct Unicode character; (ii) map the Unicode character to the correct OpenType glyph at the correct size and with the correct kerning. The bulk of these mappings are done by five files: `char-def.lua`, `math-ini.lua`, `math-map.lua`, `math-noa.lua` and `math-vfu.lua`. Below I explain briefly what these files do. Please take everything in this article with a pinch of salt. I do not understand how OpenType math fonts work, so specific details may be wrong. The main idea of this article is to convey the gist of how things are done in ConTEXt, and where to search if you want to know specific details.

2 char-def.lua

Mapping UTF-8 input chapters to Unicode characters is straightforward — the byte sequence of the input character is the same as the Unicode slot. Thus, input byte sequence `0x1D6FC` is the Unicode character `0x1D6FC`. However, mapping macros to Unicode byte sequences is different. We need to explicitly tell ConTEXt that `\alpha` corresponds to the Unicode character `0x1D6FC`. Furthermore, just getting the correct glyph is not sufficient for typesetting mathematics. We also need to know the math class of the glyph, so that TEX can correctly position the characters.

All this information is stored as a Lua table in `char-def.lua`. This is a huge file, initially generated from Unicode tables and later updated manually from data present in ConTEXt files and elsewhere (it is still not complete). For example, the entry for `0x1D6FC` is:

```
[0x1D6FC]={
 category="ll",
 description="MATHEMATICAL ITALIC SMALL ALPHA",
 direction="l",
 linebreak="al",
 mathclass="default",
 mathname="alpha",
 specials={ "font", 0x03B1 },
 unicodeslot=0x1D6FC,
},
```

Amongst other things, this tells ConTEXt that the macro `\alpha` (indicated by `mathname="alpha"`) corresponds to this Unicode slot. It also tells that the math class of this character is `default`. Similar details are provided for a large fraction of the Unicode characters.

Some Unicode characters correspond to more than one macro (with different math classes). One such example is `0x007C`, which corresponds to `\arrowvert`, `\vert`, `\lvert`, `\rvert`, and `\mid`, each belonging to a different math class. This Uni-

code character is represented as:

```
{
 adobename="bar",
 category="sm",
 cjkwd="na",
 contextname="textbar",
 description="VERTICAL LINE",
 direction="on",
 linebreak="ba",
 mathspec={
  { class="nothing",   name="arrowvert" },
  { class="delimiter", name="vert"      },
  { class="open",      name="lvert"     },
  { class="close",     name="rvert"     },
  { class="relation",  name="mid"       },
 },
 unicodeslot=0x007C,
},
```

The different macros and their corresponding math classes are encoded as part of a `mathspec` key. When the character `|` (`0x007C`) is typed the math class is set to the class of the first element of the `mathspec` table (`nothing` in this case).

3 math-ini.lua

All the information in the Lua table in `char-def.lua` by itself is useless. We need to tell ConTEXt how to use it. For the math mappings, this is done in `math-ini.lua`.

This file begins by defining names for the different math classes:

```
local classes = {
    ord       =  0,  -- mathordcomm
    op        =  1,  -- mathopcomm
    bin       =  2,  -- mathbincomm
    rel       =  3,  -- mathrelcomm
    open      =  4,  -- mathopencomm
    close     =  5,  -- mathclosecomm
    punct     =  6,  -- mathpunctcomm
    alpha     =  7,  -- mathalphacomm
    accent    =  8,  -- class 0
    radical   =  9,
    xaccent   = 10,  -- class 3
    topaccent = 11,  -- class 0
    botaccent = 12,  -- class 0
    under     = 13,
    over      = 14,
    delimiter = 15,
    inner     =  0,  -- mathinnercomm
    nothing   =  0,  -- mathnothingcomm
    choice    =  0,  -- mathchoicecomm
    box       =  0,  -- mathboxcomm
    limop     =  1,  -- mathlimopcomm
    nolop     =  1,  -- mathnolopcomm
}
```

For each math class, this file has functions to return the LuaTEX code to define the corresponding math characters. A few examples of such functions:

```
local function delcode(target,family,slot)
    return format('\\Udelcode%s="%X "%X ',
                  target,family,slot)
end
local function mathchar(class,family,slot)
    return format('\\Umathchar "%X "%X "%X ',
                   class,family,slot)
end
local function mathaccent(class,family,slot)
    return format('\\Umathaccent "%X "%X "%X ',
                  0,family,slot)
end
```

Similar functions are there for defining math symbols, for example:

```
function setmathsymbol(name,class,family,slot)
  if class == classes.accent then
    texsprint(
        format("\\unexpanded\\xdef\\%s{%s}",
        name,
        mathaccent(class,family,slot)))
  elseif class == classes.topaccent then
    texsprint(
        format("\\unexpanded\\xdef\\%s{%s}",
        name,
        mathtopaccent(class,family,slot)))
  elseif class == classes.botaccent then
    texsprint(
        format("\\unexpanded\\xdef\\%s{%s}",
        name,
        mathbotaccent(class,family,slot)))
  ...
end
```

`math-ini.lua` then defines a Lua function named `mathematics.define` that goes through all the elements in the table in `char-def.lua` and maps them to the correct LuaTEX command.

These mappings are all that is needed to work with OpenType math fonts like Cambria and Asana Math. ConTEXt has predefined typescripts for Cambria; so, to use Cambria you can just type

```
\usetypescript[cambria]
\setupbodyfont[cambria]
```

There are no predefined typescripts for Asana Math, but defining one on our own is not too hard. First, we need to define a math typescript:

```
\starttypescript [math] [asana] [name]
  \definefontsynonym
    [MathRoman]
    [name:Asana-Math]
    [features=math\mathsizesuffix]
\stoptypescript
```

The `features=math\mathsizesuffix` option activates the OpenType math features. The rest of the typescript can be defined in the usual manner.

```
\starttypescript [asana]
  \definetypeface [asana] [rm] [serif]
      [palatino] [default]
  \definetypeface [asana] [ss] [sans]
      [modern]   [default] [rscale=1.075]
  \definetypeface [asana] [tt] [mono]
      [modern]   [default] [rscale=1.075]
  \definetypeface [asana] [mm] [math]
      [asana]    [default]
  \quittypescriptscanning
\stoptypescript
```

To use this typescript, we need to type

```
\usetypescript[asana]
\setupbodyfont[asana]
```

4 math-map.lua and math-noa.lua

So far, we have mapped Unicode input and macros to OpenType math fonts. However, when using TEX one expects traditional TEX input to work. Thus, `$a$` should typeset math italic *a*. No one is likely to type `0x1D44E`, even if Unicode suggests that. The same is true for bold fonts. One expects `${\bfa}$` to give bold **a**, even if Unicode suggests that we should have typed `0x1D41A`.

To accommodate this, in math mode ConTEXt maps upper and lower case letters `A-Z`, `a-z`, digits `0-9`, and upper and lower case Greek letters `α-ω`, `A-Ω` to the corresponding ranges in Unicode math, depending on the current font style. These mappings are defined in `math-map.lua` file.

The mappings are defined using a Lua table, which looks like this.

```
mathematics.alphabets = {
    regular = {
              tf = { ... },
              it = { ... },
              bi = { ... },
              bf = { ... },
    },
    sansserif = {
              tf = { ... },
              it = { ... },
              bi = { ... },
              bf = { ... },
    },
    monospaced = {
              tf = { ... },
    },
    blackboard = {
              tf = { ... },
    },
    fraktur = {
              tf = { ... },
              bf = { ... },
    },
    script  = {
              tf = { ... },
              bf = { ... },
    }
}
```

Each of these subtables maps input letters to their corresponding Unicode characters. These subtables look as follows.

```
regular = {
    ...
    it = {
      ucletters = 0x1D434,
      lcletters = { -- H
        [0x00061]=0x1D44E, [0x00062]=0x1D44F,
        [0x00063]=0x1D450, [0x00064]=0x1D451,
        [0x00065]=0x1D452, [0x00066]=0x1D453,
        [0x00067]=0x1D454, [0x00068]=0x0210E,
        ... },
     symbols = {
        [0x0391]=0x1D6E2, [0x0392]=0x1D6E3,
        [0x0393]=0x1D6E4, [0x0394]=0x1D6E5,
        [0x0395]=0x1D6E6, [0x0396]=0x1D6E7,
        ... },
    },
},
```

The line `ucletters = 0x1D434` tells ConTEXt to map upper case letters to Unicode characters starting from `0x1D434`. The line `lcletters = {...}` tells ConTEXt to map `0x00061` to `0x1D44E`, `0x00062` to `0x1D44F`, as so on. For lower case letters, simply specifying `lcletters = 0x1D44E` would not work because Unicode mathematical italic small letters are not in contiguous slots. For example, the slot `0x1D455` (which corresponds to lower case h) is empty; lower case h should map to slot `0x0210E`.

Other subtables are filled in a similar manner.

`math-map.lua` also defines Lua functions that use these tables to remap characters on the fly. The actual transformation takes place in `math-noa.lua` which goes through the math noad list and carries out the actual transformations according to the mappings in `math-map.lua`.

5 math-vfu.lua

Using the above infrastructure, it is easy to use OpenType math fonts in ConTEXt. Unfortunately, at present there are only two Unicode math fonts — Cambria and Asana Math. OpenType math version of TEX Gyre math fonts are planned, but until they are developed, we need a way to use traditional TEX fonts in ConTEXt MkIV. ConTEXt creates virtual OpenType math fonts to use traditional TEX fonts. The mappings for creating the virtual font are in `math-vfu.lua`. Once a virtual OpenType math font is created, the above infrastructure can be used to access the font.

First, `math-vfu.lua` defines many encoding

vectors that map Unicode characters to glyph locations of the font. One such encoding vector is

```
fonts.enc.math["tex-mi"] = {
    [0x1D6E4] = 0x00, -- Gamma
    [0x1D6E5] = 0x01, -- Delta
    [0x1D6E9] = 0x02, -- Theta
    [0x1D6F3] = 0x02, -- varTheta (not in TeX)
    [0x1D6EC] = 0x03, -- Lambda
    [0x1D6EF] = 0x04, -- Xi
    [0x1D6F1] = 0x05, -- Pi
    [0x1D6F4] = 0x06, -- Sigma
    ...
}
```

This tells that Unicode character `0x1D6E4` should be mapped to the font glyph `0x00` and so on. A virtual font that associates such encoding vectors with traditional TEX fonts is created using

```
mathematics.make_font ( "lmroman10-math", {
  { name="lmroman10-regular.otf",
    features="virtualmath", main=true },
  { name="lmmi10.tfm", vector="tex-mi",
    skewchar=0x7F },
  { name="lmsy10.tfm", vector="tex-sy",
    skewchar=0x30, parameters=true },
  { name="lmex10.tfm", vector="tex-ex",
    extension=true },
  { name="msam10.tfm", vector="tex-ma" },
  { name="msbm10.tfm", vector="tex-mb" },
  { name="lmroman10-bold.otf", "tex-bf" } ,
  { name="lmmib10.tfm", vector="tex-bi",
    skewchar=0x7F },
  { name="lmsans10-regular.otf",
    vector="tex-ss", optional=true },
  { name="lmmono10-regular.otf",
    vector="tex-tt", optional=true },
  { name="eufm10.tfm", vector="tex-fraktur",
    optional=true },
  { name="eufb10.tfm",
    vector="tex-fraktur-bold", optional=true },
} )
```

This creates a virtual font `lmroman10-math` which takes bits and pieces from various fonts. This virtual font can be used as follows.

```
\starttypescript [math] [modern]
  ...
  \definefontsynonym
    [LMMathRoman10-Regular]
    [LMMath10-Regular@lmroman10-math]
  ...
\stoptypescript
```

The `@lmroman-math` in the name uses the above virtual font. The `LMMathRoman10-Regular` font can be used to complete the math typescript in the usual manner.

ConTEXt provides virtual OpenType math fonts for Latin Modern, Times (`txfonts` and MathTime), Palatino (`pxfonts`), Iwona, Lucida, and MathDesign (Charter, Garamond, and Utopia) fonts.

6 Conclusion

ConTEXt now supports OpenType math fonts. In fact, even support for traditional TEX fonts now involves creating a virtual OpenType math font. Thus, as far as ConTEXt MkIV is concerned, OpenType math fonts are the future. However, the current implementation is still evolving, so some of the implementation details described in this paper will likely change with time.

◇ Aditya Mahajan
adityam (at) umich dot edu