Ideas for ε-TEX/$\mathcal{NTS}$

Matthias Clasen

This document collects some ideas for improvements to TEX's math typesetting. Many of the items listed here can already be found in the archives of the NTS-L mailing list, others have come up in discussions with Ulrik Vieth and Michael J. Downes.

• Provide an integer parameter `\mathstyle` where `\ifnum\mathstyle=0` is true when the current style is displaystyle, 1=textstyle, 2=scriptstyle, 3 and up = scriptscriptstyle. This parameter could well be writable; `\mathstyle=0` would then be a synonym for `\displaystyle`.

To discriminate between cramped and non-cramped styles, a new primitive `\ifcramped` would be needed. We propose a separate `\if` rather than more values for `\mathstyle`, since font size does not depend on 'crampedness'. A logical companion for `\ifcramped` would be a `\cramp` primitive to switch to a cramped style.

In order for this to be workable the syntax of the six generalized fraction commands `\over`/`\above`\atop`[withdelims]` have to be changed from infix form to some sort of prefix form, because of the way they change the math style of *preceding* items in a math list.

This affects many things in TeX, e.g., `\vrule` could then use `mu` units, `\mathchoice` could be implemented as a simple `\ifcase\mathstyle`, etc.

This is perhaps too much of a change for ε-TEX, since its full benefit lies in the fact that the removal of `\over` and friends makes a much more straightforward implementation of math-processing possible. Adding `\mathstyle` to ε-TEX while retaining `\over` for compatibility reasons will probably lead to even more involved code.

• Allow kerning between all kinds of atoms if they are placed directly next to each other, i.e. not just Ord-Close, but also Open-Ord, etc.

• Provide a way to avoid boxing of subformulas, since that leads to suboptimal spacing in certain situations and prevents `\left...\right` and similar constructs to be broken. Michel J. Downes `breqn.sty` shows that this can be done on the macro level for `\left...\right` at the cost of avoiding almost all of TEX's builtin math spacing and doing it all manually.

• Remove the dependence on strange glyph positioning; accents should be allowed to be on the baseline, radicals should not have to have huge depths. This would enable accent glyphs to serve double duty as over and under accents and increase the chance of non-TEX software being able to use TEX's math fonts.

This should be implemented in a way such that the 'traditional' positioning still works, e.g., through additional fontdimensions, whose absence would signal the 'traditional' setup.

Even better might be the possibility to use a glyph instead of a rule for the radical (some math fonts already provide such glyphs). This would best be done by a change of the `tfm` format, since this information belongs in the font, but an extended syntax like `\radical"270370 rules"271371` or even `\radical"270370271371` would also be an option (but the latter would make the code too long to be read as a single 32bit-integer).

One argument against lowering accents (at least in text fonts) is memory consumption in the most common case:

```
\^{x}

.\kern 0.1389 (for accent)
.\tenrm ^
.\kern -5.13892 (for accent)
.\tenrm x
```

(Memory cost: 2 kerns = 4 words of main mem.)

```
$\hat{x}$

.\vbox(6.94444+0.0)x5.71527
..\hbox(6.94444+0.0)x0.0, shifted 0.63542
...\tenrm ^
..\kern-4.30554
..\hbox(4.30554+0.0)x5.71527
...\teni x
```

(Memory cost: 3 boxes and 1 kern = 23 words of main mem.)

But as the examples clearly show, this argument is not valid as far as math fonts are concerned, since the current TEX does already use the more costly alternative in math mode.

• Reduce the amount of overloading on the fontdimensions used in math mode, most importantly for `axis_height` and `default_rule_thickness`. Berthold Horn mentioned that Lucida needs a different math axis for delimiters, so introducing a separate delimter axis and operator axis independent of the default axis (used in `\vcenter` and

fractions) might be a good idea. Similarly, separating the radical rule thickness and clearance amounts form the default rule thickness (used in fractions, sub/superscript combinations and over/underlines) would be nice. Consider that we have five parameters for big operators, but only one rule thickness for many purposes.

To make these changes backward compatible, ε-TEX would check for the additional fontdimensions and if it doesn't find them, use the corresponding value from the traditional set of fontdimensions.

- Other parameters which influence math typesetting are fixed and could be made accessible (either as fontdimensions or via primitives). Examples are
  - the table describing the spacing.
  - the `clr` parameter used as a minimal overshoot in positioning the radical. The corresponding parameters `\delimitershortfall` and `\delimiterfactor` for delimiters are accessible.
  - the clearance between denominator, numerator and bar.

- There might be a need for a right analogue of radicals in quantum mechanics. I have also seen such constructs in knot theory.

- Horizontal analogues of varchar recipes. This would not break any existing document, since existing fonts simply do not have such beasts. ε-TEX should do the right thing when the last character in the charlist for an math accent is an extensible glyph. This would make it easier to have many sizes of horizontal parens, braces, brackets, etc and might also make the implementation of `\overbrace` easier. This is one of the cases where DSSSL has a flow object class (marks) which is not directly supported in TEX the program.

- Adding support for nested accents to the `\mathaccent` (or a new) primitive would also be very nice, since the macros for that tend to be complicated and slow.

- A primitive for underaccents. This would need some decisions about where to store the needed metric information. I think this would need two additional dimensions: the skew, and one additional parameter to position accent parameters vertically. My proposal would be: Use ⟨*skewchar*⟩-⟨*accentee*⟩ kerning for the former and ⟨*accent*⟩-⟨*accent*⟩ kerning or ⟨*accent*⟩ italic correction for the latter.

The design of this primitive must go hand in hand with generalizing the `\mathaccent` primitives. It should be designed so that it works with glyphs which are designed for the `\mathaccent` primitive.

- More than 16 families. Looking at `tex.web`, it seems that the `fam` is always stored in a byte, so that it shouldn't be difficult to allow 256 families. But it would require changes to the syntax of every primitive which reads a `fam` value (`\textfont`, `\scriptfont`, `\scriptscriptfont`, `\mathchardef`, `\mathchar`, `\mathcode`, `\delcode`, `\delimiter`,`\radical`, `\mathaccent`).

- Separate hash table for math mode. Examples: `\^` in text and `\^` in math should call directly different functions, not force users to remember `\hat` is the math form of `\^`. When used in math mode `\&` seems to give an adequate result but it does not have the right math-symbol class because it is only `\chardef`'d, not `\mathchardef`'d. This could be well solved if `\&` automatically called a different function in math mode.

  More and more LaTeX macros are getting `\relax\ifmmode X\else Y\fi` wrappers which indicates that the current single hash table is *not* the best natural approach.

  (More generally, a separate hash table for each language (where math is treated somewhat like a language), and a common hash table for mode-independent things like `\message` or `\relax`. Then Babel doesn't have to jump through hoops any more to redefine a list of things like `\chaptername`, `\bibliographyname`, `\theoremname`, `\dateform`, every time a language change occurs—just put the language-specific definitions in their proper hash table.)

- The decision to box subformulas at their natural width might lead to poor spacing in certain situations. Compare the examples in figure . This boxing should be avoided at least for `\left\dots\right` constructions, to allow line-breaks. t

- To aid the implementation of more complicated display-math handling (`breqn.sty`), it would be nice to have a generalized `\discretionary` command which allows direct specification of the charged penalty. This change should be coordinated with possible extensions of TEXs hyphenation routines to deal with break-points of varying quality. To make this really useful, discretionaries in math mode would have to be allowed to contain math material.

```
\hbox to 5cm{$a+(b+c)$}
\hbox to 5cm{$a+{(b+c)}$}
\hbox to 5cm{$a+\left(b+c\right)$}
```

$$a + (b + c)$$
$$a + (b + c)$$
$$a + (b + c)$$

Figure 1: Examples of boxed subformulas

• In the current code, setting operator names like sin or log uses ligatures and kerning like ordinary text mode, but the code is separate. Maybe it would be simpler to use true text mode here? There are problems with determining the proper font and font size to use though, if this would be done.

• Change the `tfm` format. Once we start this, a lot of things are possible, e.g.

- go from 8-bit to 16-bits character sets,
- get rid of the 15 heights/15 depths/63 italic corrections limitation,
- introduce a verbose format such as AFM that allows to store arbitrary values,
- get rid of the counter-intuitive interpretation of width and italic correction in math fonts to represent subscript/superscript position, just introduce new fields in the font metrics for the right (and left?) subscript and superscript position,
- get rid of the whole `\skewchar` business to represent accent positions, just introudce new fields in the font metrics for the 'center of gravitiy' for over- and under accents.

• Should extensible operators like growing integrals be supported? (Some math fonts already have the necessary glyphs.) If so, we probably need a diagonal variant of extensible characters too, for slanted integrals. The parts of the extensible characters should have enough unused dimensions to allow for the specification of the slant (e.g., the italic correction of the repeatable piece).

If this idea would be combined with having two different extension pieces, things like extensible angles would be possible.

Support for extensible operators would imply that there needs to be a way to specify that a symbol is at the same time extensible and an operator. One way to achieve this would be to make the extensible variant the successor of the `\displaystyle` variant and then treat constructions like `\left\int_a^bf\right.dx` properly.

Open question: How would the width of a slanted extensible glyph be determined?

• Should there be support for left superscripts and subscripts? This would probably be a major change, since it would imply changing the design of mathlists to have four instead of two script fields and might also require a left analogue of italic correction. DSSSL has a flow object class (script) that allows six different scripts to be attached to a math atom: sub, sup, pre-sub, pre-sup, mid-sub and mid-sup.