Tricky fences

Hans Hagen

Occasionally one of my colleagues notices some suboptimal rendering and asks me to have a look at it. Now, one can argue about "what is right" and indeed there is not always a best answer to it. Such questions can even be a nuisance; let's think of the following scenario. You have a project where TEX is practically the only solution. Let it be an XML rendering project, which means that there are some boundary conditions. Speaking in 2017 we find that in most cases a project starts out with the assumption that everything is possible.

Often such a project starts with a folio in mind and therefore by decent tagging to match the educational and esthetic design. When rendering is mostly automatic and concerns too many (variants) to check all rendering, some safeguards are used (an example will be given below). Then different authors, editors and designers come into play and their expectations, also about what is best, often conflict. Add to that rendering for the web, and devices and additional limitations show up: features get dropped and even more cases need to be compensated (the quality rules for paper are often much higher). But, all that defeats the earlier attempts to do well because suddenly it has to match the lesser format. This in turn makes investing in improving rendering very inefficient (read: a bottomless pit because it never gets paid and there is no way to gain back the investment). Quite often it is spacing that triggers discussions and questions what rendering is best. And inconsistency dominates these questions.

So, in case you wonder why I bother with subtle aspects of rendering as discussed below, the answer is that it is not so much professional demand but users (like my colleagues or those on the mailing lists) that make me look into it and often something that looks trivial takes days to sort out (even for someone who knows his way around the macro language, fonts and the inner working of the engine). And one can be sure that more cases will pop up.

All this being said, let's move on to a recent example. In ConTEXt we support MathML although in practice we're forced to a mix of that standard and ASCIIMATH. When we're lucky, we even get a mix with good old TEX-encoded math. One problem with an automated flow and processing (other than raw TEX) is that one can get anything and therefore we need to play safe. This means for instance that you can get input like this:

```
f(x) + f(1/x)
```

or in more structured TEX speak:

```
$f(x) + f(\frac{1}{x})$
```

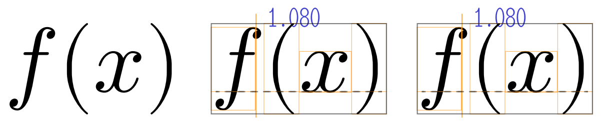
Using TEX Gyre Pagella, this renders as: $f(x) + f(\frac{1}{x})$, and when seeing this a TEX user will revert to:

```
$f(x) + f\left(\frac{1}{x}\right)$
```

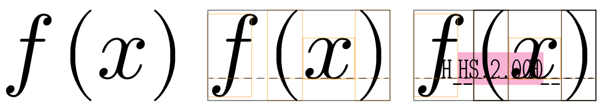
which gives: $f(x) + f\left(\frac{1}{x}\right)$. So, in order to be robust we can always use the `\left` and `\right` commands, can't we?

```
$f(x) + f\left(x\right)$
```

which indeed gives $f(x) + f(x)$, but let's blow up this result a bit showing some additional tracing from left to right, now in Latin Modern:



just characters just characters just characters



using delimiters using delimiters using delimiters

When we visualize the glyphs and kerns we see that there's a space instead of a kern when we use delimiters. This is because the delimited sequence is processed as a subformula and injected as a so-called inner object and as such gets spaced according to the ordinal (for the f) and inner ("fenced" with delimiters x) spacing rules. Such a difference normally will go unnoticed but as we mentioned authors, editors and designers being involved, there's a good chance that at some point one will magnify a PDF preview and suddenly notice that the difference between the f and (is a bit on the large side for simple unstacked cases, something that in print is likely to go unnoticed. So, even when we don't know how to solve this, we do need to have an answer ready.

When I was confronted by this example of rendering I started wondering if there was a way out. It makes no sense to hard code a negative space before a fenced subformula because sometimes you don't want that, especially not when there's nothing before it. So, after some messing around I decided to have a look at the engine instead. I wondered if we could just give the non-scaled fence case the same treatment as the character sequence.

Unfortunately here we run into the somewhat complex way the rendering takes place. Keep in mind that it is quite natural from the perspective of TEX because normally a user will explicitly use `\left` and `\right` as needed, while in our case the fact that we automate and therefore want a generic solution interferes (as usual in such cases).

Once read in the sequence `f(x)` can be represented as a list:

```
list = {
 {
  id = "noad", subtype = "ord", nucleus = {
   {
    id = "mathchar", fam = 0, char = "U+00066",
   },
  },
 },
 {
  id = "noad", subtype = "open", nucleus = {
   {
    id = "mathchar", fam = 0, char = "U+00028",
   },
  },
 },
 {
  id = "noad", subtype = "ord", nucleus = {
   {
    id = "mathchar", fam = 0, char = "U+00078",
   },
  },
 },
 {
  id = "noad", subtype = "close", nucleus = {
   {
    id = "mathchar", fam = 0, char = "U+00029",
   },
  },
 },
}
```

The sequence `f \left( x \right)` is also a list but now it is a tree (we leave out some unset keys):

```
list = {
 {
  id = "noad", subtype = "ord", nucleus = {
   { id = "mathchar", fam = 0,
     char = "U+00066",},
  },
 },
 {
  id = "noad", subtype = "inner", nucleus = {
   {
    id = "submlist", head = {
     {
      id = "fence", subtype = "left",
       delim = { { id = "delim", small_fam = 0,
                   small_char = "U+00028", },
      },
     },
     {
      id = "noad", subtype = "ord",
       nucleus = { { id = "mathchar", fam = 0,
                     char = "U+00078", },
      },
     },
     {
      id = "fence", subtype = "right",
      delim = { { id = "delim", small_fam = 0,
                  small_char = "U+00029", },
      },
     },
    },
   },
  },
 },
}
```

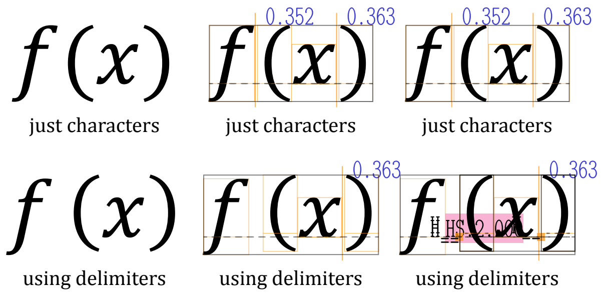
So, the formula `f(x)` is just four characters and stays that way, but with some inter-character spacing applied according to the rules of TEX math. The sequence `f \left( x \right)` however becomes two components: the `f` is an ordinal noad,[1] and `\left( x \right)` becomes an inner noad with a list as a nucleus, which gets processed independently. The way the code is written this is what (roughly) happens:

- A formula starts; normally this is triggered by one or two dollar signs.
- The `f` becomes an ordinal noad and TEX goes on.
- A fence is seen with a left delimiter and an inner noad is injected.
- That noad has a sub-math list that takes the left delimiter up to a matching right one.
- When all is scanned a routine is called that turns a list of math noads into a list of nodes.
- So, we start at the beginning, the ordinal `f`.
- Before moving on a check happens if this character needs to be kerned with another (but here we have an ordinal–inner combination).
- Then we encounter the subformula (including fences) which triggers a nested call to the math typesetter.
- The result eventually gets packaged into a hlist and we're back one level up (here after the ordinal `f`).
- Processing a list happens in two passes and, to cut it short, it's the second pass that deals with choosing fences and spacing.
- Each time when a (sub)list is processed a second pass over that list happens.
- So, now TEX will inject the right spaces between pairs of noads.
- In our case that is between an ordinal and an inner noad, which is quite different from a sequence of ordinals.

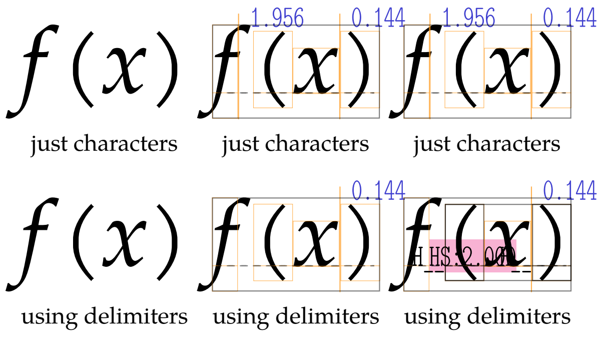
[1] Noads are the mathematical building blocks. Eventually they become nodes, the building blocks of paragraphs and boxed material.

It's these fences that demand a two-pass approach because we need to know the height and depth of the subformula. Anyway, do you see the complication? In our inner formula the fences are not scaled, but this is not communicated back in the sense that the inner noad can become an ordinal one, as in the simple `f(` pair. The information is not only lost, it is not even considered useful and the only way to somehow bubble it up in the processing so that it can be used in the spacing requires an extension. And even then we have a problem: the kerning that we see between `f(` is also lost. It must be noted that this kerning is optional and triggered by setting `\mathitalicsmode=1`. One reason for this is that fonts approach italic correction differently, and cheat with the combination of natural width and italic correction.

Now, because such a workaround is definitely conflicting with the inner workings of TEX, our experimenting demands another variable be created: `\mathdelimitersmode`. It might be a prelude to more manipulations but for now we stick to this one case. How messy it really is can be demonstrated when we render our example with Cambria.



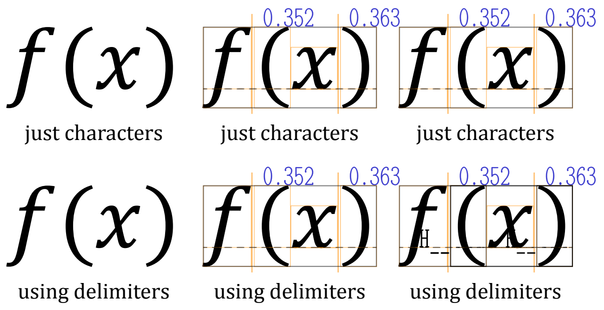
If you look closely you will notice that the parenthesis are moved up a bit. Also notice the more accurate bounding boxes. Just to be sure we also show Pagella:



When we really want the unscaled variant to be somewhat compatible with the fenced one we now need to take into account:

- the optional axis-and-height/depth related shift of the fence (bit 1)
- the optional kern between characters (bit 2)
- the optional space between math objects (bit 4)

Each option can be set (which is handy for testing) but here we will set them all, so, when `\mathdelimitersmode=7`, we want cambria to come out as follows:



When this mode is set the following happens:

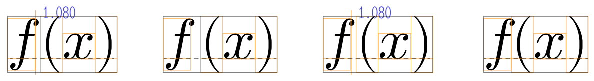
- We keep track of the scaling and when we use the normal size this is registered in the noad (we had space in the data structure for that).
- This information is picked up by the caller of the routine that does the subformula and stored in the (parent) inner noad (again, we had space for that).
- Kerns between a character (ordinal) and subformula (inner) are kept, which can be bad for other cases but probably less than what we try to solve here.
- When the fences are unscaled the inner property temporarily becomes an ordinal one when we apply the inter-noad spacing.

Hopefully this is good enough but anything more fancy would demand drastic changes in one of the most sensitive mechanisms of TEX. It might not always work out right, so for now I consider it an experiment, which means that it can be kept around, rejected or improved.

In case one wonders if such an extension is truly needed, one should also take into account that automated typesetting (also of math) is probably one of the areas where TEX can shine for a while. And while we can deal with much by using Lua, this is one of the cases where the interwoven and integrated parsing, converting and rendering of the math machinery makes it hard. It also fits into a further opening up of the inner working by modes.

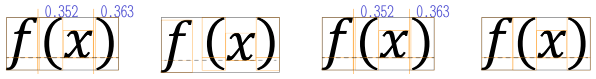
Another objection to such a solution can be that we should not alter the engine too much. However, fences already are an exception and treated specially (tests and jumps in the program) so adding this fits reasonably well into that part of the design.

In the following examples we demonstrate the results for Latin Modern, Cambria and Pagella when `\mathdelimitersmode` is set to zero or one. First we show the case where `\mathitalicsmode` is disabled:



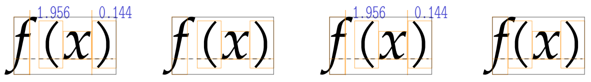
`\mathdelimitersmode =0` `\mathdelimitersmode =7`

modern



`\mathdelimitersmode =0` `\mathdelimitersmode =7`

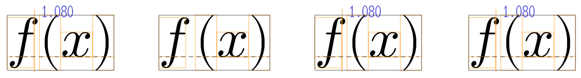
cambria



`\mathdelimitersmode =0` `\mathdelimitersmode =7`

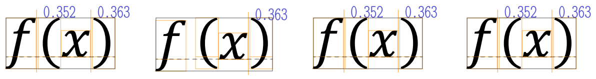
pagella

When we enable `\mathitalicsmode` we get:



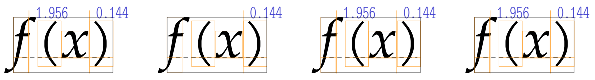
`\mathdelimitersmode =0` `\mathdelimitersmode =7`

modern



`\mathdelimitersmode =0` `\mathdelimitersmode =7`

cambria



`\mathdelimitersmode =0` `\mathdelimitersmode =7`

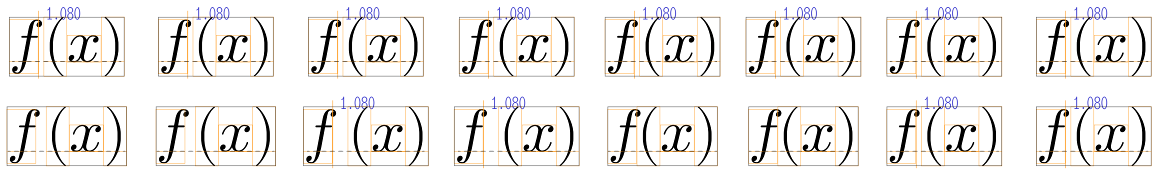
pagella

So is this all worth the effort? I don't know, but at least I got the picture and hopefully now you have too. It might also lead to some more modes in future versions of LuaTEX.

In ConTEXt, a regular document can specify `\setupmathfences [method=auto]`, but in MathML or ASCIIMATH this feature is enabled by default (so that we can test it).

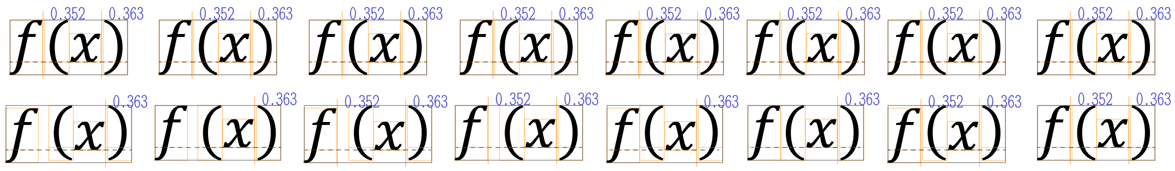
We end with a summary of all the modes (assuming italics mode is enabled) in the table below.

◇ Hans Hagen
Pragma ADE
http://pragma-ade.com



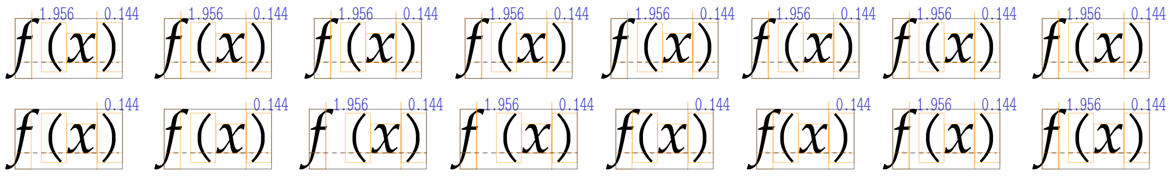
`ns` `it` `ns it` `or` `ns or` `it or` `ns it or`

modern



`ns` `it` `ns it` `or` `ns or` `it or` `ns it or`

cambria



`ns` `it` `ns it` `or` `ns or` `it or` `ns it or`

pagella