Macros for doing math more comfortably

Version: 0.16, 2024-10-17

Petr Olšák[1], *2022, 2023*

Table of contents

1 Summary

This package provides various extensions usable for math typesetting. Mostly of them are inspired from OpTEX tricks www page.

The following macros are defined in this package:

- \bigp, \bbigp, \Bigp, \biggp, \Biggp, \autop, \normalp gives better controlling of sizes of parentheses.
- \smartdots declares \dots macro more intelligent. \smartvert declares "|" for better spacing.
- \rmsbsp activates roman subscripts and superscripts in [...].
- There are many common math macros for sets or for operators, for example \R or \sgn.
- \eqsystem enables to write systems of equations comfortably,
- \toright, \toleft puts the \eqmark to desired position, \subeqmark prints the given suffix as a part of the equation mark.
- \scriptspaces sets more spaces around rel, bin in script and scripscript styles.
- \bfserif sets \bf and \bi for math typesetting as for bold-serif, bold-italic-serif.
- \flexipa enables flexible partial symbol.
- \rmchars sets selected characters printed as \rm, \vargreek sets Greek leters to their variants.
- \textvariables, \textdigits, \textmoremath enables characters from used text font in math mode (variables, digits, more characters).
- \replacemissingchars allows to re-declare all characters missing in math font for printing them from additional math font.
- \enablefic enables final italic correction of inline-math lists.

[1] https://petr.olsak.net

Following options are provided by the `math` package. You can set them by `\mathsetup{`⟨*options*⟩`}` after `\load[math]`, for example `\mathsetup{dots, vert, vargreek={\epsilon,\rho}}`. The options are:

- `dots` sets more intelligent `\dots`, the same as `\smartdots`.
- `interval` sets `\_` as prefix of intervals, see section 4.
- `vert` sets more intelligent `|`, the same as `\smartvert`.
- `rmsbsp` sets roman sub/supscripts in `[...]`, the same as `\rmsbsp`.
- `bfserif` sets bold-serif, bold-italic-serif, the same as `\bfserif`.
- `flexipa` enables flexible partial symbol, the same as `\flexipa`.
- `mstyle=`⟨*style*⟩, `bstyle=`⟨*style*⟩ are math styles explained in section 11.
- `rmchars={`⟨*list*⟩`}` sets `\rm` for selected characters, the same as `\rmchars`, see section 12.
- `vargreek={`⟨*list*⟩`}` sets variants for Greek letetters, the same as `\vargreek`, see section 12.
- `text={`⟨*list*⟩`}` sets `\textvariables`, `\textdigits`, or `\textmoremath`, see section 14,
- `enablefic` enables final italic correction of inline-math lists, does `\enablefic`.

This package is not definitive. I plan to add more features in new versions if needed. Moreover, this package gives an example for package writers how to write their own packages, see section 18.

2 Controlled sizes of parentheses

If you write `$f(x(y+z))$` then the outer parentheses should be bigger. Classical Plain TEX provides macros `\bigl`, `\bigr`, etc., they can be used in this manner: `$f\bigl(x(y+z)\bigr)$`. But the source file looks bad with such markup. Better is to say that parentheses have to be bigger using a single prefix before functional symbol, i.e. `$\bigp f(x(y+z))$`. This should be print the same as previous example with `\bigl`, `\bigr`.

The prefixes `\bigp` (big pair), `\bbigp` (bbig pair), `\Bigp` (Big pair), `\biggp` (bigg pair) and `\Biggp` (Bigg pair) are provided, they can be used before a functional symbol. The scaled parentheses surrounding the functional parameter can be (...) or [...] or `\{...\}` or {...}. I.e. `\Bigp\Gamma [x]` is the same as `\Gamma \Bigl[x\bigr]`. Moreover, the functional parameter gets its own TEX group, so `\Bigp G(a\over2)` results to `G\Bigl({a\over2}\bigr)`. There are two more prefixes `\autop` and `\normalp`. First one applies `\left`, `\right` to the parentheses of the parameter, second one keeps the parentheses unscaled. If you want to scale the parentheses without preceding functional symbol then use dot instead this symbol, for example `\Bigp.(a)` is equal to `\Bigl(a\Bigr)`.

Examples:

```
$$
  \displaylines{
  \biggp F (1+\Bigp g (1+\bbigp f(1+\bigp f(1+f(x))))) \cr
  f(x(y+z)),\quad  \bigp f(x(y+z)),\quad \autop f (a\over b)\cr
  \Bigp f(a\over b+c),\quad \Bigp f(x^2\over2),\quad \Bigp.(a\over b)
  }
$$
```

gives:

$$F\Bigl(1+g\bigl(1+f\bigl(1+f(1+f(x))\bigr)\bigr)\Bigr)$$

$$f(x(y+z)),\quad f(x(y+z)),\quad f\Bigl(\frac{a}{b}\Bigr)$$

$$f\Bigl(\frac{a}{b+c}\Bigr),\quad f\Bigl(\frac{x^2}{2}\Bigr),\quad \Bigl(\frac{a}{b}\Bigr)$$

3 Intelligent `\dots` like in AMSTEX

AMSTEX provides `\dots` macro which works depending on the context. If it is surrounded by symbols like +, −, = then it works like `\cdots`, if it is surrounded by comma or similar symbols then it works like `\ldots`. This package keeps `\dots` unchanged but it is changed (and behaves as mentioned above) after the `\smartdots` declaration.

You can try this after the `\smartdots` declaration:

`$a_1, a_2, \dots, a_n$` prints $a_1, a_2, \dots, a_n$,
`$a_1 + a_2 + \dots + a_n$` prints $a_1 + a_2 + \cdots + a_n$,

4 Creating intervals more comfortable

Several math books uses `\langle`, `\rangle` for denoting the interval boundary if the boundary number is element of the interval too. For example `$\langle 0,1)$` is printed as $\langle 0,1)$ and it means an interval from zero to one, zero is element of the interval but one isn't. The TEX source of such math books looks badly because we cannot mark them like `$<0,1)$` because it prints $<0,1)$ but we want $\langle 0,1)$.

This package creates the `\.interval` macro which can be set to be equal to `\_` (using the `interval` option). Then `\_` can be used just before the interval as a prefix. The `<` or `>` are automatically replaced by `\langle`, `\rangle` if `\_` is prefixed. So, you can write `$\_<0,1)$` or `$\_(0,1>$` or `$\_<0,1>$` or `$\_(0,1)$` in order to get $\langle 0,1)$ or $(0,1\rangle$ or $\langle 0,1\rangle$ or $(0,1)$. The source with such intervals looks better.

Note that `interval` option does `\let\_=\.interval`, so the original meaning of the `\_` control sequence (from plain TEX) is re-defined. The `\_` control sequence can be used for this purpose because the next token is `(` or `<`, i.e. it is non-letter.

5 Using vertical bars with better spacing

The character "`|`" is declared with Ord class by default in Plain TEX, but we are using it typically in the context `$|x|$`. It means there should be Open and Close classes. This example gives correct result but try to use `$|-1|$` which gives bad spacing: $|-1|$. And `$||x||$` gives bad result too.

When you declare `\smartvert`, these problems are solved. Moreover, the "`|`" or "`||`" are expected to be always in pairs and they are scaled by `\left` and `\right` primitives automatically. If you don't want to use it in a pair, use `\singlevert` or `\big|`, or `\Big|` etc. Compare the result of `$|\sum a_n|+||x||$`:

$$|\sum a_n| + ||x|| \quad \text{if } \texttt{\textbackslash smartvert} \quad \text{isn't initialized,}$$
$$\left|\sum a_n\right| + \|x\| \quad \text{if } \texttt{\textbackslash smartvert} \quad \text{is initialized.}$$

6 Roman subscript and superscript in `[...]`

When you declare `\rmsbsp`, then you can write `x_[text]` or `x^[text]` and it is equivalent to `x_{\mathbox{text}}` or `x^{\mathbox{text}}`.

7 Basic and typical macros for sets, functions etc.

These typical macros are defined in `math.opm`: `\N` for $\mathbb{N}$, `\Z` for $\mathbb{Z}$, `\Q` for $\mathbb{Q}$, `\R` for $\mathbb{R}$, `\C` for $\mathbb{C}$, `\sgn`, `\argmin`, `\argmax`, `\grad`, `\rank`, `\tr`, `\diag`, `\Span`, `\Rng`, `\Null`, `\Ker`, `\Res`, `\tg`, `\cotg`, `\arctg`, `\arccotg`.

I hate the `\frac`, `\dfrac` and `\tfrac` macros defined in LATEX but someone may want to use them. This package defines them. But I note: usage of `$1\over2$` for $\frac12$ is much more understandable than LATEX's `$\frac12$`.

The vectors and matrices are usually printed by `{\bf A}{\bf x}`. The package provides a shortcut `\.`⟨*letter*⟩ to do the same, so user can write `\.A\.x` for multiplication of a matrix $\mathbf{A}$ by a vector $\mathbf{x}$. We strictly don't recommend usage of `\.`, `\v`, `\=`, etc. for accents, so `math.opm` can define `\.` differently than the classical meaning "dotaccent".

8 System of equations printed by `\eqsystem`

The `\eqsystem{`⟨*equations*⟩`}` enables to write systems of equations more comfortably. The equations are separated by `\cr` and the aligned columns are separated by space. For example:

```
$$
  \eqsystem{  x +  y - 2z = 10 \cr
             2x - 7y +  z = 13 \cr
             -x +  y ~  ~ = -5 }
$$
```

prints

$$\begin{aligned} x+\ \ y-2z &= \ 10\\ 2x-7y+\ \ z &= \ 13\\ -x+\ \ y\ \ \ \ \ &= -5 \end{aligned}$$

Note that empty columns have to be filled by `~` mark. There are columns for variables (possibly multiplied by a constant) and for binary operators `+` and `-` or relations `=`, `>` etc. or constants. Each column is aligned to right. The number of columns is unlimited (we have 7 columns in the example above). All given equations are packed to the `\vcenter` box.

The spaces between lines are enlarged by the value of `\eqskip` and the horizontal spaces between columns are enlaged by `\eqsep`. Both registers are set to 0 pt by default.

The `\eqfil` register is "left filler" applied to each item in the `\eqsystem` columns. Its default is `\eqfil={\hfill}`. The right filler is hardwired and it is `\hfil`. This makes columns aligned to right by default. For example, when you set `\eqfil={\hfil}` then you have columns centered.

The `\eqsystem` macro allows optional parameter which is processed inside group before printing equations. You can do local settings here, for example `\eqsystem[\eqskip=2pt \eqsep=5pt]{...}`.

9 Equation marks in atypical cases

We may want to put equation marks `\eqmark` in more lines in display mode when we are using macros not designed for such case. For example in the lines of the `\cases` macro:

```
$$ f(x) = \cases{0 & for $x<0$\toright\eqmark \cr
                 1 & otherwise\toright\eqmark } $$
```

This puts the equation marks to the right margin in each line generated by the `\cases` macro.

$$f(x) = \begin{cases} 0 & \text{for } x < 0 \\ 1 & \text{otherwise} \end{cases} \tag{1}\tag{2}$$

The `\toright\eqmark` is used here. Analogically, `\toleft\eqmark` puts the equation mark to the left margin. The position of these marks are correct after second or more TEX run because TEX needs to read data from its previous run in this case.

Sometimes we want to declare a bunch of equations with the same numeric equation marks but with different suffixes, for example (1.1a), (1.1b). We provide the macro `\subeqmark`⟨*suffix*⟩ here. If ⟨*suffix*⟩ is `a` or `A` then `\subeqmark` starts a new bunch of equations with the next number. Following `\subeqmark b`, `\subeqmark c`, etc. use the same equation number, they differ only by given suffixes: You can put `[`⟨*label*⟩`]` after ⟨*suffix*⟩ for referencing purposes. Example:

```
$$ \eqsystem[\eqskip=3pt]{
     x + 2y + 3z = 600 \toright{\subeqmark a}\cr
   12x +  y - 3z = -7  \toright{\subeqmark b[label]}\cr
    4x -  y + 5z =  5  \toright{\subeqmark c}\cr }
$$
The equation~\ref[label] has negative right side. Moreover, it applies
$$
  a^2 + b^2 = c^2. \eqmark
$$
```

prints

$$\begin{aligned} x + 2y + 3z &= 600 && \text{(3a)} \\ 12x + y - 3z &= -7 && \text{(3b)} \\ 4x - y + 5z &= 5 && \text{(3c)} \end{aligned}$$

The equation (3b) has negative right side. Moreover, it applies

$$a^2 + b^2 = c^2. \tag{4}$$

10 Setting more spaces in script styles

Classical TEX puts `\thickmuskip` around relations and `\medmuskip` around binary operators only in `\textstyle` and `\displaystyle`. These spaces are missing in `\scripstyle` and `\scriptscriptstyle`. It means that we get, for example

$$\sum_{i=j+1}^{\infty} a_i$$

The formula $i = k + 1$ has no spaces here, so it looks unattractive. LuaTEX provides better control of all such spaces, so `math.opm` declares the macro `\scriptspaces{`⟨*s-rel*⟩`}{`⟨*s-bin*⟩`}{`⟨*ss-rel*⟩`}{`⟨*ss-bin*⟩`}` for setting these spaces. ⟨*s-rel*⟩ is "muskip" value used around relations in `\scriptstyle`, ⟨*s-bin*⟩ is "muskip" used around binary operators in `\scriptstyle` and the last two parameters gives these spacing in `\scriptscriptstyle`. If a parameter is empty, it means that it has zero value. For example after `\scriptspaces {2mu}{1.3mu}{}{}` the formula mentioned above looks like

$$\sum_{i=j+1}^{\infty} a_i$$

It looks better, doesn't it?

11 \bf and \bi shapes, math styles

OpTEX sets `\bf` and `\bi` math selectors as sans serif, because this follows the old traditional math typesetting of vectors and matrices. But Knuth's TEX has another default behavior:`\bf` and `\bi` select serifed shapes. So, many people consider it as a standard. You can declare `\bfserif` if you want serifed `\bf` and `\bi` math letters.

Moreover, this package provides `mstyle=`⟨*style*⟩ and `bstyle=`⟨*style*⟩ options. The `mstyle` option can be `TeX`, `ISO`, `french` or `upright` and `bstyle` option can be `TeX`, `OpTeX`, `ISO`, `upright`. The `mstyle` and `bstyle` options set the upright/italic versions of math Latin/Greek variables in the same manner as `math-style` and `bold-style` options (from LATEX's `unicode-math`) do it. The `bstyle=OpTeX` sets sans serif bold variables, which is default in OpTEX.

12 Selected upright letters and variants for Greek letters

Some mathematicians claim that the letters e, i and π in meaning "a constant" should be printed in upright form. TEX prints all variables in math italic, but this package enables to set exceptions for some letters. For example after `\rmchars{e, i, \pi}`, all occurrences of these three letters in math mode will be set in upright shape. If you set this, then the well-known math identity `$e^{i\pi}=-1$` looks like this:

$$\mathrm{e}^{\mathrm{i}\pi} = -1, \qquad \text{compare with:} \quad e^{i\pi} = -1.$$

The syntax is `\rmchars{`⟨*list*⟩`}`, where ⟨*list*⟩ is a list of characters separated by (optional) commas. The character is `a` to `z` or `A` to `Z` or `\alpha` to `\omega`.

If you set a character by `\rmchars` globally and you want to print it in italic locally then use `\mit`, for example `{\mit e}` prints e.

Several Greek letters have their variant shape: `\epsilon` ϵ, `\varepsilon` ε, `\sigma` σ, `\varsigma` ς, `\phi` ϕ, `\varphi` φ, `\theta` θ, `\vartheta` ϑ, `\pi` π, `\varpi` ϖ, `\kappa` κ, `\varkappa` $\varkappa$, `\rho` ρ, `\varrho` ϱ, `\Theta` Θ, `\varTheta` Θ. Maybe, there is a tradition of usage variant shapes instead of standard ones in your mathematics field. Then you can use `\vargreek{`⟨*list*⟩`}`, where ⟨*list*⟩ includes the list of no-var control sequences for these letters (separated by optional comma). For example `\vargreek{\epsilon \phi \rho}` causes that `\epsilon` is printed as ε, `\phi` as φ and `\rho` as ϱ.

If you want to declare a Greek letter by both `\vargreek` and `\rmchars`, use `\rmchars` first.

The package provides two options `rmchars` and `vargreek`. The equation sign must follow and then the `{`⟨*list*⟩`}` with syntax mentioned above. For example `\mathsetup{vargreek={\epsilon,\rho}}`.

13 Flexible partial symbol ∂

Classical TEX with Computer Modern fonts uses slanted `\partial` symbol ∂. On the other hand, default setting of Unicode math gives upright `\partial` ∂ but six variants of this symbol are provided: roman (upright) `\mrmpartial` ∂, bold `\mbfpartial` $\mathbf{\partial}$, italic `\mitpartial` $\mathit{\partial}$, bold italic `\mbfitpartial` $\boldsymbol{\partial}$, bold sans serif `\mbfsanspartial` $\mathbf{\partial}$, bold italic sans serif `\mbfitsanspartial` $\boldsymbol{\partial}$.

When you declare `\flexipa` or `\flexiblepartial` or use the `flexipa` package option, then the `\partial` symbols get italic variant as default and behaves like others Greek symbols: it changes its variant according to the `\rm`, `\it`, `\bf`, `\bi` selectors in math mode. You can declare what variant of the `\partial` character will be shown at what selector. This can be done by the `\partialsymbolvars` ⟨*default*⟩ ⟨*rm*⟩ ⟨*bf*⟩ ⟨*it*⟩ ⟨*bi*⟩ macro. Each of these five parameters can be `{rm}` or `{bf}` or `{it}` or `{bfit}` or `{bfsans}` or `{bfitsans}`. For example the third parameter declares what

variant of the partial symbol is printed when `\bf` selector is used in math mode. The `\flexipa` macro runs the following default setting:

```
\partialsymbolvars {it} {rm} {bfsans} {it} {bfitsans}
```

It means that default is italic variant and when `\rm` is selected then roman (upright) variant is printed, when `\bf` is selected then bold sans serif variant is printed etc.

14 Variables and digits from currently used text font

When Unicode math font is loaded then all variables and digits are printed from it in math mode. If you are using text fonts with another visual concept then you can see a differences when you use digits in text mode and in math mode. You can specify `\textdigits` if you want to use digits from current text `\rm` font in math and `\textvariables` if you want to use variables from current text `\it` font in math. You can set printing of $+-*/=\langle\rangle\{([])\}$ from text `\rm` font in math by `\textmoremath`. You can inspire from the `\textmoremath` macro and set more similar characters from text font.

You have to load a text font family (using `\fontfam` for example) first and use `\textdigits`, `\textvariables`, `\textmoremath` after it. This is due to these macros reads *current* text `\rm` and `\it` fonts and set them to math printing.

Note that we cannot avoid a visual incompatibility of parentheses when they are use in the context `\left`, `\right`. These parentheses must be printed from math font always because text font is unable to create bigger versions of them.

The package provides the option `text={`⟨*list of words*⟩`}`, each ⟨*word*⟩ from the ⟨*list*⟩ can be `digits` or `variables` or `moremath`. It runs corresponding macro(s) described above. For example `\mathsetup{text=digits variables}` is equal to the declaration of `\textdigits \textvariables`.

15 Replacing all missing math characters from another font

If we load an additional math font by `\addUmathfont`, for example:

```
\addUmathfont \xits {[XITSMath-Regular]}{} {[XITSMath-Bold]}{} {}
```

then we can re-declare the code of arbitrary math character in such a way that it is printed from this additional font. It can be done by `\mathchars` provided by OpTEX, for example:

```
\mathchars \xits {\leftdasharrow \updasharrow \rightdasharrow \downdasharrow}
```

But this method enables to re-declare only selected characters. Maybe, you want to re-declare *all* Unicode math characters which are missing in the main font. This can be done by `\replacemissingchars` ⟨*family*⟩ provided by the `math.opm`. For example

```
\replacemissingchars \xits
```

replaces all characters missing in the main font by characters from the `\xits` declared by previous `\addUmathfont`. The names of all replaced characters are printed in log file. If the additional math font doesn't provide all Unicode math characters then you can load a next additional math font using another `\addUmathfont` and do `\replacemissingchars` ⟨*family*⟩ again. Only those characters not replaced by previous steps are replaced.

16 Final italic correction

Classical TEX adds italic correction after each Ord atom with a single letter from math italic font including the last one in inline-math. LuaTEX and OpTEX does the same only when classical `tfm` math fonts are used. When you switch to Unicode math, the italic correction of the last glyph of the inline-math list is lost, unfortunately. Try this:

```
$T$' the italic correction after $T$ is inserted (classical fonts)

\fontfam[lm]
$T$' the italic correction after $T$ isn't inserted (Unicode fonts)
\bye
```

This package provides the `\enablefic` command which enables the classical TeX feature: the final italic correction of the inline-math list is automatically added.

When \enablefic (or enablefic option) is declared then you can control the behavior of this feature by the \finalitalcorr register: if it is positive then the feature is activated and if it is zero, the feature is deactivated. The \enablefic macro sets \finalitalcorr=1.

17 Miscellaneous commands

I created various commands at the requests of users. They asked me to create commands similar to ones from LaTeX packages.

\mathclap{⟨*formula*⟩} creates {\hbox to0pt{\hss $⟨*formula*⟩$\hss}} and respects the math style. \mathrlap{⟨*formula*⟩} and \mathllap{⟨*formula*⟩} is \rlap and \llap analogue of \mathclap.

18 General recommendation for writing OpTeX packages

This section has nothing common with the subject of this package but this package can serve as inspiration for another package writers. It should be a template for another ⟨*pkg*⟩.opm files. We emphasize several principles here. The basic information can be found in section 2.2 of the OpTeX manual. Try to run[2]

```
optex -jobname math-doc '\docgen math'
```

for creating this documentation. You can see (from the log file) that the math.opm is read four times during this process. First one is due to \docgen math. It skips the part before _endcode and searches the following _doc..._cod pair in the file and processes it (see the end of the file math.opm). The macros and main instruction about generating toc, index, etc. are here. First instruction is \load [doc,math] which initializes doc mode of OpTeX and loads math.opm secondly because we want to show some effects provided by this package. Then there is \printdoctail math.opm which loads the math.opm again and prints the documentation starting from _endcode. Finally, there is \printdoc math.opm which prints the codes mixed by the documentation text inside pairs _doc..._cod. This causes the fourth loading of the math.opm file.

The first part of the math.opm file looks like:

```
% Optional comments

\_def\_⟨pkg⟩_version {⟨version-number⟩, ⟨version-date⟩}
\_codedecl \pkgsequence {Doing the life more comfortable <\_⟨pkg⟩_version>}
\_namespace{⟨pkg⟩}
```

The _⟨*pkg*⟩_version macro should be declared here. The macro should expand to version number followed by version date. User can check the package version simply by expanding this macro after the package is loaded. And we want to have this data only at single place of the file. You may check the log file if the text given by _codedecl isn't too long and isn't broken to more lines. Keeping single line is better because users can grep @: on log file in order to get information of all loaded packages and their version numbers.

The _namespace {⟨*pkg*⟩} opens the name space used by your package where all \.foo are internally transformed to _⟨*pkg*⟩_foo. Next part of the file includes the code itself documented in _doc..._cod pairs. It is finished by _endnamespace which finalizes the scope where \.foo are transformed to _⟨*pkg*⟩_foo and by _endcode which does \endinput when the macros are load. Final part of the file after _endcode can include more detailed documentation.

If your package requires other packages then insert \load [⟨*package1*⟩,⟨*package2*⟩] after _codedecl and before the the _namespace command. Each package uses its own namespace, so it is important to load these packages before your _namespace is opened.

If you have any idea of creating a macro package, you probably start with experimental macros in the public namespace. It means that there are \def\mymacro etc. Once such a code is working, you can include it to the macro package introduced by _namespace {⟨*pkg*⟩}. You have to go through your code carefully sequence per sequence and insert _ or . in front of their names. The "_" prefix have to be used if the sequence is a primitive or an OpTeX macro and the "." prefix if it is your macro. So, the code fragment \def\mymacro have to be rewritten to _def\.mymacro. If the macro \mymacro is intended for end users, then export it to the public name space after it is defined by the _nspublic \mymacro ; command.

[2] Run it three times because Table of contents and Index are created.

Sometimes you may want to define a macro only in public namespace. Then use prefix `\_newpublic` before your declaration, see declaration of `\sgn` in this package as an example. The reason is: if a user has defined such a macro already then the warning is printed. The user can read this warning and declare the macro after `\load[`⟨*pkg*⟩`]` in this case.

19 Implementation

math.opm

```
\_def\_math_version {0.16, 2024-10-17}
\_codedecl \replacemissingchars {Doing math more comfortably <\_math_version>}
\_namespace{math}
```

The `math` package provides options, they can be declared using the `\mathsetup{`⟨*options*⟩`}` macro. For example `\mathsetup {vert, dots}`. If you create your own package with options, provide a similar `\`⟨*pkg*⟩`set` macro. The `\.kv` macro is similar to OpTEX's `\kv`, but with specific `pkg:math` dictionary.

math.opm

```
\_def\.mathsetup #1{%
   \_edef\.restorekvdict{\_kvdict{\_the\_kvdict}}%
   \_kvdict{pkg:math}%
      \_nokvx {\_opwarning{\_the\_kvdict: unknown option "##1", ignored}}%
      \_kvx {vert}     {\.smartvert}%     sets | as math active, to do better |x| or ||x||
      \_kvx {dots}     {\.smartdots}%     \dots behaves like \ldots or \cdots
      \_kvx {interval} {\_let\_=\.interval}% enables \_<0,1)
      \_kvx {rmsbsp}   {\.rmsbsp}%        activates x_[text] and x^[text]
      \_kvx {bfserif}  {\.bfserif}%       \bf, \bi select fonts with serifs
      \_kvx {flexipa}  {\.flexipa}%       flexible partial symbol
      \_kvx {mstyle}   {\.mstyle{##1}}%   sets mstyle=TeX or ISO or french or upright
      \_kvx {bstyle}   {\.bstyle{##1}}%   sets bstyle=TeX or ISO or upright or OpTeX
      \_kvx {rmchars}  {\.rmchars{##1}}%  does \.rmchars{<list>}
      \_kvx {vargreek} {\.vargreek{##1}}% does \.vargreek{<list>}
      \_kvx {text}     {\.dotext{##1}}%   does \.dotext{<list>}
      \_kvx {enablefic}{\.enablefic}%     enables final italic correction
      \_readkv{#1}%
   \.restorekvdict
}
\_def\.kv #1{\_trycs{_kv:pkg:math:#1}{\_kvunknown}}% for accessing values given by \mathsetup

\_nspublic \mathsetup ;
\_newpublic \_let\mathset = \.mathsetup  % for backward compatibility
```

`\bigp`, `\bbigp`, `\Bigp`, `\biggp`, `\Biggp`, `\autop`, `\normalp` are inspired from OpTEX trick 0094

math.opm

```
\_def\.bigp #1{\.fparam{#1}\_bigl\_bigr}
\_def\.bbigp #1{\.fparam{#1}\_bbigl\_bbigr}
\_def\.Bigp #1{\.fparam{#1}\_Bigl\_Bigr}
\_def\.biggp#1{\.fparam{#1}\_biggl\_biggr}
\_def\.Biggp#1{\.fparam{#1}\_Biggl\_Biggr}
\_def\.autop#1{\.fparam{#1}\_left\_right}
\_def\.normalp#1{\.fparam{#1}\_relax\_relax}
\_def\.fparam#1#2#3{%
   \_isequal .{#1}\_iffalse #1\_fi
   \_let\.bigleft=#2\_let\.bigright=#3\_nospacefuturelet\.next\.fparamA}
\_def\.fparamA{%
   \_casesof \.next
   (        {\.fparamB()}%
   [        {\.fparamB[]}%
   \{       {\.fparamB\{\}}%
   \_bgroup {\_def\.lparen{\{}\_def\.rparen{\}}\.fparamC}%
   \_finc   {}%
}
\_def\.fparamB#1#2{%
   \_def\.lparen{#1}\_def\.rparen{#2}%
   \_def\.next#1##1#2{\_trick_ensurebalanced#1#2\.fparamC{##1}}%
   \.next
}
\_def\.fparamC#1{%
   \_ifx\.bigleft\_left \_mathopen{}\_bgroup\_fi
   \.bigleft\.lparen{#1}\.bigright\.rparen
```

```
   \_ifx\.bigright\_right \_egroup\_fi
}
\_nspublic \bigp \bbigp \Bigp \biggp \Biggp \autop \normalp ;
```

We need macro `\.ensurebalanced`⟨*open-b*⟩⟨*close-b*⟩⟨*macro*⟩ for balancing nested parentheses from OpTEX trick 0043. We use `trick` namespace for these macros.

math.opm

```
\_resetnamespace{trick}
   \_def\.ensurebalanced#1#2#3{\_immediateassigned{%
      \_def\.balopen{#1}\_def\.balclose{#2}\_let\.balaction=#3%
      \_def\.readnextbal##1##2#2{\.ensurebalancedA{##1#2##2}}}%
      \.ensurebalancedA}
   \_def\.ensurebalancedA#1{\.isbalanced#1%
      \_iftrue\_afterfi{\.balaction{#1}}\_else\_afterfi{\.readnextbal{#1}}\_fi}
   \_def\.isbalanced#1\_iftrue{\_immediateassignment\_tmpnum=0 \.isbalancedA#1{\.isbalanced}}
   \_def\.isbalancedA#1#{\.countbalanced#1\.isbalanced \.isbalancedB}
   \_def\.isbalancedB#1{%
      \_ifx\.isbalanced#1\_afterfi{\_cs{ifnum}\_tmpnum=0 }\_else\_ea\.isbalancedA\_fi}
   \_def\.countbalanced#1{\_ea\_ifx\.balopen #1\_immediateassignment\_incr\_tmpnum\_fi
                          \_ea\_ifx\.balclose#1\_immediateassignment\_decr\_tmpnum\_fi
                          \_ifx\.isbalanced#1\_else\_ea\.countbalanced\_fi}
\_resetnamespace{math}
```

`\smartdots` re-sets `\dots` to `\.dots`. The `\.dots` lets `\.next` using `\futurelet` and checks the `\.next`: It it is declared by `\_chardef` then `\.mchar` is the real Unicode character with `\chardef`ed code. If `\.next` is a real Unicode character then `\.mchar` includes it. This is done by the expandable `\cstochar` macro provided by OpTEX. If `\.next` is something else (i.e. `\.mchar` is empty) then print `\ldots` else print `\cdots` for Op, Bin, Rel, Open, Close math classes of the `\.next` math object or prints `\ldots` in other cases.

math.opm

```
\_def\.smartdots {\_let\dots=\.dots}
\_def\.dots{\_relax \_ifmmode \_ea\.specdots \_else \_dots \_fi}
\_def\.specdots{\_futurelet\.next\.specdotsA}
\_def\.specdotsA{%
   \.ischardef\.next\_iftrue \_edef\.mchar{\_Uchar\.next}%
   \_else \_edef\.mchar{\_cstochar\.next}%
   \_fi
   \_ifx\.mchar\_empty \_ldots
   \_else \_Umathcharnumdef\.next=\_Umathcode\_ea`\.mchar \_relax
      \_ifcase \.readclass\.next
         \_ldots\_or \_cdots\_or \_cdots\_or \_cdots\_or \_cdots\_or \_cdots \_else \_ldots \_fi
   \_fi
}
\_def\.ischardef #1\_iftrue {\_ea\.ischardefA\_meaning#1\_fin}
\_def\.ischardefA #1#2#3#4#5#6\_fin {\_def\.tmpa{#1#2#3#4#5}\_ifx\.tmpa\.stringchar}
\_edef\.stringchar{\_string\char}
\_def\.readclass#1{\_ea\.readclassA\_meaning#1\_fin}
\_def\.readclassA#1"#2"#3\_fin{#2}

\_nspublic \smartdots ;
```

The macro `\.interval` reads following tokens until the `)` or `>` is found in the input queue and replaces `<` to `\langle` and `>` to rangle. The reading and replacing process saves the tokens to the `\intevalL` macro only and finally this macro is launched.

math.opm

```
\_def\.interval{\_def\.intervalL{}\.intervalA}
\_def\.intervalA{\_nospacefuturelet\.next\.intervalB}
\_def\.intervalB{\_ifx\.next\_bgroup \_ea\.intervalC \_else \_ea\.intervalD \_fi}
\_def\.intervalC#1{\_addto\.intervalL{{#1}}\.intervalA}
\_def\.intervalD#1{\_casesof #1
   < {\_addto\.intervalL{\langle}\.intervalA}
   > {\_addto\.intervalL{\rangle}\.intervalL}
   ) {\_addto\.intervalL{)}\.intervalL}
   \_finc {\_addto\.intervalL{#1}\.intervalA}%
}
```

`\smartvert` sets `|` as math-active character and declares it equal to `\.autovert` macro. This macro checks two variants: there is single `|` or there is double `||`. It runs `\.autovertA` or `\.autoVertA`. These

macros find the closing | or || and use | or ‖ in context of \left, \right. The \singlevert is declared here if a user want to use a single vertical bar.

math.opm

```
\_newpublic\_mathchardef \singlevert=\_mathcode`|
\_def\.autovert {\_isnextchar|{\.autoVertA}{\.autovertA}}
\_def\.autovertA #1|{\_mathopen{}\_mathclose{\_left|#1\_right|}}
\_def\.autoVertA|#1||{\_mathopen{}\_mathclose{\_left‖#1\_right‖}}
\_bgroup \_lccode`\~=`\| \_lowercase{\_egroup
   \_def\.smartvert{\_let~=\.autovert \_mathcode`|="8000 }}

\_nspublic \smartvert ;
```

\rmsbsp activates ^ and _ so they check the next character. If it is [then \mathbox is used for subscript or superscript, else normal behavior of subscript or superscript is kept.

math.opm

```
\_def\.rmsbsp{%
   \_adef ^{\_isnextchar[{\.rmsp}{\_sp}}
   \_adef _{\_isnextchar[{\.rmsb}{\_sb}}
   \_def\.rmsp[##1]{\_sp{\_mathbox{##1}}}
   \_def\.rmsb[##1]{\_sb{\_mathbox{##1}}}
   \_catcode `\^=12   \_catcode`\_=11
   \_mathcode`_="8000 \_mathcode`^="8000
}
\_nspublic \rmsbsp ;
```

The control sequences \N, \Z, \Q, \R, \C \sgn, \argmin, \argmax, \grad, \rank, \tr, \diag, \Span, \Rng, \Null, \Ker, \Res, \tg, \cotg, \arctg, \arccotg, \frac, \dfrac, \tfrac, \. are defined directly in the user space by _newpublic. The \.pdef is a shortcut for \protected\def.

math.opm

```
\_def\.pdef{\_protected\_def}

\_newpublic\.pdef \N {{\_bbchar N}}
\_newpublic\.pdef \Z {{\_bbchar Z}}
\_newpublic\.pdef \Q {{\_bbchar Q}}
\_newpublic\.pdef \R {{\_bbchar R}}
\_newpublic\.pdef \C {{\_bbchar C}}

\_newpublic\.pdef \sgn     {\_mathop{\_rm sgn}\_nolimits}
\_newpublic\.pdef \argmin  {\_mathop{\_rm argmin}}
\_newpublic\.pdef \argmax  {\_mathop{\_rm argmax}}
\_newpublic\.pdef \grad    {\_mathop{\_rm grad}\_nolimits}
\_newpublic\.pdef \rank    {\_mathop{\_rm rank}\_nolimits}
\_newpublic\.pdef \tr      {\_mathop{\_rm tr}\_nolimits}
\_newpublic\.pdef \diag    {\_mathop{\_rm diag}\_nolimits}
\_newpublic\.pdef \Span    {\_mathop{\_rm Span}\_nolimits}
\_newpublic\.pdef \Rng     {\_mathop{\_rm Rng}\_nolimits}
\_newpublic\.pdef \Null    {\_mathop{\_rm Null}\_nolimits}
\_newpublic\.pdef \Ker     {\_mathop{\_rm Ker}\_nolimits}
\_newpublic\.pdef \Res     {\_mathop{\_rm Res}\_nolimits}
\_newpublic\.pdef \tg      {\_mathop{\_rm tg}\_nolimits}
\_newpublic\.pdef \cotg    {\_mathop{\_rm cotg}\_nolimits}
\_newpublic\.pdef \arctg   {\_mathop{\_rm arctg}\_nolimits}
\_newpublic\.pdef \arccotg {\_mathop{\_rm arccotg}\_nolimits}

\_newpublic\.pdef \frac  #1#2{{{#1}\_over#2}}
\_newpublic\.pdef \dfrac #1#2{{\_displaystyle{{#1}\_over#2}}}
\_newpublic\.pdef \tfrac #1#2{{\_textstyle{{#1}\_over#2}}}

\_newpublic\.pdef \.#1{{\_bf#1}}
```

\eqsystem{⟨*equations*⟩} saves its parameter to \.tmpb and does a collection of _replstrings. It replaces all spaces by & (but ignores the optional first and last space), it removes spaces before \cr, it precedes \mathord before all - (but not if the minus is alone in the column) and runs \halign. The \baselineskip is enlarged by \openup\eqskip. Each item adds 0.5\eqsep around it and \eqfil to the left side and \hfil to the right side.

math.opm

```
\_protected \_optdef\.eqsystem[]#1{\_vcenter{%
   \_def\.tmpb{\.bb#1\.bb}\_replstring\.tmpb{ }{&}%
   \_replstring\.tmpb{\.bb&}{}\_replstring\.tmpb{&\.bb}{}\_replstring\.tmpb{&\cr}{\cr}%
```

```
    \_replstring\.tmpb{-}{\_mathord-}\_replstring\.tmpb{&\_mathord-&}{&-&}%
    \_let ~=\_relax
    \_the\_opt \_relax \_openup\.eqskip
    \_halign{&\_the\.eqfil\_kern.5\.eqsep$\_displaystyle{{}##{}}$\_kern.5\.eqsep\_hfil\_cr
             \.tmpb\_crcr}%
}}
\_let\.bb=\_empty
\_newdimen\.eqskip
\_newdimen\.eqsep
\_newtoks\.eqfil   \.eqfil={\_hfill}
\_nspublic \eqsystem \eqskip \eqsep \eqfil ;
```

The \toright and \toleft macros are based on the \setpos and \posx macros. The printing point is shifted by \posx to the left (i.e. to the left boundary of the sheet) and then it is shifted to the desired space by \kern\hoffset+\hsize. This idea is moved from OpTEX trick 0028.

math.opm

```
\_newcount \.tomarginno
\_def\.toright #1{\_incr\.tomarginno {\_setpos[_math_tr:\_the\.tomarginno]%
    \_rlap{\_kern-\_posx[_math_tr:\_the\.tomarginno]\_kern\_hoffset\_kern\_hsize\_llap{#1}}}}
\_def\.toleft #1{\_incr\.tomarginno {\_setpos[_math_tr:\_the\.tomarginno]%
    \_rlap{\_kern-\_posx[_math_tr:\_the\.tomarginno]\_kern\_hoffset\_rlap{#1}}}}

\_nspublic \toright \toleft ;
```

The \subeqmark works because the internal _thednum is re-defined. The \.dnumpost is added here. If you want to use another format for _thednum then you have to add the \.dnumpost to it too.

math.opm

```
\_def \_thednum    {(\_the\_dnum \.dnumpost)}
\_def\.dnumpost{}
\_def\.subeqmark #1{\_def\.dnumpost{#1}\_lowercase{\_ifx a#1}\_else \_decr\_dnum\_fi \_eqmark}

\_nspublic \subeqmark ;
```

\bfserif re-defines internal OpTEX _mabf and _mabi macros.

math.opm

```
\_def\.bfserif{\.unionly\bfserif{%
    \_protected\_def\_mabf {\_inmath{\_bfvariables\_bfgreek\_bfGreek\_bfdigits}}%
    \_protected\_def\_mabi {\_inmath{\_bivariables\_bigreek\_bfGreek\_bfdigits}}%
}}

\_nspublic \bfserif ;
```

\flexipa (or \flexiblepartial) runs \partialsymbolvars {it}{rm}{bfsans}{it}{bfitsans}. The \partialsymbolvars macro adds two tokens \.partialvar \m⟨*var*⟩partial to _mit, _marm, _mabf, _mait, _mabi macros. The \.partialvar macro sets appropriate _Umathcode of the \.partialchar to the code given by the parameter \m⟨*var*⟩partial. Five macros \mbfpartial, \mitpartial, \mbfitpartial, \mbfsanspartial, \mbfitsanspartial are declared in the macro file unimath-table.opm. The math character \mrmpartial (for upright variant) is declared here.

math.opm

```
\_chardef\.partialchar="2202
\_Umathchardef\.mrmpartial=0 1 \.partialchar
\_def\.partialvar #1{\_Umathcode \.partialchar 0 1
                                 \_ifx#1\.mrmpartial \.partialchar \_else\_ea`#1 \_fi
}
\_def\.inadd#1#2{\_ea\.inaddA#1{#2}#1}
\_def\.inaddA\_inmath#1#2#3{\_protected\_def#3{\_inmath{#1#2}}}

\_def\.partialsymbolvars #1#2#3#4#5{%
    \_ifx\_ncharrmA\_undefined \_opwarning{\_string\flexipa: Unicode math must be loaded first}%
    \_else
      \_def\.tmp{\_ea\_addto \_ea\_mit  \_ea {\_ea\.partialvar \_csname m#1partial\_endcsname}}%
      \_ifx\_mit\mit \.tmp \_let\mit=\_mit \_else \.tmp \_fi
      \_ea\.inadd \_ea\_marm \_ea {\_ea\.partialvar \_csname m#2partial\_endcsname}%
      \_ea\.inadd \_ea\_mabf \_ea {\_ea\.partialvar \_csname m#3partial\_endcsname}%
      \_ea\.inadd \_ea\_mait \_ea {\_ea\.partialvar \_csname m#4partial\_endcsname}%
      \_ea\.inadd \_ea\_mabi \_ea {\_ea\.partialvar \_csname m#5partial\_endcsname}%
      \_mit
    \_fi
}
```

```
\_def\.flexipa{\.unionly\flexipa{\.partialsymbolvars {it}{rm}{bfsans}{it}{bfitsans}}}
\_newpublic \_let \flexiblepartial=\.flexipa
\_nspublic \flexipa \partialsymbolvars \mrmpartial ;
```

The options `mstyle`, resp. `bstyle` run `\.mstyle`, resp. `\.bstyle` and these macros set required shapes of math variables. This can be done only when Unicode-math is loaded already. This is a reason why `\.unionly{`⟨*text*⟩`}{`⟨*code*⟩`}` is used: it runs ⟨*code*⟩ only when Unicode-math is loaded, otherwise it prints a warning.

math.opm

```
\_def\.mstyle #1{\.unionly{mstyle}{\_lowercase{\_cs{_math_mstyle_#1}}}}
\_def\.bstyle #1{\.unionly{bstyle}{\_lowercase{\_cs{_math_bstyle_#1}}}}

\_def\.mstyle_tex {%
   \_protected\_def\_mit {\_itvariables \_rmdigits \_itgreek \_rmGreek}\_mit
}
\_def\.mstyle_iso {%
   \_protected\_def\_mit {\_itvariables \_rmdigits \_itgreek \_itGreek}\_mit
}
\_def\.mstyle_french {%
   \_protected\_def\_mit {\_umathrange{A-Z}71\_ncharrmA \_umathrange{a-z}71\_ncharita
                          \_rmdigits \_rmgreek \_rmGreek}%
   \_mit
}
\_def\.mstyle_upright {%
   \_protected\_def\_mit {\_rmvariables \_rmdigits \_rmgreek \_rmGreek}\_mit
}
\_def\.bstyle_tex {%
   \_protected\_def\_mabf {\_inmath{\_bfvariables\_bigreek\_bfGreek\_bfdigits}}%
   \_protected\_def\_mabi {\_inmath{\_bivariables\_bigreek\_bfGreek\_bidigits}}%
}
\_def\.bstyle_optex {%
   \_protected\_def\_mabf {\_inmath{\_bsansvariables \_bsansgreek \_bsansGreek \_bsansdigits}}%
   \_protected\_def\_mabi {\_inmath{\_bisansvariables \_bisansgreek \_bsansGreek \_bsansdigits}}%
}
\_def\.bstyle_iso {%
   \_protected\_def\_mabf {\_inmath{\_bivariables\_bigreek\_biGreek\_bfdigits}}%
   \_protected\_def\_mabi {\_inmath{\_bivariables\_bigreek\_bfGreek\_bidigits}}%
}
\_def\.bstyle_upright {%
   \_protected\_def\_mabf {\_inmath{\_bfvariables\_bfgreek\_bfGreek\_bfdigits}}%
   \_protected\_def\_mabi {\_inmath{\_bivariables\_bigreek\_biGreek\_bidigits}}%
}
\_def\.unionly #1{\_ifx\_rmvariables\_undefined
   \_opwarning{pkg:math: \_string#1 ignored: Unicode-math must be loaded}%
   \_ea\_ignoreit \_else \_ea\_useit \_fi
}
```

`\rmchars{`⟨*list*⟩`}` is implemented using `\foreach`. The list is expanded first because we want to expand control sequences like `\alpha` to a real character α.
`\vargreek{`⟨*list*⟩`}` is implemented using `\foreach`. The parameter is not expanded because we want to keep control sequences like `\alpha` unchanged.

math.opm

```
\_def\.rmchars#1{\.unionly\rmchars{\_ea\_foreach\_expanded{#1}\_do{\_ifx,##1\_else\.rmchar##1\_fi}}}
\_def\.rmchar#1{\_Umathcode`#1=0 1 `#1 }

\_def\.vargreek#1{\_foreach#1\_do{\_ifx,##1\_else \.vargreekchar##1\_fi}}
\_def\.vargreekchar#1{%
   \_ifcsname var\_csstring#1\_endcsname \_slet{\_csstring#1}{var\_csstring#1}%
   \_else \_opwarning{\_string\vargreek: the \_bslash var\_csstring#1\_space doesn't exists}%
   \_fi
}
\_nspublic \rmchars \vargreek ;
```

`\textvariables`, `\textdigits`, `\textmoremath` initialize new two families 5, 6 using `\.textmathini` and sets `\mathcodes` of given characters to these families. Moreover, `\textvariables` adds `\fam` register setting to `\rm` and `\it` selectors and re-set Greek variables to use only math font (because we are not sure if Greek letters are in the current text fonts).

`\.dotext{`⟨*list of words*⟩`}` runs `\.text`⟨*word*⟩ for each ⟨*word*⟩ in the list. It is used when the option `text={`⟨*list of words*⟩`}` is used.

math.opm

```
\_def\.textmathini{%
   \_fontdef\.mathrm{\_rm}\_fontdef\.mathit{\_it}%
   \_fontdef\.mathbf{\_bf}\_fontdef\.mathbi{\_bi}%
   \_addto\_normalmath{%
      \_setmathfamily    5 \.mathrm
      \_setmathfamily    6 \.mathit
   }%
   \_addto\_boldmath{%
      \_setmathfamily    5 \.mathbf
      \_setmathfamily    6 \.mathbi
   }%
   \_normalmath
   \_let\.textmathini=\_relax
}
\_def\.textvariables {\.unionly\textvariables {%
   \.textmathini \_mathcodes 6 {7{\_Urange a-z \_Urange A-Z}}%
   \_addto\_marm {\_fam5 }\_addto\_mait{\_fam6 }%
   \_protected\_def\_itgreek    {\_umathrangegreek01\_greekita}%
   \_protected\_def\_rmgreek    {\_umathrangegreek01\_greekrma}%
   \_protected\_def\_itGreek    {\_umathrangeGREEK01\_greekitA}
   \_protected\_def\_rmGreek    {\_umathrangeGREEK01\_greekrmA}
   \_itgreek \_rmGreek
}}
\_def\.textdigits    {\.unionly\textdigits{\.textmathini \_mathcodes 5 {7{\_Urange 0-9}}}}
\_def\.textmoremath  {\.unionly\textmoremath{%
   \.textmathini
   \_mathcodes 5 {5{!?} 2{*+-} 3{=<>} 6{,:;} 0{./|} 4{([\{} 5{\})]}}%
   \_Umathcode `- = 2 5 "2212  % hyphen behaves like minus in math mode
}}
\_def\.dotext#1{\_foreach #1 \_do
   ##1 {\_trycs{_math_text##1}{\_opwarning{text option: "##1" unknown}}}}

\_nspublic \textvariables \textdigits \textmoremath ;
```

`\replacemissingchars`⟨*family*⟩ defines `\UnicodeMathSymbol` and reads `unimath-table.opm`, i.e. it does for each math character following if the character is missing in main math font and if it is present in added font and if it is not already replaced character then apply new math code or `\Umathaccent` definition. Its name is added to `\.alist` or `\.clist`. The new codes are declared by `\matchars`⟨*family*⟩`{`⟨*expanded*⟩`\clist}`. The `\.rlist` is the list of characters already replaced. They are not replaced again if a new `\replacemissingchars` is used.

math.opm

```
\_def\.rlist{\sqrt\cuberoot\fourthroot} % they cannot be replaced by \mathchars
\_def\.replacemissingchars#1{\.unionly\replacemissingchars{%
   \_def\.alist{}\_def\.clist{}
   \_def\UnicodeMathSymbol##1##2##3##4{%
      \_iffontchar\_textfont1##1 \_else     % not in main math font
         \_iffontchar\_textfont#1 ##1       % is presnet in added font
            \_isinlist\.rlist{##2}\_iffalse % not already replaced
               \_ifx##3\_mathaccent
                  \_protected\_def##2{\_Umathaccent fixed 7 #1 ##1 }%
                  \_addto\.alist{##2}%
               \_else
                  \_addto\.clist{##2}%
      \_fi\_fi\_fi\_fi
   }
   \_input unimath-table.opm
   \_wlog{^^J\_string\replacemissingchars: From \_string\fam=\_string#1 is printed now:^^J%
         CHARACTERS: \_unexpanded\_ea{\.clist}^^JACCENTS: \_unexpanded\_ea{\.alist}^^J}%
   \_def\.tmp{\_mathchars #1}\_ea\.tmp\_ea{\.clist}%
   \_ea\_addto \_ea\.rlist \_ea{\.clist}\_ea\_addto \_ea\.rlist \_ea{\.alist}%
   \_def\.alist{}\_def\.clist{}\_let\UnicodeMathSymbol=\_undefined
}}
\_nspublic \replacemissingchars ;
```

`\scriptspaces{`⟨*s-rel*⟩`}{`⟨*s-bin*⟩`}{`⟨*ss-rel*⟩`}{`⟨*ss-bin*⟩`}` sets internal LuaTEX registers represented by appropriate primitives, see section 7.5 in the LuaTEX manual.

math.opm

```
\_def\.scriptspaces #1#2#3#4{%
   \_Umathordrelspacing\_scriptstyle=\.orzeromu{#1}\_relax
   \_Umathrelordspacing\_scriptstyle=\.orzeromu{#1}\_relax
   \_Umathrelopspacing \_scriptstyle=\.orzeromu{#1}\_relax
   \_Umathordrelspacing\_crampedscriptstyle=\.orzeromu{#1}\_relax
   \_Umathrelordspacing\_crampedscriptstyle=\.orzeromu{#1}\_relax
   \_Umathrelopspacing \_crampedscriptstyle=\.orzeromu{#1}\_relax
   \_Umathordbinspacing\_scriptstyle=\.orzeromu{#2}\_relax
   \_Umathbinordspacing\_scriptstyle=\.orzeromu{#2}\_relax
   \_Umathbinopspacing \_scriptstyle=\.orzeromu{#2}\_relax
   \_Umathordbinspacing\_crampedscriptstyle=\.orzeromu{#2}\_relax
   \_Umathbinordspacing\_crampedscriptstyle=\.orzeromu{#2}\_relax
   \_Umathbinopspacing \_crampedscriptstyle=\.orzeromu{#2}\_relax
   \_Umathordrelspacing\_scriptscriptstyle=\.orzeromu{#3}\_relax
   \_Umathrelordspacing\_scriptscriptstyle=\.orzeromu{#3}\_relax
   \_Umathrelopspacing \_scriptscriptstyle=\.orzeromu{#3}\_relax
   \_Umathordrelspacing\_crampedscriptscriptstyle=\.orzeromu{#3}\_relax
   \_Umathrelordspacing\_crampedscriptscriptstyle=\.orzeromu{#3}\_relax
   \_Umathrelopspacing \_crampedscriptscriptstyle=\.orzeromu{#3}\_relax
   \_Umathordbinspacing\_scriptscriptstyle=\.orzeromu{#4}\_relax
   \_Umathbinordspacing\_scriptscriptstyle=\.orzeromu{#4}\_relax
   \_Umathbinopspacing \_scriptscriptstyle=\.orzeromu{#4}\_relax
   \_Umathordbinspacing\_crampedscriptscriptstyle=\.orzeromu{#4}\_relax
   \_Umathbinordspacing\_crampedscriptscriptstyle=\.orzeromu{#4}\_relax
   \_Umathbinopspacing \_crampedscriptscriptstyle=\.orzeromu{#4}\_relax
}
\_def\.orzeromu#1{\_ifx^#1^0mu\_else#1\_fi}

\_nspublic \scriptspaces ;
```

`\mathclap{`⟨*formula*⟩`}`, `\mathrlap{`⟨*formula*⟩`}`, and `\mathllap{`⟨*formula*⟩`}` are based on the OpTEX macros `\mathstyles` and `\currstyle`.

math.opm

```
\_def\.mathclap#1{\_mathstyles{\_hbox to0pt{\_hss$\_currstyle#1$\_hss}}}
\_def\.mathrlap#1{\_mathstyles{\_rlap{$\_currstyle#1$}}}
\_def\.mathllap#1{\_mathstyles{\_llap{$\_currstyle#1$}}}

\_nspublic \mathclap \mathrlap \mathllap ;
```

`\enablefic` enables final italic correction. The relevant lua function is registered to `mlist_to_hlist` callback and `\finalitalcorr` is set to one.

math.opm

```
\_newcount \.finalitalcorr
\_directlua{
   function math.final_ital_corr(head, style)
        if style=="text" and tex.count.\_pkglabel _finalitalcorr>0 then
            for n in node.traverse(head) do
                if n.next == nil and n.id == 29 then % last is glyph
                    local k = font.fonts[n.font].characters[n.char].italic
                    if not(k==nil) and (k>0) then
                        local kn = node.new("kern")
                        kn.kern = k kn.subtype = 3
                        node.insert_after(head, n, kn) % kern node is inserted
                    end
                end
            end
        end
        return head
   end
}
\_def\.enablefic {\_directlua{ % math.final_ital_corr is registered to mlist_to_hlist
   luatexbase.add_to_callback("mlist_to_hlist",
       function(head, style, penalties)
           head = node.mlist_to_hlist(head, style, penalties)
           return math.final_ital_corr(head, style)
       end, "italcorr after math")
   }
   \.finalitalcorr=1
}
```

```
\_nspublic \enablefic \finalitalcorr ;

\_endnamespace
```

Index