drawing the line

Contents

Introduction

When looking at possible new features or demands related to developments elsewhere we have to make decisions. Extensions or adaptations can be trivial and it can even be that we already have something but done differently, for instance because we were early adopters. However, when we're talking about features that are only of little use, or that reflect the current fashion we don't want to bloat the core and even if we outsource to modules the question is what the maintenance burden is. Especially when demands relate to the World Wide Web one can also wonder how long term valid something is. We've seen possibilities come and go and/or change over time.

Here we will collect some of the considerations for either or not adding support for something. It is of course subjective but better wrap it up than later trying to recall why something was not done. There have been plenty possible features that were rejected without wrapping up the experiments.

We might sound criticizing at times but that is mostly because we want to draw conclusions that explain why at the time of writing we expect something to be a bug or shortcoming and better wait for it to become better.

Hans Hagen
Mikael Sundqvist
....

1 Tagged PDF mapping

todo

2 MathML

This is a preprint for the 2024 ConTEXt meeting. We hope to have some discussion about where to go with tagging pdf and export to html. This is part of the upcoming "Drawing lines" document.

Mikael Sundqvist & Hans Hagen

Introduction

After we (mostly) finished upgrading math in ConTEXt and LuaMetaTEX, we sat down to look at tagging and exporting. Tagging (in pdf) needed some attention because it looked like there was some activity, and because MathML was (re)introduced in browsers we also had to check that. Both had to be done in the perspective of reimplementing some mechanism. We're not talking of drastic changes in low level tagging and export code here. But it could be nice if some hacks could now be avoided, and maybe better choices can be made. Below is not a complete wrapup, it just highlights some aspects that caught our attention. It also shows why we are not going to bother much about details in MathML: it is just too unreliable and unpredictable so the more we tweak the worse it can get. We draw a line.

Namespaces

Standards are not really standards when they are not downward compatible. Dealing with math is an example: MathML support comes and goes in browsers. Right from the start ConTEXt handled presentation and content MathML, later also OpenMath, which was not really a success and kind of morphed into MathML. Due to dropped or absent native support in browsers for MathML, the JavaScript driven MathJax could come to rescue.

Mid 2024 MathML has gotten more complex on the one hand and got simplified on the other by going 'core'. The `<math>` element is now a top level html element without the need for a namespace and therefore per 2024 we are advised not to use a namespace prefix because browsers and tools can have issues with it, just like they seem to have issues with `mfenced` that disappeared. It means that in a document with many formulas one gets larger files by delegating the namespace to the element itself, which doesn't hurt html and is still needed for xhtml, which is what we prefer. This change is likely to break older browsers but one can wonder how many old ones are still around. Alas, Amaya is no longer maintained as reference.

Accessibility

Unfortunately we cannot put the 'meaning' on a math element and given the constant transition of math it is hard to predict if it ever will be possible. There is no `<annotation>` available for spoken math and the standard is bit puzzling about the location of the main MathML blob in the `<semantics>` wrapper: supposedly it has to come first but there is no rule so we play safe in the related css file. So, for now we draw a line here: we set what is possible even if it's not used so that one can patch the result once an formal attribute or feature becomes available.

Fonts

There has been a gradual evolution with respect to fonts in html. There is a model that includes `serif`, `sans-serif`, `monospaced` on one axis and `normal` and `italic` on another plus `bold` on yet another. Then there is the so called font family and that one can be bound to font names (using operating system features) or via css lookups to a set of filenames, optionally with properties like weight and width, although that cannot be used reliable. We realize that it is not trivial to evolve and also handle the old methods.

In order to see if we could get various math fonts working with the `<math>` top level element, we tried several approaches but ran into problems when using the same file name several times. There are possibilities to have a main family with style and weight fields set but as of today we didn't get reliable `Serif` plus `bold` etc working so we had to settle for `SerifBold`. Because we have an independent setup this works okay. So, we now predefine `Serif*`, `Sans*`, `Mono*` and `Math*` sets as well as `Text*` aliases for the main text font. Users can define a proper body font (likely related to the real one) that suits the html representation. We also assume fonts to be part of the package.

It must be noted that contrary to ConTEXt producing pdf, we cannot get the same quality in html. This because we can't tweak the fonts, can't use the additional constructs, have little influence on the spacing model, and have to map high level structure onto low level presentation. Fonts are an important factor but not the only one.

We might pass/set some more in the future when it comes to font features but that also depends a bit on how css fonts evolve. There is plenty that can be supported, if only because we have much at the TEX end already, but much makes no sense in the kind of documents in play. Again we have to draw a line.

With respect to math fonts: we set them up as good as possible. On MS Windows often Cambria is the default, but in Linux it depends. When we were testing, some browsers used Noto Sans with serif regular shapes, although it looks like there will be a switch to sans shapes instead, so it is kind of unpredictable what one gets. We also noticed Stix kicking in. Cambria, which is also the reference font, is a reasonable default but instead of Noto[1] one can better configure the browser to default for instance to Stix if a match for a Times like serif is wished for.

But . . . even better is to be explicit in choosing a font. We started with just `Math` and `MathBold` abstractions but then found that Firefox can handle the first, and Chrome can't so we went for `MathNormal` instead. For bold we have to put a higher weight on the element in the css, because a font face rule doesn't work here. The max permitted weight is 900 which is not enough but we have to take what we can get here. Full bold math is anyway only needed in section headings and such and then often is relatively simple.

In Figure 2.1 we see a reason for not enriching the MathML too much with respect to fonts, styles and dimensions. It took us a while to accept this as a side effect in Firefox: the chosen variant depends on a combination of font size and scaling. The large $\sum$ is taken from (in our case) the configured Bonum font but as we scale in the browser it can come from (on our systems) Cambria. There is some heuristic kicking in that is platform dependent as well as depends on what fonts are installed. There is no way to control this. Other subtle shape changes are in the substack rendering and it looks like sometimes some scaling happens in one dimension only. However, at least there is no character width (vs. italic correction) mix up here.

Spacing

We did play a bit with `lspace` and `rspace` to see if our more advanced spacing model could be supported but lack of preceding sibling as well as not really working backward rules made us decide to also draw a line here, at least for now. Think of:

---

[1] The default can be serif or sans, where sans for math is a bad choice for a default.

$$\textstyle\sum_{\substack{0<j<n\\0\le i\le m}} P(i,j) \text{ vs. } \sum_{\substack{0<j<n\\0\le i\le m}} P(i,j)$$

$$\textstyle\sum_{\substack{0<j<n\\0\le i\le m}} P(i,j) \text{ vs. } \sum_{\substack{0<j<n\\0\le i\le m}} P(i,j)$$

$$\textstyle\sum_{i=1}^{p} \sum_{j=1}^{q} \sum_{k=1}^{r} a_{ij} \text{ vs. } \sum_{i=1}^{p} \sum_{j=1}^{q} \sum_{k=1}^{r} a_{ij}$$

10pt @ 100%

$$\textstyle\sum_{\substack{0<j<n\\0\le i\le m}} P(i,j) \text{ vs. } \sum_{\substack{0<j<n\\0\le i\le m}} P(i,j)$$

$$\textstyle\sum_{\substack{0<j<n\\0\le i\le m}} P(i,j) \text{ vs. } \sum_{\substack{0<j<n\\0\le i\le m}} P(i,j)$$

$$\textstyle\sum_{i=1}^{p} \sum_{j=1}^{q} \sum_{k=1}^{r} a_{ij} \text{ vs. } \sum_{i=1}^{p} \sum_{j=1}^{q} \sum_{k=1}^{r} a_{ij}$$

10pt @ 200%

$$\textstyle\sum_{\substack{0<j<n\\0\le i\le m}} P(i,j) \text{ vs. } \sum_{\substack{0<j<n\\0\le i\le m}} P(i,j)$$

$$\textstyle\sum_{\substack{0<j<n\\0\le i\le m}} P(i,j) \text{ vs. } \sum_{\substack{0<j<n\\0\le i\le m}} P(i,j)$$

$$\textstyle\sum_{i=1}^{p} \sum_{j=1}^{q} \sum_{k=1}^{r} a_{ij} \text{ vs. } \sum_{i=1}^{p} \sum_{j=1}^{q} \sum_{k=1}^{r} a_{ij}$$

20pt @ 200%

Figure 2.1 Hard to predict fallback fonts and side effects.

```
/* https://developer.mozilla.org/en-US/docs/Web/CSS/:has */

.digit:has( + .relation) {
       color        : rgb(60%,60%,0%) ; /* works   */
       margin-right : .2em ;            /* doesn't */
}

.relation:has( + .radical) {
       color        : rgb(0%,60%,60%) ; /* works   */
       margin-right : .2em ;            /* doesn't */
}
```

with:

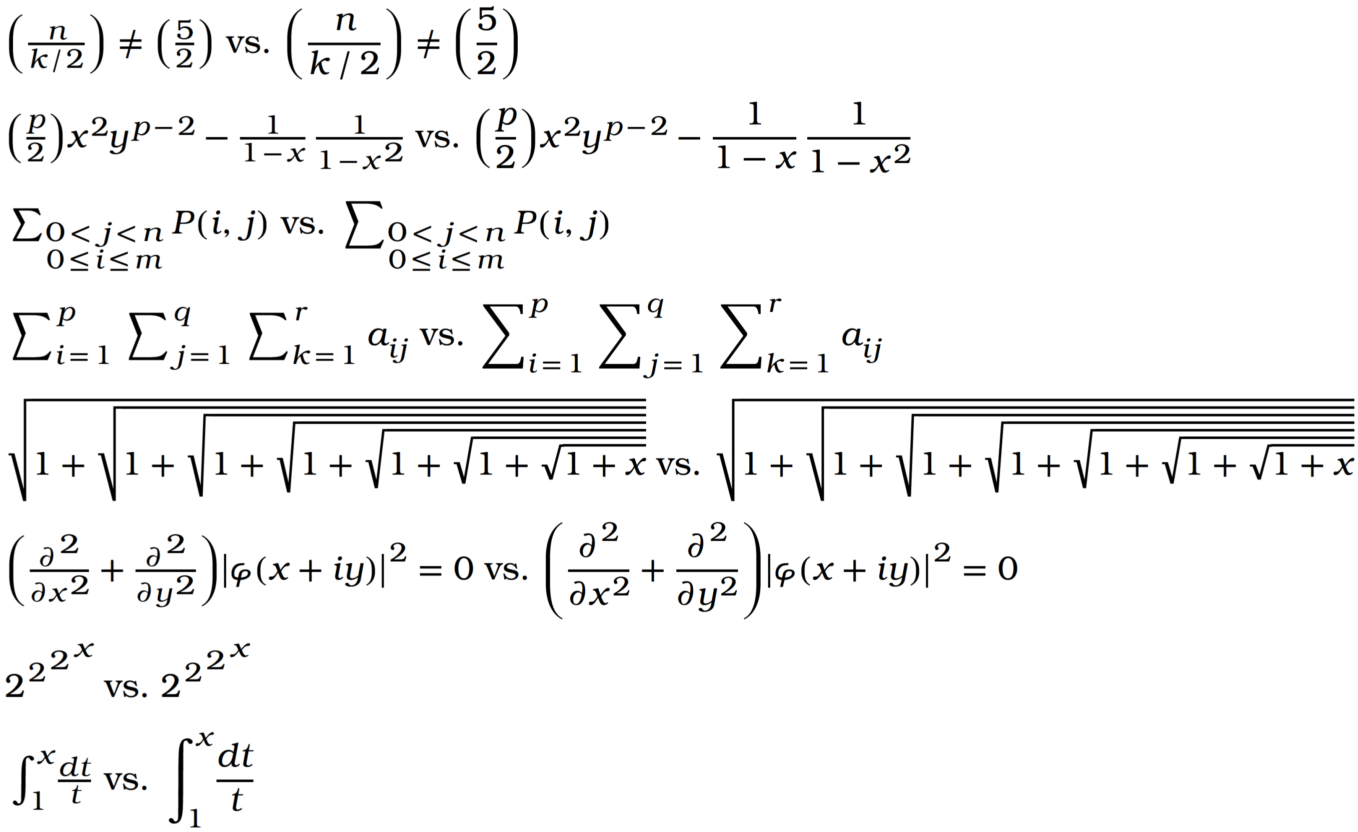
```
<math xmlns="http://www.w3.org/1998/Math/MathML">
    <mrow>
        <mn class="digit">1</mn>
        <mo class="relation">=</mo>
        <msqrt class="radical">2</msqrt>
    </mrow>
</math>
```

Default spacing between for instance binary operators and integrals is rather bad and there is not much we can do about that. Of course using `<mspace>` is an option but that’s kind of ugly. So we settled for “One gets what one deserves.” here. We’re decades from when this all started and it is telling that we’re still not there.

Accuracy

Accuracy is another aspect. In the process of improving math rendering in ConTEXt we paid a lot of attention to detail. Not all fonts are perfect and tweaks are needed and we understand that this is not an option in browsers. However, some things are easy to fix, like making sure that the pieces that make a radical align at least as good as possible: being one rule thickness off definitely suggests a fix. And when rules are used (instead of characters) it is no problem to get the width right. In Figure 2.2 some nested radicals are shown. The right blob in the top corner is part of a choice to translate from Japanese, so somehow this page with math is seen as such, on MS Windows as well as linux. If one peeks into the reported bugs it looks like spacing in radicals is a persistent problem and not really gets solved. In Figure 2.4 one can also observe that mixing radicals gives an inconsistent look and feel, something that we always paid a lot of attention to in ConTEXt but can’t really control reliable here.

When look a bit longer at this figure you also start to notice the spacing between atoms, scripts and individual characters. Much probably goes unnoticed when scrolling on a display, or watching at a smaller size. It makes one wonder how quality control happens today, because the large tech organizations behind this technology for sure have (money for) testing capabilities. Of course it can be that the expectations are low so maybe the TEX community should educate the online world a bit better on the quality they expect.



$\tbinom{n}{k/2} \neq \tbinom{5}{2}$ vs. $\dbinom{n}{k/2} \neq \dbinom{5}{2}$

$\tbinom{p}{2} x^2 y^{p-2} - \tfrac{1}{1-x} \tfrac{1}{1-x^2}$ vs. $\dbinom{p}{2} x^2 y^{p-2} - \dfrac{1}{1-x} \dfrac{1}{1-x^2}$

$\textstyle\sum_{\substack{0<j<n \\ 0\leq i\leq m}} P(i,j)$ vs. $\displaystyle\sum\nolimits_{\substack{0<j<n \\ 0\leq i\leq m}} P(i,j)$

$\textstyle\sum_{i=1}^{p} \sum_{j=1}^{q} \sum_{k=1}^{r} a_{ij}$ vs. $\displaystyle\sum\nolimits_{i=1}^{p} \sum\nolimits_{j=1}^{q} \sum\nolimits_{k=1}^{r} a_{ij}$

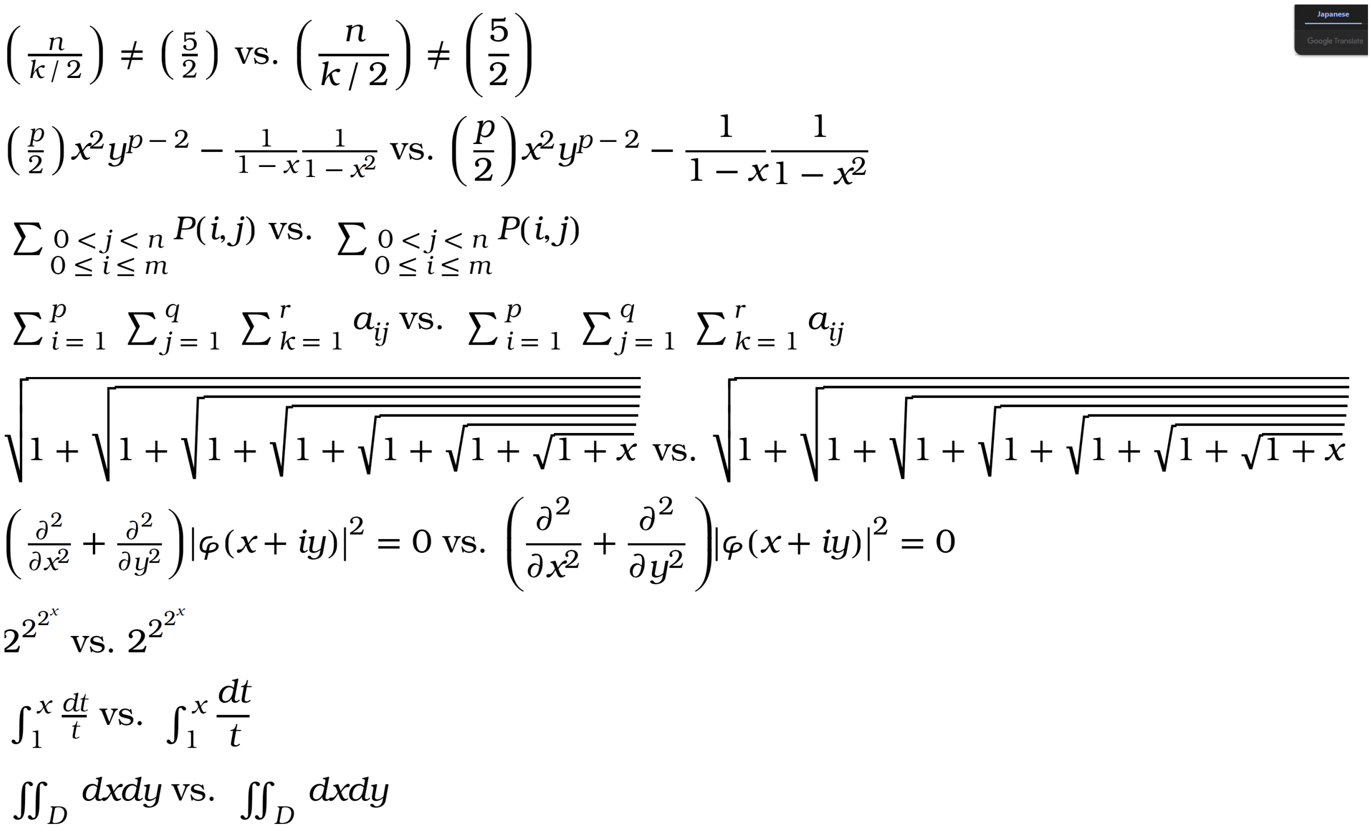
$\sqrt{1+\sqrt{1+\sqrt{1+\sqrt{1+\sqrt{1+\sqrt{1+\sqrt{1+x}}}}}}}$ vs. $\sqrt{1+\sqrt{1+\sqrt{1+\sqrt{1+\sqrt{1+\sqrt{1+\sqrt{1+x}}}}}}}$

$\left(\tfrac{\partial^2}{\partial x^2} + \tfrac{\partial^2}{\partial y^2}\right) |\varphi(x+iy)|^2 = 0$ vs. $\left(\dfrac{\partial^2}{\partial x^2} + \dfrac{\partial^2}{\partial y^2}\right) |\varphi(x+iy)|^2 = 0$

$2^{2^{2^x}}$ vs. $2^{2^{2^x}}$

$\textstyle\int_1^x \frac{dt}{t}$ vs. $\displaystyle\int_1^x \frac{dt}{t}$

firefox



$\tbinom{n}{k/2} \neq \tbinom{5}{2}$ vs. $\dbinom{n}{k/2} \neq \dbinom{5}{2}$

$\tbinom{p}{2} x^2 y^{p-2} - \tfrac{1}{1-x} \tfrac{1}{1-x^2}$ vs. $\dbinom{p}{2} x^2 y^{p-2} - \dfrac{1}{1-x} \dfrac{1}{1-x^2}$

$\textstyle\sum_{\substack{0<j<n \\ 0\leq i\leq m}} P(i,j)$ vs. $\textstyle\sum_{\substack{0<j<n \\ 0\leq i\leq m}} P(i,j)$

$\textstyle\sum_{i=1}^{p} \sum_{j=1}^{q} \sum_{k=1}^{r} a_{ij}$ vs. $\textstyle\sum_{i=1}^{p} \sum_{j=1}^{q} \sum_{k=1}^{r} a_{ij}$

$\sqrt{1+\sqrt{1+\sqrt{1+\sqrt{1+\sqrt{1+\sqrt{1+\sqrt{1+x}}}}}}}$ vs. $\sqrt{1+\sqrt{1+\sqrt{1+\sqrt{1+\sqrt{1+\sqrt{1+\sqrt{1+x}}}}}}}$

$\left(\tfrac{\partial^2}{\partial x^2} + \tfrac{\partial^2}{\partial y^2}\right) |\varphi(x+iy)|^2 = 0$ vs. $\left(\dfrac{\partial^2}{\partial x^2} + \dfrac{\partial^2}{\partial y^2}\right) |\varphi(x+iy)|^2 = 0$

$2^{2^{2^x}}$ vs. $2^{2^{2^x}}$

$\textstyle\int_1^x \frac{dt}{t}$ vs. $\textstyle\int_1^x \dfrac{dt}{t}$

$\textstyle\iint_D dxdy$ vs. $\textstyle\iint_D dxdy$

chrome

Figure 2.2 Browsers differ in the quality spacing.

Extensibles

A nasty thing in MathML is the application of `stretchy`, `symmetric` and `largeop` for instance applied to integrals and fences. This is pretty much database driven although there are also some heuristics involved. Firefox (mid 2024) is happy when one sets the first one, and a controlled setting is really needed due to these somewhat unfortunate dictionary driven defaults. Here the lack of `<mfenced>` shows. At this time Chrome can't handle integrals that grow (as in Bonum): they stay small and are below the axis but we expect this to be fixed. So far an attempt to get more consistent spacing in radicals and fractions failed because here Chrome can be tuned using the dimensions of `<mspace>` while Firefox ignores these. Also setting `minheight` on large operators is not applicable because here Firefox works sort of expected but Chrome goes wild. Maybe the right larger variant is chosen (dimension wise) but the wrong glyph is referenced.

Another observation is that the choice for a size variant in Chrome can be absent completely and that in display mode the second size of the smaller or maybe the larger integral is chosen and then positioned wrong. There are plenty of reasons to use Firefox as the reference implementation. Side note: Edge of course behaves like Chrome.

Nested `<mover>` scale down per nesting in Chrome but not in Firefox, so control with `displaystyle` and `scriptlevel` are disappointing, as shown in Figure 2.3. It looks like Chrome has more than three sizes. Here are some examples where we set the script level on fraction. Setting the level on the numerator and denominator has a similar result. This means that one can best *not* try to optimize the rendering by setting the styles: it can only get worse.

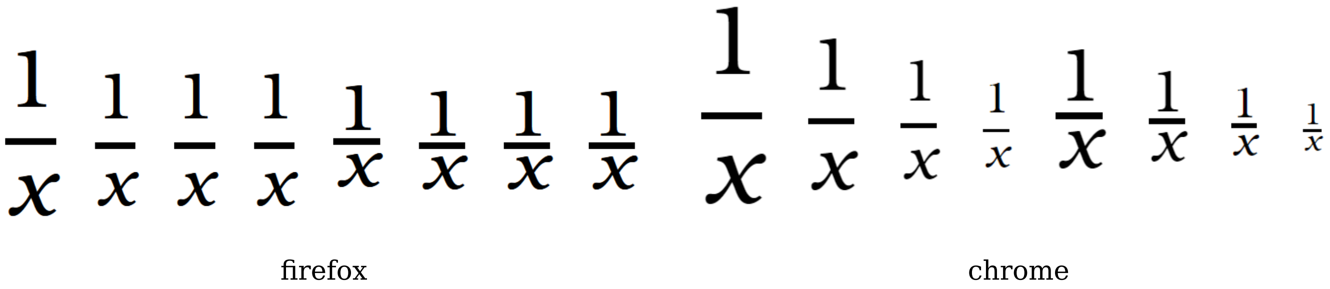
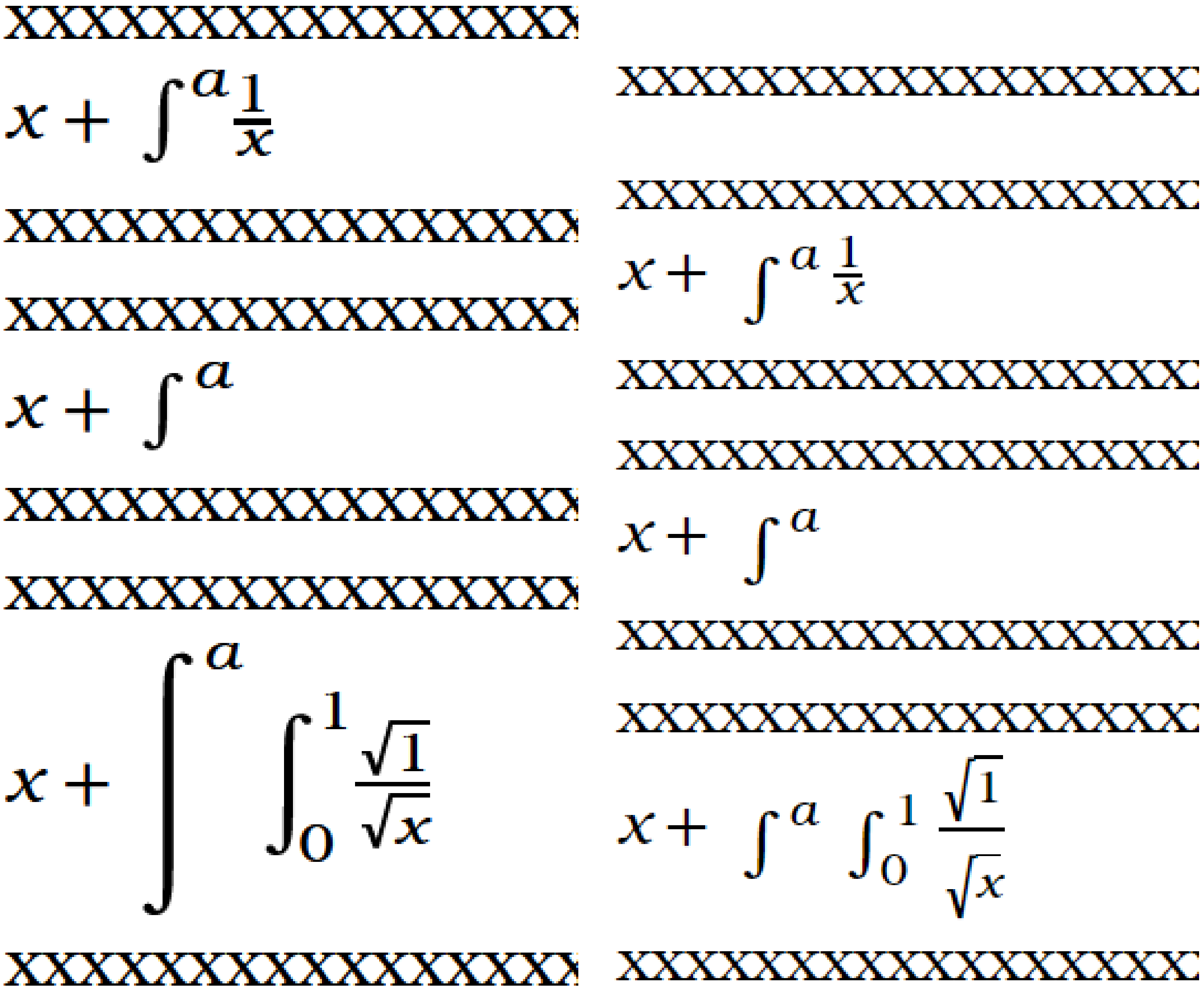


Figure 2.3 Different interpretations of `scriptlevel`.

In Figure 2.4 we see some bad variant choices, wrong positioning and support for minimum height. Here we set `minheight` to `4em` (we often use `em` as measure for as it often relates to the design size). Notice again that the spacing of scripts in Chrome also needs work.

Scaling

After converting some math extensive documents we've come to the conclusion that although a lot is possible today it can differ per browser and version. When we want to show formulas properly rendered as svg we have to deal with scaling and in the end we decided to just use the same bodyfont and line width in the browser as in the original document. Our test main test document however used a 8.5 point body font which is pretty small for a browser. However, we can set the `zoom` in the outer element to for instance 8.5/12 to get our preferred scaling up to 12 points. When testing that we found that on Mikaels machine Firefox didn't support that while on my machine it worked. Upgrading Firefox on the chromebook resulted in a massive replacement of libraries, a broken Okular, and eventually a non functioning virtual machine. Of course in the end trying out several different variant worked out (there's always some solution) but it also learns that long term stability is a dream, especially when

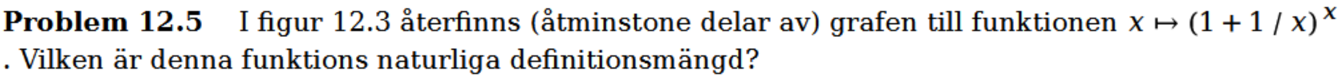


firefox chrome

Figure 2.4 Variants and extensibles.

you come from a TeX ecosystem where a single binary with no dependencies (LuaMetaTeX) does the job. Nevertheless we decided to go the “original font size and width combined with zoom in browser” route because it frees us from messing with scales in the images.

When testing how scaling works out, we also noticed that inline math has its own problems with scaling because linebreaks in math formulas are not supported which can result in overflowing lines as well as badly adjusted lines. A peculiar side effect is shown in figure 2.5 where the mid 2024 browsers happily break a line between a math formula and a period. We just assume that this will be addressed and will not waste time on solutions.



Problem 12.5 I figur 12.3 återfinns (åtminstone delar av) grafen till funktionen $x \mapsto (1 + 1 / x)^x$
. Vilken är denna funktions naturliga definitionsmängd?

Figure 2.5 Linebreaks before periods.

Notes

Given the focus on MathML core one can wonder if the rest of the standard is wishful thinking, or a historic overview. It would have been nice if some more specific features had been added, like native matrix rendering with appropriate spacing. The example of typesetting a matrix using grid styling is somewhat pathetic and if we go the same route with MathML as with svg, that is more and more style and css trickery, we're getting away from structure even more. Of course one can argue that a matrix is a grid but putting style attributes on the element in order to control the columns and rows contradicts otherwise present attributes. One cannot delegate this to a separate style unless each matrix wrapping `<mrow>` gets its own class, which of course makes cut'n'paste no longer an option. Now, a user agent that interprets the MathML also has to support css.

We like to use a decent coded xml file because it's cleaner but while browsers and css are very powerful some 'trivial' features are not supported, like hyperlinks that assume the `<a>` element being used in a regular html file. One can conveniently format an xml file with css but as of now we can't use for instance `xlink:*` attributes; these are only supported for svg and MathML. So it's a disappointment that in 2024 we still have to bloat to a `<div>` for everything variant as we strongly object to using a using a curious mix of html for main element and `<div>` for structure elements that we miss.

Conclusion

So from this subset of examples these cases we decided to draw a line and not spend too much time on optimizing the look and feel of an export. After all, its main purpose is to have a variant of the result that can reflow and be used for (e.g.) accessibility purposes. For that the quality of the rendering in whatever form can be delegated to the web client anyway. We can only hope that bugs, some of which are long standing, gets resolved. Of course solving our own bugs as soon as they are reported remains our priority.