The HEP-MATH-FONT package*

Composable math alphabets that follow the text font, and dashed and dotted vector accents

Jan Hajer†

2026/11/13

Abstract

The HEP-MATH-FONT package provides math fonts matching the text fonts of the HEP-FONT package. Its math alphabet commands are composable: bold, upright and the serif, sans-serif and typewriter families can be combined in any order, e.g. `\mathbf{\mathsf{x}}` and `\mathsf{\mathbf{x}}` give the same bold sans-serif x. Math follows the surrounding text, so that math in a sans-serif or bold heading is sans-serif or bold as well. The package also provides vector accents with a dashed and a dotted shaft. With LuaLaTeX and XeLaTeX the package uses the new computer modern (NCM) math fonts, with pdfLaTeX it uses latin modern (LM) and computer modern (CM) fonts combined into virtual fonts.

The package is loaded using `\usepackage{hep-math-font}`. It can be used without the HEP-FONT package, but if both are used the HEP-FONT package has to be loaded first. The main font family of the document has to be chosen before loading this package, so that the operator and symbol fonts match it as well.

1 Math alphabets

The state of a math alphabet consists of three independent parts: the family (serif, sans-serif or typewriter), the weight (medium or bold) and the shape (italic or upright). Each command changes one part of this state and keeps the others. Bold and upright therefore commute with the family commands and with each other, and repeating a command has no further effect. If different families are nested, the innermost family wins. Groups restore the state of the surrounding math. The commands neither change the parsing of fractions, scripts, accents and radicals nor the arguments of macros. They affect Latin and Greek letters, the variant Greek letters, `\partial`, `\nabla`, `\ell`, `\hbar`, the dotless i and j, and digits but not operators, delimiters or accents. The blackboard alphabet of the default `symbols` option is described below. Uniform resource locators (URLs) typeset by the URL package, and therefore also by HYPERREF and BIBLATEX, keep the letters and digits of their text font.

`\mathbf` `\mathup` `\mathrm` `\mathit`

The `\mathbf{`⟨*math*⟩`}` command adds bold and `\mathup{`⟨*math*⟩`}` adds upright, `\mathit{`⟨*math*⟩`}` returns to italic. The `\mathrm` command is a synonym of `\mathup` and therefore preserves the family and the weight. This deliberately differs from the usual meaning of resetting to medium upright serif and makes `\mathrm` composable. The commands are defined when the package is loaded and again at the beginning of the document, so copies such as `\let\mat\mathbf` work both in the preamble and in `\AtBeginDocument`. The same holds for the Greek letter macros, and redefinitions such as `\renewcommand{\epsilon}{\varepsilon}` in the preamble are kept.

*This document corresponds to HEP-MATH-FONT v2.0.

†jan.hajer@tecnico.ulisboa.pt

Command	Family	Added attributes
`\mathbfup`, `\mathbfrm`	kept	bold, upright
`\mathbfsf`	sans-serif	bold
`\mathsfup`, `\mathsftext`	sans-serif	upright
`\mathbfsfup`	sans-serif	bold, upright
`\mathbftt`	typewriter	bold
`\mathttup`	typewriter	upright
`\mathbfttup`	typewriter	bold, upright

Table 1: Combined math alphabet commands.

`\mathserif` `\mathsf` `\mathtt` The `\mathserif{`⟨*math*⟩`}`, `\mathsf{`⟨*math*⟩`}` and `\mathtt{`⟨*math*⟩`}` commands select the serif, sans-serif and typewriter family, respectively. Upright serif math that keeps the surrounding weight is obtained with `\mathserif{\mathup{`⟨*math*⟩`}}`.

`\mathnormal` The `\mathnormal{`⟨*math*⟩`}` command restores the default state of the surrounding text.

`\mathbfup` `\mathbfsf` `\mathsfup` `\mathbfsfup` `\mathbftt` `\mathttup` `\mathbfttup` The combined commands are listed in table 1. `\mathbfrm` is a synonym of `\mathbfup`, and `\mathsftext` is a synonym of `\mathsfup`.

Formulas start in the family and weight of the surrounding text. Hence the commands `\rmfamily`, `\sffamily`, `\ttfamily`, `\bfseries` and `\mdseries` also affect math, e.g. in headings.

The BM package [1] remains available for bold symbols other than letters and digits.

`\bm` `\mathbb` With the default `symbols=true` the blackboard alphabet is composable as well. The `\mathbb{`⟨*math*⟩`}` command adds blackboard as a fourth part of the state, which applies to Latin letters and digits. The family and the weight apply to the blackboard letters, so that `\mathbf{\mathbb{R}}` and `\mathbb{\mathbf{R}}` give the same bold blackboard $\mathbb{R}$, while the shape has no effect. The blackboard alphabet has no typewriter family and uses the serif shapes instead. With the other values of the `symbols` option `\mathbb` is a decorative alphabet.

`\mathcal` With `symbols=true` and `symbols=ams` the calligraphic alphabet is composable in the same way. The `\mathcal{`⟨*math*⟩`}` command adds calligraphic as a fourth part of the state, which applies to Latin capitals. With LuaLaTeX and XeLaTeX the serif and sans-serif calligraphic capitals share their design, but the strokes of the sans-serif capitals have a uniform width. pdfLaTeX uses the computer modern (CM) calligraphic capitals for the serif family as well, but no sans-serif calligraphic font differs enough from them, so the sans-serif capitals are those of Zapf Chancery, scaled to the same height. Hence `\mathsf{\mathcal{L}}` gives the sans-serif $\mathcal{L}$ next to the serif $\mathcal{L}$, and `\mathbf` gives their bold weight. The upright shape selects the upright Euler script [2] in medium and bold weight, so that `\mathup{\mathcal{L}}` gives $\mathcal{L}$, for the serif and sans-serif families alike. The typewriter family uses the sans-serif capitals. Small letters are not affected. With `symbols=minion` `\mathcal` is a decorative alphabet, and with `symbols=false` it keeps its original definition.

`\mathscr` With `symbols=true` and `symbols=ams` the script alphabet is composable as well and applies to Latin capitals. It uses the RSFS capitals [3] with pdfLaTeX and the stylistic set of the new computer modern (NCM) math fonts, which follows them, with LuaLaTeX and XeLaTeX. The serif and sans-serif families and the bold weight apply as for `\mathcal`, but with pdfLaTeX there is no sans-serif script and the bold capitals are synthesised. The upright shape reduces the slant of the capitals by 0.4 as the RSFSO package [4] does, so that `\mathup{\mathscr{L}}` gives $\mathscr{L}$ next to $\mathscr{L}$. Blackboard, calligraphic and script exclude each other, the innermost of them wins. With `symbols=minion` `\mathscr` is a decorative alphabet, and with `symbols=false` it keeps its

Value	pdfLaTeX	LuaLaTeX and XeLaTeX
`true`	American Mathematical Society (AMS) symbols, composable NCM and BBOLDX blackboard, composable calligraphic and script, Euler fraktur	NCM symbols, composable blackboard, calligraphic and script
`ams`	as `true` with decorative AMS blackboard	as `true` with decorative blackboard
`minion`	MnSymbol, decorative double-stroke blackboard and RSFS script	not supported
`false`	no additional symbol fonts	no script stylistic set

Table 2: Values of the `symbols` option, the default is `true`.

original definition.

`\mathfrak` The decorative alphabet `\mathfrak` keeps its own style. The `\sym`⟨*name*⟩ commands of other packages are not affected by the composable commands.

In bookmarks of the HYPERREF package [5] the alphabet commands and `\bm` keep their argument, so that only the math shifts are removed. The Greek letters, `\partial`, `\nabla`, `\ell`, `\hbar` and the dotless i and j become Unicode characters, where the Greek letters follow their meaning in text.

2 Vector accents

`\dashedvec` `\dottedvec` The `\dashedvec{`⟨*math*⟩`}` and `\dottedvec{`⟨*math*⟩`}` accents are vector arrows with a dashed and a dotted shaft, so that `\dashedvec{x}` and `\dottedvec{x}` give $\vec{x}$ and $\vec{x}$ next to the $\vec{x}$ of `\vec{x}`. Their glyphs are copies of the glyphs of `\vec`, in which only the shaft is cut into dashes or dots. The arrowhead, the size and the placement are therefore those of `\vec`: the accents follow the slant of italic letters, shrink in sub- and superscripts, are bold in the bold math version and can be combined with other accents. Like `\vec` they do not follow the math alphabet commands. The dashed shaft consists of two dashes and the dotted shaft of three dots, but the short and thick shafts of the bold sans-serif fonts only provide space for one dash and two dots. In bookmarks of the HYPERREF package they keep their argument.

3 Options

`symbols` The `symbols=`⟨*value*⟩ option selects the symbol fonts, see table 2. The AMS symbols [6] and MnSymbol [7] are loaded from their packages. The AMS blackboard only contains capitals and the character k. The NCM blackboard of the MATHALPHA package and the BBOLDX blackboard contain capitals, small letters and digits in medium and bold weight. The RSFS script [3] only contains capitals and the double-stroke blackboard [8] contains capitals and the characters h, k and 1. With pdfLaTeX the decorative alphabets are packed into one virtual font family, the four blackboard alphabets into another one and the script and upright calligraphic alphabets into a third one. In this way the `true`, `ams` and `minion` setups use 15, 14 and 15 of the 16 math families of pdfTeX, but additional alphabet packages can still exhaust them. The BM package does not reserve additional families.

`mathsf` With the default `mathsf=sans` the `\mathsf` command selects sans-serif in serif and sans-serif documents. With `mathsf=contrast` it selects the family opposite to the main family of the

Family	pdfLaTeX	LuaLaTeX and XeLaTeX
Serif	LM math and text, CM Unicode	NCM math
Sans-serif	CM sans math, LM sans text and CM Unicode	NCM sans math and text
Typewriter	LM typewriter, CB Greek and CM Unicode	LM typewriter text and NCM mono text
Blackboard	NCM and BBOLDX	NCM math
Calligraphic	LM math, Zapf Chancery and Euler script	NCM math and sans math, Euler math
Script	RSFS	NCM math and sans math
Vector accents	LM math italic or CM sans math italic	NCM math or sans math

Table 3: Fonts of the math alphabets.

document, which simplifies moving math between serif and sans-serif documents. This option also affects the combined commands and repeating `\mathsf` still has no further effect.

`synthetic` With pdfLaTeX some shapes exist in no font and are synthesised by slanting a related shape by the slant of the math italic font, see section 4. With `synthetic=false` the related shapes are used unchanged, that is, the italic shapes instead of the upright ones and vice versa. The bold sans-serif calligraphic and the bold script capitals are synthesised by overprinting the medium ones, and with `synthetic=false` the medium capitals are used. The upright script capitals are synthesised with all engines, and with `synthetic=false` the italic ones are used. Otherwise the option has no effect with LuaLaTeX and XeLaTeX.

`proportional` By default the typewriter alphabets set Latin letters and digits in the proportional latin modern (LM) typewriter, so that names of several letters do not fall apart. With `proportional=false` they use the monospaced typewriter fonts. The typewriter Greek letters are monospaced in both cases.

4 Fonts

The fonts depend on the engine, see table 3. With LuaLaTeX and XeLaTeX the package uses the UNICODE-MATH package [9], and every combination of family, weight and shape uses a font face designed for it, without synthetic bold or slanting, except for the upright script. The typewriter faces are text fonts without a math table, so that some specialised math metrics are not available. With pdfLaTeX the upright and typewriter Greek letters come from the CB Greek and CM Unicode fonts, which share their design.

Some combinations are not available in any font and fall back to related shapes:

- The typewriter `\partial`, `\nabla` and `\ell` use the serif shapes.
- Digits are always upright and lining, but bold and the family apply.
- `\ell` has the same shape in upright and italic.
- The proportional LM typewriter has no italic, so its oblique shapes are used, and there is no proportional typewriter Greek.

With LuaLaTeX and XeLaTeX there are two more fallbacks:

- There is no bold sans-serif blackboard and no bold sans-serif `\ell`, the medium blackboard and the bold serif `\ell` are used instead.
- The medium upright typewriter `\vartheta` uses the serif shape, as the mono face draws it as `\theta`.
- The upright script capitals are slanted back by LUAOTFLOAD or XeTeX, which keep the accent positions of the italic capitals.

With pdfLaTeX there are other fallbacks:

- The typewriter `\vartheta`, `\phi`, `\varrho` and `\varpi` use the serif shapes.
- There is no bold `\varkappa`, the medium shape is used instead.
- There is no bold Zapf Chancery and no bold RSFS, the bold sans-serif calligraphic and the bold script capitals are synthesised, see the `synthetic` option.
- There is no sans-serif script, the sans-serif and typewriter families use the RSFS capitals.
- `\varTheta` uses the shape of `\Theta`.
- The upright `\partial`, `\varpi` and `\varkappa`, the upright sans-serif `\vartheta`, `\phi` and `\varrho`, the italic `\nabla` and the italic typewriter `\varepsilon` are synthesised, see the `synthetic` option.

5 Virtual fonts

For pdfLaTeX the package uses six virtual font families, one for each family and weight, and two more for the proportional typewriter, at the sizes 5 pt, 7 pt and 10 pt. Each font contains the italic shapes in its lower and the upright shapes in its upper half. The virtual fonts refer to installed fonts and to base fonts derived from them, and copy their metrics, including italic corrections and the kerns for accents. The base fonts whose names start with `hep-` contain CM Unicode and LM text fonts in an encoding of their Greek letters and h-bar. The base fonts whose names start with `hepslant-` are slanted copies of installed fonts. The map file hep-math-font.map declares the base fonts, and the map file hep-math-font-fallback.map declares the slanted copies without slant for `synthetic=false`. The package loads the map file itself. The `hepbb` virtual fonts contain the four blackboard alphabets of `symbols=true`. Two `hepdecor` virtual fonts contain the decorative alphabets and the four calligraphic alphabets in medium and bold weight, and four more are used for `symbols=minion`. The `hepscript` virtual fonts contain the italic and upright script and the upright calligraphic alphabets in medium and bold weight.

The dashed and dotted vector accents are part of the TrueType font hepaccent.ttf, which is derived from the vector accents of the LM, CM sans-serif and NCM math fonts. pdfLaTeX uses it through the TeX font metric (TFM) file hepaccent.tfm and the encoding file hepaccent.enc, and the `hepalpha` virtual fonts contain the accents after the italic shapes. LuaLaTeX and XeLaTeX use the font directly, so that it is the only generated file they need.

The TFM and virtual font (VF) files are generated with the script hep-math-font-vf.py, which is extracted together with the package.

```
tex hep-math-font.ins
python3 hep-math-font-vf.py
```

The script reads the glyph table from hep-math-font.sty in the current directory and writes the TFM and VF files, the accent font, the encoding files hep-math-font.enc and hepaccent.enc

and the two map files there. It requires the fontTools library, the KPSEWHICH, PLTOTF and VPTOVF programs and the fonts listed above.

The script also writes the file hep-math-font-vf.def, which records the version of the package file the fonts were generated from. For a local installation all generated files have to be placed next to the document. In a TDS tree hep-math-font-vf.def belongs next to the package file into tex/latex/hep-math-font, the TFM files into fonts/tfm/public/hep-math-font, the VF files into fonts/vf/public/hep-math-font, the accent font into fonts/truetype/public/hep-math-font, the encoding files into fonts/enc/dvips/hep-math-font and the map files into fonts/map/pdftex/hep-math-font.

Without the generated fonts the package still works, with fewer features. If pdfLaTeX does not find hep-math-font-vf.def, the package warns and keeps the standard math alphabets, which use the LM fonts of the main family. They are not composable, the combined commands select the nearest standard alphabet, the decorative alphabets are the AMS blackboard, Euler fraktur and RSFS, and the Greek letters are not affected by the alphabet commands. If the file belongs to another version of the package file, the virtual fonts may contain other glyphs than the package expects. This is an error, after which the package also keeps the standard math alphabets. If the accent font is missing, the vector accents fall back to `\vec` with a warning. The math fonts, operators and symbols of all engines are not affected.

A Implementation

<*package>

A.1 Options

Distinguish the engines using the IFTEX package [10] and define a `hepmathfont` namespace for the options using the KVOPTIONS package [11].

```
\RequirePackage{iftex}
\RequirePackage{kvoptions}
\SetupKeyvalOptions{
  family=hepmathfont,
  prefix=hepmathfont@
}
```

`symbols` Define the `symbols` option selecting the symbol fonts.

```
\DeclareStringOption[true]{symbols}
```

`mathsf` Define the `mathsf` option selecting the family of `\mathsf`.

```
\DeclareStringOption[sans]{mathsf}
```

`synthetic` Define the `synthetic` option, which selects the slanted shapes of the pdfTeX backend.

```
\DeclareBoolOption[true]{synthetic}
```

`proportional` Define the `proportional` option, which selects the proportional typewriter Latin letters and digits.

```
\DeclareBoolOption[true]{proportional}
```

Process the options.

```
\ProcessKeyvalOptions*
```

Load the AMSMATH package [12] and reject unknown values of the `mathsf` and `symbols` options.

```
\RequirePackage{amsmath}
\ExplSyntaxOn
\exp_args:NnV \clist_if_in:nnF {sans, contrast} \hepmathfont@mathsf {
  \PackageError{hep-math-font}
    {Unknown~mathsf~option~'\hepmathfont@mathsf'}
    {Use~mathsf=sans~or~contrast.}
}
\exp_args:NnV \clist_if_in:nnF {true, false, ams, minion}
  \hepmathfont@symbols {
  \PackageError{hep-math-font}
    {Unknown~symbols~option~'\hepmathfont@symbols'}
    {Use~symbols=true,~false,~ams,~or~minion.}
}
```

`\ifhepmathfont@vf` `\ifhepmathfont@accents` The composable alphabets of pdfTEX need the generated virtual fonts and the vector accents of all engines the generated accent font. Both flags are cleared below if these fonts are not available, see section A.4.1.

```
\newif\ifhepmathfont@vf
\hepmathfont@vftrue
\newif\ifhepmathfont@accents
\hepmathfont@accentstrue
```

A.2 State

`\l__hep_bold_bool` `\l__hep_up_bool` `\l__hep_family_tl` `\l__hep_bb_bool` `\l__hep_cal_bool` `\l__hep_scr_bool` The state of the math alphabets consists of the weight, the shape and the family `serif`, `sans` or `mono`. For `symbols=true` the blackboard flag is a fourth part, and for `symbols=true` and `ams` the calligraphic and script flags as well. At most one of the three flags is set.

```
\bool_new:N \l__hep_bold_bool
\bool_new:N \l__hep_up_bool
\tl_new:N \l__hep_family_tl
\bool_new:N \l__hep_bb_bool
\bool_new:N \l__hep_cal_bool
\bool_new:N \l__hep_scr_bool
```

`__hep_main_family_tl` `__hep_text_family_tl` `__hep_text_bold_bool` The main family of the document and the family and weight of the surrounding text are used to initialise each formula.

```
\tl_new:N \g__hep_main_family_tl
\tl_new:N \l__hep_text_family_tl
\bool_new:N \l__hep_text_bold_bool
```

`_selected_family_int` `\l__hep_bank_int` `\l__hep_offset_int` `\l__hep_style_int` The backends translate the state into the selected math family, and for pdfTEX into the virtual font bank and the offset of its upright half, and for the Unicode engines into the style of the Unicode alphabet.

```
\int_new:N \l__hep_selected_family_int
\int_new:N \l__hep_bank_int
\int_new:N \l__hep_offset_int
\int_new:N \l__hep_style_int
```

`l__hep_bb_family_int` `l__hep_bb_offset_int` `__hep_cal_family_int` `__hep_cal_offset_int`

The blackboard alphabets and the calligraphic and script alphabets have their own families and offsets, so that the Greek letter macros keep using the selected family.

```
\int_new:N \l__hep_bb_family_int
\int_new:N \l__hep_bb_offset_int
\int_new:N \l__hep_cal_family_int
\int_new:N \l__hep_cal_offset_int
```

`__hep_glyph_setup_tl` `g__hep_math_codes_tl` `\g__hep_bb_setup_tl` `\g__hep_cal_setup_tl` `_hep_native_codes_tl` `__hep_greek_setup_tl`

Collect the code which maps the characters to the current state, the characters it maps, the code which maps them to the blackboard alphabets and to the calligraphic and script alphabets, the code which restores their original math codes, and the definitions of the Greek letter macros.

```
\tl_new:N \g__hep_glyph_setup_tl
\tl_new:N \g__hep_math_codes_tl
\tl_new:N \g__hep_bb_setup_tl
\tl_new:N \g__hep_cal_setup_tl
\tl_new:N \g__hep_native_codes_tl
\tl_new:N \g__hep_greek_setup_tl
```

`_math_detect_family:` Determine the main family of the document from `\familydefault` and the `sans` option of the HEP-FONT package. It is determined again at the beginning of the document.

```
\cs_new_protected:Npn \hep_math_detect_family: {
  \tl_gset:Nn \g__hep_main_family_tl {serif}
  \str_if_eq:eeT {\familydefault}{\sfdefault}
    {\tl_gset:Nn \g__hep_main_family_tl {sans}}
  \@ifpackageloaded{hep-font}
    {\ifhepfont@sans\tl_gset:Nn \g__hep_main_family_tl {sans}\fi}{}
}
\hep_math_detect_family:
\ExplSyntaxOff
```

A.3 Unicode engines

Load the UNICODE-MATH package [9] with ISO math style and let the math alphabet commands select symbols from the main math font.

```
\iftutex
  \RequirePackage{unicode-math}
  \unimathsetup{
    math-style=ISO,
    bold-style=ISO,
    sans-style=italic,
    partial=italic,
    nabla=italic,
    mathrm=sym,
    mathup=sym,
```

```
    mathit=sym,
    mathsf=sym,
    mathbf=sym,
    mathtt=sym
  }
```

Use the serif or sans-serif NCM math font matching the main family, and the bold math font for the bold math version.

```
  \ExplSyntaxOn
  \str_if_eq:VnTF \g__hep_main_family_tl {sans}
    {\setmathfont{NewCMSansMath-Regular.otf}[Numbers=Lining]}
    {\setmathfont{NewCMMath-Regular.otf}[Numbers=Lining]}
  \ExplSyntaxOff
  \setmathfont{NewCMMath-Bold.otf}[version=bold,Numbers=Lining]
```

Declare symbol fonts for the serif and sans-serif math fonts. Each contains the italic, upright, bold and bold italic Unicode alphabets. Unicode has no bold blackboard alphabet and no bold dotless letters, h-bar and ell, so these are taken from the bold serif math font.

```
  \newfontfamily\hep@serifmath{NewCMMath-Regular.otf}[
    NFSSFamily=hep-serif,
    Script=Math,
    Numbers=Lining,
    BoldFont=NewCMMath-Bold.otf
  ]
  \newfontfamily\hep@sansmath{NewCMSansMath-Regular.otf}[
    NFSSFamily=hep-sans,
    Script=Math,
    Numbers=Lining
  ]
  \DeclareSymbolFont{hepserif}{TU}{hep-serif}{m}{n}
  \DeclareSymbolFont{hepserifbf}{TU}{hep-serif}{b}{n}
  \DeclareSymbolFont{hepsans}{TU}{hep-sans}{m}{n}
```

There is no bold sans-serif math font, so the bold sans-serif letters without bold Unicode characters are taken from the bold NCM sans text faces.

```
  \newfontfamily\hep@sanstext{NewCMSans10-Regular.otf}[
    NFSSFamily=hep-sanstext,
    ItalicFont=NewCMSans10-Oblique.otf,
    BoldFont=NewCMSans10-Bold.otf,
    BoldItalicFont=NewCMSans10-BoldOblique.otf
  ]
  \DeclareSymbolFont{hepsansbfup}{TU}{hep-sanstext}{b}{n}
  \DeclareSymbolFont{hepsansbfit}{TU}{hep-sanstext}{b}{it}
```

The NCM mono faces are text fonts, so declare a symbol font for each of the four faces.

```
  \newfontfamily\hep@monomath{NewCMMono10-Regular.otf}[
    NFSSFamily=hep-mono,
    Numbers=Lining,
    ItalicFont=NewCMMono10-Italic.otf,
```

```
    BoldFont=NewCMMono10-Bold.otf,
    BoldItalicFont=NewCMMono10-BoldOblique.otf
  ]
  \DeclareSymbolFont{hepmonoit}{TU}{hep-mono}{m}{it}
  \DeclareSymbolFont{hepmonoup}{TU}{hep-mono}{m}{n}
  \DeclareSymbolFont{hepmonobfit}{TU}{hep-mono}{b}{it}
  \DeclareSymbolFont{hepmonobfup}{TU}{hep-mono}{b}{n}
```

`\hep@propfont` With the `proportional` option the Latin letters and digits come from the LM proportional typewriter faces, whose bold face belongs to the light family like the bold LM typewriter. They have no italic, so their oblique faces are used instead. The LaTeX kernel declares at most 16 symbol fonts, although the Unicode engines provide 256 math families. The package already uses most of them, so the four faces are allocated past this limit at the beginning of the document, after the symbol fonts of other packages.

```
\ifhepmathfont@proportional
  \newfontfamily\hep@propmath{lmmonoprop10-regular.otf}[
    NFSSFamily=hep-prop,
    ItalicFont=lmmonoprop10-oblique.otf,
    BoldFont=lmmonoproplt10-bold.otf,
    BoldItalicFont=lmmonoproplt10-boldoblique.otf
  ]
  \def\hep@propfont#1#2#3{%
    \expandafter\new@mathgroup\csname symhepprop#1\endcsname
    \expandafter\new@symbolfont\csname symhepprop#1\endcsname
      {TU}{hep-prop}{#2}{#3}}
  \AtBeginDocument{%
    \hep@propfont{it}{m}{it}\hep@propfont{up}{m}{n}%
    \hep@propfont{bfit}{b}{it}\hep@propfont{bfup}{b}{n}}
\fi
```

The MnSymbol fonts are not available as Unicode math fonts. Unless the `symbols` option is `false` use the stylistic set of the NCM math fonts, which provides a distinct script alphabet.

```
\ExplSyntaxOn
\str_if_eq:VnT \hepmathfont@symbols {minion} {
  \PackageError{hep-math-font}
    {symbols=minion~is~not~supported~by~Unicode~engines}
    {Use~symbols=true~or~ams~with~New~Computer~Modern.}
}
\str_if_eq:VnF \hepmathfont@symbols {false} {
  \str_if_eq:VnTF \g__hep_main_family_tl {sans} {
    \setmathfont{NewCMSansMath-Regular.otf}[range={scr,bfscr},
      StylisticSet=1,Numbers=Lining,BoldFont=NewCMMath-Bold.otf]
  }{
    \setmathfont{NewCMMath-Regular.otf}[range={scr,bfscr},
      StylisticSet=1,Numbers=Lining,BoldFont=NewCMMath-Bold.otf]
  }
}
```

`hep_math_family:nnnn` For `symbols=true` and `ams` allocate the math family `#1` past the limit of the LaTeX kernel at the

beginning of the document, which uses the font #3 with the features #4 as the font family #2. LUAOTFLOAD only applies the stylistic set to math characters in its base mode.

```
\cs_new_protected:Npn \__hep_math_family:nnnn #1#2#3#4 {
  \use:e {
    \exp_not:N \newfontfamily \exp_not:c {hep@#1math} {#3} [
      NFSSFamily = hep-#2, Script = Math, Numbers = Lining,
      \sys_if_engine_luatex:T {Renderer = Basic,} #4
    ]
  }
  \AddToHook{begindocument}{
    \exp_args:Nc \new@mathgroup {sym#1}
    \exp_args:Nc \new@symbolfont {sym#1}{TU}{hep-#2}{m}{n}
  }
}
```

The calligraphic capitals of the Euler math font are the upright calligraphic alphabet. The stylistic set of the NCM math fonts provides the italic script alphabets, which are slanted back for the upright ones.

```
\exp_args:NnV \clist_if_in:nnT {true, ams} \hepmathfont@symbols {
  \__hep_math_family:nnnn {hepeuler}{euler}{Euler-Math.otf}{}
  \__hep_math_family:nnnn {hepscr}{scr}{NewCMMath-Regular.otf}
    {StylisticSet = 1}
  \__hep_math_family:nnnn {hepsansscr}{sansscr}
    {NewCMSansMath-Regular.otf}{StylisticSet = 1}
  \__hep_math_family:nnnn {hepscrup}{scrup}{NewCMMath-Regular.otf}
    {StylisticSet = 1, FakeSlant = -0.4}
  \__hep_math_family:nnnn {hepsansscrup}{sansscrup}
    {NewCMSansMath-Regular.otf}{StylisticSet = 1, FakeSlant = -0.4}
}
```

`math_backend_select:` Select the serif or sans-serif math font and the style 0 to 3 for upright, italic, bold upright and bold italic.

```
\cs_new_protected:Npn \hep_math_backend_select: {
  \int_set:Nn \l__hep_selected_family_int {\symhepserif}
  \str_if_eq:VnT \l__hep_family_tl {sans}
    {\int_set:Nn \l__hep_selected_family_int {\symhepsans}}
  \int_set:Nn \l__hep_style_int {1}
  \bool_if:NT \l__hep_up_bool {\int_zero:N \l__hep_style_int}
  \bool_if:NT \l__hep_bold_bool {\int_add:Nn \l__hep_style_int {2}}
}
```

`\hep_math_map:nnnnn` Map the input character #2 and the italic character #3 to the Unicode character of the selected style. The arguments #2 to #5 are the upright, italic, bold and bold italic characters. A bold character which equals its medium character has no bold Unicode character and is taken from the bold serif math font, or for sans-serif from the bold sans text face if it exists there. The typewriter family uses the plain characters from the mono face of the selected weight and shape, but digits are always upright. With the `proportional` option the characters below the Greek block come from the proportional face, once it is allocated. Characters which do not exist in

the text faces, such as \ell and \partial, stay in the math font. So does the vartheta of the medium upright mono face, which is drawn as an ordinary theta.

```
\cs_new_protected:Npn \hep_math_map:nnnnn #1#2#3#4#5 {
  \int_set:Nn \l_tmpa_int
    {\clist_item:nn {#2,#3,#4,#5}{\l__hep_style_int+1}}
  \int_set_eq:NN \l_tmpb_int \l__hep_selected_family_int
  \bool_if:NT \l__hep_bold_bool {
    \int_compare:nNnT {\l_tmpa_int} =
      {\clist_item:nn {#2,#3}{\l__hep_style_int-1}} {
      \int_set:Nn \l_tmpb_int {\symhepserifbf}
      \bool_lazy_and:nnT
        {\str_if_eq_p:Vn \l__hep_family_tl {sans}}
        {\int_compare_p:nNn {#2} < {"2100}} {
        \int_set:Nn \l_tmpa_int {#2}
        \int_set:Nn \l_tmpb_int {\use:c {symhepsansbf
          \bool_if:NTF \l__hep_up_bool {up}{it}}}
      }
    }
  }
  \str_if_eq:VnT \l__hep_family_tl {mono} {
    \bool_lazy_or:nnF
      {\int_compare_p:nNn {#2} > {"20FF}}
      {
        \bool_lazy_and_p:nn
          {\int_compare_p:nNn {#2} = {"03D1}}
          {\int_compare_p:nNn {\l__hep_style_int} = {0}}
      } {
        \int_set:Nn \l_tmpa_int {#2}
        \tl_set:Nn \l_tmpa_tl {mono}
        \bool_lazy_and:nnT
          {\cs_if_exist_p:N \symheppropup}
          {\int_compare_p:nNn {#2} < {"0370}}
          {\tl_set:Nn \l_tmpa_tl {prop}}
        \int_set:Nn \l_tmpb_int {\use:c {symhep\l_tmpa_tl
          \bool_if:NT \l__hep_bold_bool {bf}
          \bool_if:NTF \l__hep_up_bool {up}{it}}}
        \int_compare:nT {48<=#2<=57} {
          \int_set:Nn \l_tmpb_int {\use:c {symhep\l_tmpa_tl
            \bool_if:NT \l__hep_bold_bool {bf}up}}
        }
      }
  }
  \Umathcode #2 = 7~\l_tmpb_int~\l_tmpa_int\scan_stop:
  \Umathcode #3 = 7~\l_tmpb_int~\l_tmpa_int\scan_stop:
}
```

\hep_math_bb_select:
\hep_math_bb_map:nnn

Select the serif, bold serif or sans-serif face for the blackboard alphabet. The typewriter family uses the serif face and there is no bold sans-serif face. Map the input character #2 to the blackboard character #3.

```
\cs_new_protected:Npn \hep_math_bb_select: {
  \int_set:Nn \l__hep_bb_family_int {
    \str_if_eq:VnTF \l__hep_family_tl {sans} {\symhepsans}
      {\bool_if:NTF \l__hep_bold_bool {\symhepserifbf}{\symhepserif}}
  }
}
\cs_new_protected:Npn \hep_math_bb_map:nnn #1#2#3 {
  \Umathcode #2 = 7~\l__hep_bb_family_int~#3\scan_stop:
}
```

`hep_math_bb_glyph:nn` The Unicode math fonts name their blackboard glyphs correctly.

```
\cs_new_protected:Npn \hep_math_bb_glyph:nn #1#2 {}
```

`hep_math_cal_select:` `hep_math_cal_map:nnn` Select the serif or the sans-serif math face for the italic calligraphic capitals and the Euler math font for the upright ones. Select the serif or the sans-serif script face, with `synthetic=false` always the italic one. The typewriter family uses the sans-serif capitals, whose strokes have a uniform width. Map the input character `#2` to the script character `#3`, or for bold to the bold script character, which has no gaps in Unicode.

```
\cs_new_protected:Npn \hep_math_cal_select: {
  \int_set:Nn \l__hep_cal_family_int {
    \bool_if:NTF \l__hep_scr_bool {
      \use:c {
        symhep \str_if_eq:VnF \l__hep_family_tl {serif} {sans} scr
        \ifhepmathfont@synthetic
          \bool_if:NT \l__hep_up_bool {up}
        \fi
      }
    }{
      \bool_if:NTF \l__hep_up_bool {\symhepeuler} {
        \str_if_eq:VnTF \l__hep_family_tl {serif}
          {\symhepserif}{\symhepsans}
      }
    }
  }
}
\cs_new_protected:Npn \hep_math_cal_map:nnn #1#2#3 {
  \Umathcode #2 = 7~\l__hep_cal_family_int~\int_eval:n {
    \bool_if:NTF \l__hep_bold_bool {"1D4D0+#2-‘A}{#3}
  }\scan_stop:
}
```

`p_math_glyph:nnnnnnn` Register a glyph with its slot `#1`, its macro name `#2`, the name `#3` of its text command, and its four Unicode characters. The math codes of the upright and the italic character are set. Greek letter macros select the character by its math code.

```
\cs_new_protected:Npn \hep_math_glyph:nnnnnnn #1#2#3#4#5#6#7 {
  \tl_gput_right:Nn \g__hep_glyph_setup_tl
    {\hep_math_map:nnnnn {#1}{#4}{#5}{#6}{#7}}
  \tl_gput_right:Nn \g__hep_math_codes_tl {{#4}{#5}}
```

```
    \tl_if_empty:nF {#2} {
      \cs_gset:cpn {hep_math_char_#1:}{\char_generate:nn {#4}{12}}
      \hep_math_greek_record:nnn {#1}{#2}{#3}
    }
  }
```

`ep_math_saved_code:n` Expand to the assignment which restores the current math code of the character `#1`.

```
  \cs_new:Npn \__hep_math_saved_code:n #1 {
    \exp_not:N \Umathcodenum #1 = \tex_the:D \Umathcodenum #1
    \exp_not:N \scan_stop:
  }
```

`\hep_math_accent:nnn` `p_math_accent_shape:` The accent font hepaccent.ttf contains the accents derived from the serif, sans-serif and bold NCM math fonts as consecutive private use characters, starting at the character `#3`. Like the script families it is allocated past the limit of the LaTeX kernel at the beginning of the document. LUAOTFLOAD only reads the accent attachments of its math table with the math script, which is passed directly, because FONTSPEC only recognises scripts with features. Each accent selects the shape matching the math font of the math version, whose vector accent is used by `\vec`, and is fixed in size like `\vec`. Without the accent font the accents fall back to `\vec` with a warning. The name of the font is a macro, so that the tests can simulate a missing font.

```
  \providecommand*\hepmathfont@accentfont{hepaccent.ttf}
  \IfFontExistsTF{\hepmathfont@accentfont}{}{
    \hepmathfont@accentsfalse
    \PackageWarningNoLine{hep-math-font}{
      The~accent~font~\hepmathfont@accentfont\space
      was~not~found.\MessageBreak
      The~vector~accents~fall~back~to~\protect\vec.\MessageBreak
      Install~the~fonts~of~hep-math-font
    }
  }
  \ifhepmathfont@accents
    \newfontfamily\hep@accentmath{\hepmathfont@accentfont}[
      NFSSFamily=hep-accent, RawFeature={script=math}]
    \AddToHook{begindocument}{
      \new@mathgroup \symhepaccent
      \new@symbolfont \symhepaccent {TU}{hep-accent}{m}{n}
    }
    \cs_new:Npn \__hep_math_accent_shape: {
      \str_if_eq:VnTF \math@version {bold} {2}
        {\str_if_eq:VnTF \g__hep_main_family_tl {sans} {1}{0}}
    }
    \cs_new_protected:Npn \hep_math_accent:nnn #1#2#3 {
      \cs_new_protected:cpn {#1} {
        \Umathaccent fixed~0~\symhepaccent
          \int_eval:n {#3+\__hep_math_accent_shape:}\scan_stop:
      }
    }
  \else
    \cs_new_protected:Npn \hep_math_accent:nnn #1#2#3
```

```
      {\cs_new_protected:cpn {#1} {\vec}}
\fi
```

\bm Provide the \bm macro, which is not needed by the Unicode math fonts.

```
\ProvideDocumentCommand\bm{m}{\boldsymbol{#1}}
\ExplSyntaxOff
```

A.4 pdfTEX

Use the LM fonts for operators, letters and symbols.

```
\else
  \DeclareSymbolFont{operators}{OT1}{lmr}{m}{n}
  \DeclareSymbolFont{letters}{OML}{lmm}{m}{it}
  \DeclareSymbolFont{symbols}{OMS}{lmsy}{m}{n}
  \DeclareSymbolFont{largesymbols}{OMX}{lmex}{m}{n}
  \SetSymbolFont{operators}{bold}{OT1}{lmr}{bx}{n}
  \SetSymbolFont{letters}{bold}{OML}{lmm}{b}{it}
  \SetSymbolFont{symbols}{bold}{OMS}{lmsy}{b}{n}
```

Use the CM sans-serif math fonts of the SANSMATHFONTS package [13] in sans-serif documents.

```
  \ExplSyntaxOn
  \str_if_eq:VnT \g__hep_main_family_tl {sans} {
    \DeclareSymbolFont{operators}{OT1}{lmss}{m}{n}
    \SetSymbolFont{operators}{bold}{OT1}{lmss}{bx}{n}
    \DeclareSymbolFont{letters}{OML}{cmssm}{m}{it}
    \SetSymbolFont{letters}{bold}{OML}{cmssm}{b}{it}
    \DeclareSymbolFont{symbols}{OMS}{cmsssy}{m}{n}
    \SetSymbolFont{symbols}{bold}{OMS}{cmsssy}{b}{n}
    \DeclareSymbolFont{largesymbols}{OMX}{cmssex}{m}{n}
  }
```

Load the symbol fonts selected by the `symbols` option.

```
  \str_if_eq:VnF \hepmathfont@symbols {false} {
    \str_if_eq:VnTF \hepmathfont@symbols {minion}
      {\RequirePackage{MnSymbol}}
      {\RequirePackage{amssymb}}
  }
```

A.4.1 Generated fonts

\hepmathfont@vfversion The generator script writes the file hep-math-font-vf.def next to the virtual fonts, which defines \hepmathfont@vfversion as the date, version and description of the package file whose glyph tables the fonts were generated from. The file is installed with the package. If it is missing, so are the fonts presumably, and the package falls back to the standard math alphabets with a warning. If its version differs from that of this package file, the fonts may have other slots than the glyph table and would select wrong glyphs. This is an error, after which the package also falls back to the standard math alphabets. The version contains spaces, so the file is read with

the standard category codes. An empty `\hepmathfont@vfversion` defined before the package is loaded counts as missing fonts.

```
\ExplSyntaxOff
\@ifundefined{hepmathfont@vfversion}
  {\InputIfFileExists{hep-math-font-vf.def}{}{}}{}
\@ifundefined{hepmathfont@vfversion}{\def\hepmathfont@vfversion{}}{}
\ExplSyntaxOn
\tl_if_empty:NTF \hepmathfont@vfversion {
  \hepmathfont@vffalse
  \PackageWarningNoLine{hep-math-font}{
    The~virtual~fonts~for~pdfLaTeX~were~not~found.\MessageBreak
    The~math~alphabets~are~not~composable~and~the\MessageBreak
    vector~accents~fall~back~to~\protect\vec.\MessageBreak
    Install~the~fonts~of~hep-math-font
  }
}{
  \str_if_eq:eeF {\hepmathfont@vfversion}
    {\use:c {ver@hep-math-font.sty}} {
    \hepmathfont@vffalse
    \PackageError{hep-math-font}{
      The~virtual~fonts~were~generated~for~another\MessageBreak
      version~of~hep-math-font.sty
    }{
      Generate~them~again~with~hep-math-font-vf.py~or~install~the
      ~package~again.~The~document~uses~the~standard~math
      ~alphabets~instead.
    }
  }
}
```

`ep_math_tounicode:nn` `hep_math_tounicode:w` Some source fonts name their glyphs after the characters of a text encoding, so that the text extracted from the document would be wrong. Map the glyphs `#2`, given as ⟨*glyph*⟩=⟨*hexadecimal character*⟩ pairs, of the font `#1` to the correct Unicode characters, which only affects this font.

```
\cs_new_protected:Npn \__hep_math_tounicode:nn #1#2 {
  \clist_map_inline:nn {#2}
    {\__hep_math_tounicode:w #1 / ##1 \q_stop}
}
\cs_new_protected:Npn \__hep_math_tounicode:w #1 / #2 = #3 \q_stop
  {\pdfglyphtounicode {tfm:#1/#2}{#3}}
\cs_generate_variant:Nn \__hep_math_tounicode:nn {nV}
```

`__hep_math_oml_clist` The CM sans-serif math italic fonts and their slanted copies carry the glyph names of the T1 encoding in the slots of the OML encoding. The sans-serif AMS blackboard does the same, the AMS kappa symbol is named as the ordinary kappa, and the LM math italic Delta and mu are named as the increment and micro signs.

```
\clist_const:Nn \c__hep_math_oml_clist {
  grave=0393, acute=0394, circumflex=0398, tilde=039B,
  dieresis=039E, hungarumlaut=03A0, ring=03A3, caron=03A5,
```

```
    breve=03A6, macron=03A8, dotaccent=03A9, cedilla=03B1,
    ogonek=03B2, quotesinglbase=03B3, guilsinglleft=03B4,
    guilsinglright=03F5, quotedblleft=03B6, quotedblright=03B7,
    quotedblbase=03B8, guillemotleft=03B9, guillemotright=03BA,
    endash=03BB, emdash=03BC, cwm=03BD, perthousand=03BE,
    dotlessi=03C0, dotlessj=03C1, ff=03C3, fi=03C4, fl=03C5,
    ffi=03D5, ffl=03C7, space=03C8, exclam=03C9, quotedbl=03B5,
    numbersign=03D1, dollar=03D6, percent=03F1, ampersand=03C2,
    quoteright=03C6, parenleft=21BC, parenright=21BD,
    asterisk=21C0, plus=21C1, period=25B9, slash=25C3,
    colon=002E, semicolon=002C, equal=002F, question=22C6,
    at=2202, bracketleft=266D, backslash=266E, bracketright=266F,
    asciicircum=2323, underscore=2322, quoteleft=2113,
    braceleft=0131, bar=0237, braceright=2118
  }
```

`th_tounicode_sizes:n` Correct the glyph names of these fonts at all sizes, where `#1` is empty for the installed fonts and `hepslant-` for the slanted copies.

```
  \cs_new_protected:Npn \__hep_math_tounicode_sizes:n #1 {
    \int_step_inline:nnn {5}{10} {
      \__hep_math_tounicode:nV {#1cmssmi##1} \c__hep_math_oml_clist
      \__hep_math_tounicode:nV {#1cmssmib##1} \c__hep_math_oml_clist
      \__hep_math_tounicode:nn {#1ssmsbm##1}{braceleft=03F0}
      \__hep_math_tounicode:nn {#1msbm##1}{kappa=03F0}
      \__hep_math_tounicode:nn {#1lmmi##1}{Delta=0394, mu=03BC}
      \__hep_math_tounicode:nn {#1lmmib##1}{Delta=0394, mu=03BC}
    }
  }
  \__hep_math_tounicode_sizes:n {}
```

The rest of the pdfTEX setup uses the virtual fonts.

```
  \ifhepmathfont@vf
```

A.4.2 Decorative alphabets

Pack the decorative alphabets into the `hepdecor` virtual fonts, whose slot 255 is an empty skew character, and use them as a single family. For `symbols=minion` these are all four decorative alphabets, because the MnSymbol fonts already use eight math families. They are provided in the banks 0 to 3 for the serif and sans-serif blackboard in medium and bold weight. Otherwise these are the Euler fraktur [2] in the banks 4 and 5 for medium and bold weight. These banks also contain the four composable italic calligraphic alphabets after the decorative ones. The bold math version uses the bold bank.

```
  \str_if_eq:VnF \hepmathfont@symbols {false} {
    \int_step_inline:nnn {0}{5} {
      \DeclareFontFamily{U}{hepdecor#1}{\skewchar\font=255}
      \DeclareFontShape{U}{hepdecor#1}{m}{n}
        {<-6>hepdecor#1-5<6-9>hepdecor#1-7<9->hepdecor#1-10}{}
    }
    \int_set:Nn \l_tmpa_int {
```

```
    \str_if_eq:VnTF \hepmathfont@symbols {minion}
      {\str_if_eq:VnTF \g__hep_main_family_tl {sans} {2}{0}}{4}
  }
  \use:e {
    \exp_not:N \DeclareSymbolFont {hepdecor}{U}
      {hepdecor\int_use:N \l_tmpa_int}{m}{n}
    \exp_not:N \SetSymbolFont {hepdecor}{bold}{U}
      {hepdecor\int_eval:n {\l_tmpa_int+1}}{m}{n}
  }
```

\mathcal
\mathscr
\mathbb
\mathfrak

Each decorative alphabet command maps its characters to the packed family within a group.

```
  \clist_map_inline:nn {mathcal, mathscr, mathbb, mathfrak}
    {\tl_new:c {g__hep_decor_#1_tl}}
  \clist_set:Nn \l_tmpa_clist {mathfrak}
  \str_if_eq:VnT \hepmathfont@symbols {minion} {
    \clist_set:Nn \l_tmpa_clist {mathcal, mathscr, mathbb, mathfrak}
  }
  \clist_map_inline:Nn \l_tmpa_clist {
    \cs_set_protected:cpn {#1} ##1 {
      \group_begin:
      \tex_fam:D -1\scan_stop:
      \tl_use:c {g__hep_decor_#1_tl}
      \tl_clear:N \l__hep_applied_tl
      ##1\group_end:
    }
  }
```

`math_decor_glyph:nnn` Map the character `#2` of the alphabet `#1` to the slot `#3`.

```
  \cs_new_protected:Npn \hep_math_decor_glyph:nnn #1#2#3 {
    \tl_gput_right:cn {g__hep_decor_#1_tl}
      {\mathcode #2=\int_eval:n {"7000+256*\symhepdecor+#3}\scan_stop:}
  }
```

`__hep_math_decor:nnn` Register the `#3` consecutive characters starting at the hexadecimal character code `#2` of the alphabet `#1` in the next free slots.

```
  \int_new:N \g__hep_math_decor_int
  \cs_new_protected:Npn \__hep_math_decor:nnn #1#2#3 {
    \int_step_inline:nnn {0}{#3-1} {
      \use:e {
        \exp_not:N \hep_math_decor_glyph:nnn {#1}
          {\int_eval:n {\int_from_hex:n {#2}+##1}}
          {\int_use:N \g__hep_math_decor_int}
      }
      \int_gincr:N \g__hep_math_decor_int
    }
  }
```

The decorative glyph table is also read by the generator script of the virtual fonts. The slots of the decorative `\mathcal`, `\mathscr` and `\mathbb` are empty in the banks 4 and 5. RSFS only

contains capitals and the double-stroke blackboard [8] contains capitals and the characters h, k and 1.

```
    \__hep_math_decor:nnn {mathcal}{0041}{26}
    \__hep_math_decor:nnn {mathscr}{0041}{26}
    \__hep_math_decor:nnn {mathbb}{0041}{26}
    \__hep_math_decor:nnn {mathbb}{0068}{1}
    \__hep_math_decor:nnn {mathbb}{006B}{1}
    \__hep_math_decor:nnn {mathbb}{0031}{1}
    \__hep_math_decor:nnn {mathfrak}{0041}{26}
    \__hep_math_decor:nnn {mathfrak}{0061}{26}
    \__hep_math_decor:nnn {mathfrak}{0030}{10}
}
```

A.4.3 Script alphabets

For `symbols=true` and `ams` the `hepscript` virtual font contains the italic and the upright script and the upright calligraphic alphabets, each in medium and bold weight. It does not depend on the math version either.

```
\exp_args:NnV \clist_if_in:nnT {true, ams} \hepmathfont@symbols {
  \DeclareFontFamily{U}{hepscript}{\skewchar\font=255}
  \DeclareFontShape{U}{hepscript}{m}{n}
    {<-6>hepscript-5<6-9>hepscript-7<9->hepscript-10}{}
  \DeclareSymbolFont{hepscript}{U}{hepscript}{m}{n}
}
```

A.4.4 Blackboard alphabet

For `symbols=true` the `hepbb` virtual font contains the NCM blackboard of the MATHALPHA package and the sans-serif blackboard of the BBOLDX package, both in medium and bold weight. The blackboard does not depend on the math version, because the state already includes the weight.

```
\str_if_eq:VnT \hepmathfont@symbols {true} {
  \DeclareFontFamily{U}{hepbb}{\skewchar\font=255}
  \DeclareFontShape{U}{hepbb}{m}{n}
    {<-6>hepbb-5<6-9>hepbb-7<9->hepbb-10}{}
  \DeclareSymbolFont{hepbb}{U}{hepbb}{m}{n}
}
```

A.4.5 Composable alphabets

The eight `hepalpha` virtual font families contain the serif, sans-serif, typewriter and proportional typewriter alphabets in medium and bold. Their lower half contains the italic and their upper half the upright shapes, slot 255 is an empty skew character. The fonts are provided at 5 pt, 7 pt and 10 pt.

```
\int_step_inline:nnn {0}{7} {
  \DeclareFontFamily{U}{hepalpha#1}{\skewchar\font=255}
  \DeclareFontShape{U}{hepalpha#1}{m}{n}
    {<-6>hepalpha#1-5<6-9>hepalpha#1-7<9->hepalpha#1-10}{}
}
```

__hep_math_bank:n Only six banks are declared as symbol fonts to save math families. With the `proportional` option the typewriter banks 4 and 5 use the proportional font families 6 and 7. The bold math version uses the bold family of the same bank.

```
\cs_new:Npn \__hep_math_bank:n #1 {
  \int_eval:n {#1 \ifhepmathfont@proportional
    \int_compare:nNnT {#1} > {3} {+2} \fi}
}
\int_step_inline:nnn {0}{5} {
  \use:e {
    \exp_not:N \DeclareSymbolFont {hepalpha#1}{U}
      {hepalpha\__hep_math_bank:n {#1}}{m}{n}
    \exp_not:N \SetSymbolFont {hepalpha#1}{bold}{U}
      {hepalpha\__hep_math_bank:n
        {2*\int_div_truncate:nn {#1}{2}+1}}{m}{n}
  }
}
```

A.4.6 Base fonts

The virtual fonts also refer to base fonts which the generator script derives from installed fonts. These are the CM Unicode and LM text fonts in an encoding of their Greek letters and h-bar, and slanted copies of math fonts. Their map file declares the slanted copies with the `SlantFont` operator. With `synthetic=false` the alternative map file declares them without it, so that the original shapes are used.

```
\pdfmapfile{=hep-math-font\ifhepmathfont@synthetic\else-fallback\fi.map}
```

Correct the glyph names of the slanted copies of these fonts as well.

```
\__hep_math_tounicode_sizes:n {hepslant-}
```

The CM Unicode fonts distinguish the Greek Delta, Omega and mu from the increment, ohm and micro signs by a suffix of the glyph name, which pdfTEX drops.

```
\clist_map_inline:nn {rm, bx, ss, sx, tt, tb, it, tx} {
  \__hep_math_tounicode:nn {hep-cmun#1}
    {Delta=0394, Omega=03A9, mu=03BC}
}
```

math_backend_select: Select the bank 0, 2 or 4 for serif, sans-serif or typewriter, plus one for bold, and the offset 128 for upright.

```
\cs_new_protected:Npn \hep_math_backend_select: {
  \int_set:Nn \l__hep_bank_int {
    \str_case:Vn \l__hep_family_tl {{serif}{0}{sans}{2}{mono}{4}}
  }
  \bool_if:NT \l__hep_bold_bool {\int_incr:N \l__hep_bank_int}
  \int_set:Nn \l__hep_selected_family_int
    {\use:c {symhepalpha\int_use:N \l__hep_bank_int}}
  \int_zero:N \l__hep_offset_int
  \bool_if:NT \l__hep_up_bool
    {\int_set:Nn \l__hep_offset_int {128}}
```

```
}
```

`\hep_math_map:nnnnn` Map the input character `#2` to the slot `#1` in the selected half of the selected bank. Only characters below 128 can be input directly.

```
\cs_new_protected:Npn \hep_math_map:nnnnn #1#2#3#4#5 {
  \int_compare:nNnT {#2} < {128} {
    \mathcode #2 = \int_eval:n {
      "7000 + 256 * \l__hep_selected_family_int
      + \l__hep_offset_int + #1
    }\scan_stop:
  }
}
```

`\hep_math_bb_select:` `\hep_math_bb_map:nnn` Select the blackboard alphabet 0 to 3 for serif and sans-serif in medium and bold weight. The typewriter family uses the serif alphabet. Map the input character `#2` to the slot `#1` of the selected alphabet.

```
\cs_new_protected:Npn \hep_math_bb_select: {
  \int_set:Nn \l__hep_bb_family_int {\symhepbb}
  \int_set:Nn \l__hep_bb_offset_int {
    \g__hep_math_bb_int * (
      \str_if_eq:VnTF \l__hep_family_tl {sans} {2}{0}
      + \bool_if:NTF \l__hep_bold_bool {1}{0}
    )
  }
}
\cs_new_protected:Npn \hep_math_bb_map:nnn #1#2#3 {
  \mathcode #2 = \int_eval:n {
    "7000 + 256 * \l__hep_bb_family_int + \l__hep_bb_offset_int + #1
  }\scan_stop:
}
```

`hep_math_bb_glyph:nn` The BBOLDX fonts name their glyphs after the ASCII characters. Map the glyph of the input character `#1` to the blackboard character `#2`.

```
\cs_new_protected:Npn \hep_math_bb_glyph:nn #1#2 {
  \clist_map_inline:nn {Regular, Bold} {
    \pdfglyphtounicode {
      tfm:BBOLDX-##1/
      \int_compare:nNnTF {#1} < {58} {
        \clist_item:nn
          {zero, one, two, three, four, five, six, seven, eight, nine}
          {#1-47}
      }{\char_generate:nn {#1}{12}}
    }{\int_to_Hex:n {#2}}
  }
}
```

`hep_math_cal_select:` `_math_script_select:` `hep_math_cal_select:` `hep_math_cal_map:nnn` Select the italic calligraphic alphabet 0 to 3 for serif and sans-serif in medium and bold weight, which follow the decorative alphabets in the `hepdecor` virtual fonts. With `synthetic=false`

the sans-serif and typewriter families use the medium alphabet in both weights. The typewriter family uses the sans-serif alphabet. The script alphabets 0 to 3 of the `hepscript` virtual fonts are italic and upright in medium and bold weight, the alphabets 4 and 5 are the upright calligraphic alphabets. With `synthetic=false` the medium italic script alphabet is used. Map the input character `#2` to the slot `#1` of the selected alphabet.

```
\cs_new_protected:Npn \hep_math_cal_select: {
  \bool_lazy_or:nnTF {\l__hep_scr_bool}{\l__hep_up_bool}
    {\__hep_math_script_select:}{\__hep_math_cal_select:}
}
\cs_new_protected:Npn \__hep_math_script_select: {
  \int_set:Nn \l__hep_cal_family_int {\symhepscript}
  \int_set:Nn \l__hep_cal_offset_int {
    \g__hep_math_cal_int * (
      \bool_if:NTF \l__hep_bold_bool {1}{0}
      + \bool_if:NTF \l__hep_scr_bool
        {\bool_if:NTF \l__hep_up_bool {2}{0}}{4}
    )
  }
  \ifhepmathfont@synthetic\else
    \bool_if:NT \l__hep_scr_bool
      {\int_zero:N \l__hep_cal_offset_int}
  \fi
}
\cs_new_protected:Npn \__hep_math_cal_select: {
  \int_set:Nn \l__hep_cal_family_int {\symhepdecor}
  \int_set:Nn \l__hep_cal_offset_int
    {\bool_if:NTF \l__hep_bold_bool {1}{0}}
  \str_if_eq:VnF \l__hep_family_tl {serif} {
    \ifhepmathfont@synthetic\else
      \int_zero:N \l__hep_cal_offset_int
    \fi
    \int_add:Nn \l__hep_cal_offset_int {2}
  }
  \int_set:Nn \l__hep_cal_offset_int {
    \g__hep_math_decor_int
    + \g__hep_math_cal_int * \l__hep_cal_offset_int
  }
}
\cs_new_protected:Npn \hep_math_cal_map:nnn #1#2#3 {
  \mathcode #2 = \int_eval:n {
    "7000 + 256 * \l__hep_cal_family_int + \l__hep_cal_offset_int + #1
  }\scan_stop:
}
```

`p_math_glyph:nnnnnnn` Register a glyph with its slot `#1`, its macro name `#2`, the name `#3` of its text command, and its four Unicode characters. Record the original math codes of the directly input characters, which are restored for operator names and the decorative alphabets. Otherwise a slot of the virtual fonts would be interpreted as a slot of the original font. Greek letter macros select the character by its math character.

```
\cs_new_protected:Npn \hep_math_glyph:nnnnnnn #1#2#3#4#5#6#7 {
  \tl_gput_right:Nn \g__hep_glyph_setup_tl
    {\hep_math_map:nnnnn {#1}{#4}{#5}{#6}{#7}}
  \int_compare:nNnT {#4} < {128} {
    \tl_gput_right:Nn \g__hep_math_codes_tl {{#4}}
    \tl_gput_right:Nx \g__hep_native_codes_tl
      {\mathcode #4=\the\mathcode #4\scan_stop:}
  }
  \tl_if_empty:nF {#2} {
    \cs_gset_protected:cpn {hep_math_char_#1:} {
      \mathchar\int_eval:n {
        256 * \l__hep_selected_family_int + \l__hep_offset_int + #1
      }\scan_stop:
    }
    \hep_math_greek_record:nnn {#1}{#2}{#3}
  }
}
```

ep_math_saved_code:n Expand to the assignment which restores the current math code of the character #1.

```
\cs_new:Npn \__hep_math_saved_code:n #1 {
  \exp_not:N \mathcode #1 = \tex_the:D \mathcode #1
  \exp_not:N \scan_stop:
}
```

\hep_math_accent:nnn Declare the accent #1 in the slot #2 of the serif or sans-serif bank, matching the letters font, whose vector accent is used by \vec. The bold math version uses the bold bank, just as \vec uses the bold math italic font.

```
\cs_new_protected:Npn \hep_math_accent:nnn #1#2#3 {
  \use:e {
    \exp_not:N \DeclareMathAccent \exp_not:c {#1} {\exp_not:N \mathord}
      {hepalpha \str_if_eq:VnTF \g__hep_main_family_tl {sans} {2}{0}}
      {#2}
  }
}
```

A.4.7 Fallback

Without the virtual fonts the glyph table registers nothing and the vector accents are the ordinary one.

```
\else
  \cs_new_protected:Npn \hep_math_glyph:nnnnnnn #1#2#3#4#5#6#7 {}
  \cs_new_protected:Npn \hep_math_bb_glyph:nn #1#2 {}
  \cs_new_protected:Npn \hep_math_accent:nnn #1#2#3
    {\cs_new_protected:cpn {#1} {\vec}}
```

The standard math alphabets use the LM fonts of the main family. Upright is the operator font, the serif family keeps the italic shape in sans-serif documents, and the combined commands select the nearest standard alphabet.

```
    \tl_set:Ne \l_tmpa_tl
      {\str_if_eq:VnTF \g__hep_main_family_tl {sans} {lmss}{lmr}}
    \cs_new_protected:Npn \__hep_math_standard:NNnnn #1#2#3#4#5 {
      \use:e {
        \exp_not:N \DeclareMathAlphabet \exp_not:N #1
          {OT1}{#2}{#3}{#4}
        \exp_not:N \SetMathAlphabet \exp_not:N #1
          {bold}{OT1}{#2}{#5}{#4}
      }
    }
    \__hep_math_standard:NNnnn \mathbf \l_tmpa_tl {bx}{n}{bx}
    \__hep_math_standard:NNnnn \mathit \l_tmpa_tl {m}{it}{bx}
    \tl_set:Nn \l_tmpb_tl {lmss}
    \__hep_math_standard:NNnnn \mathsf \l_tmpb_tl {m}{n}{bx}
    \__hep_math_standard:NNnnn \mathbfsf \l_tmpb_tl {bx}{n}{bx}
    \tl_set:Nn \l_tmpb_tl {lmtt}
    \__hep_math_standard:NNnnn \mathtt \l_tmpb_tl {m}{n}{b}
    \__hep_math_standard:NNnnn \mathbftt \l_tmpb_tl {b}{n}{b}
    \str_if_eq:VnTF \g__hep_main_family_tl {sans} {
      \tl_set:Nn \l_tmpb_tl {lmr}
      \__hep_math_standard:NNnnn \mathserif \l_tmpb_tl {m}{it}{bx}
    }{\cs_gset_eq:NN \mathserif \mathnormal}
    \cs_gset_eq:NN \mathup \mathrm
    \cs_gset_eq:NN \mathbfrm \mathbf
    \cs_gset_eq:NN \mathbfup \mathbf
    \cs_gset_eq:NN \mathsfup \mathsf
    \cs_gset_eq:NN \mathsftext \mathsf
    \cs_gset_eq:NN \mathbfsfup \mathbfsf
    \cs_gset_eq:NN \mathttup \mathtt
    \cs_gset_eq:NN \mathbfttup \mathbftt
```

The decorative alphabets are the AMS blackboard and Euler fraktur and the RSFS script, unless the symbol fonts already provide them.

```
    \str_if_eq:VnF \hepmathfont@symbols {false} {
      \cs_if_exist:NF \mathbb {\DeclareMathAlphabet \mathbb {U}{msb}{m}{n}}
      \cs_if_exist:NF \mathfrak
        {\DeclareMathAlphabet \mathfrak {U}{euf}{m}{n}}
      \cs_if_exist:NF \mathscr
        {\DeclareMathAlphabet \mathscr {U}{rsfs}{m}{n}}
    }
  \fi
  \ExplSyntaxOff
```

`\bm` Load the BM package [1] without reserving additional math families.

```
  \providecommand\bmmax{0}
  \providecommand\hmmax{0}
  \RequirePackage{bm}
\fi
```

A.5 Glyph table

Load the alphabeta package [14] for the Greek text commands.

```
\RequirePackage{alphabeta}
\ExplSyntaxOn
```

`ath_greek_record:nnn` `p_greek_pdfstring_tl` `\g__hep_greek_clist`

A Greek letter macro `#2` selects the glyph in slot `#1` in math and the text command `\text#3` in text. The math variants such as `\varepsilon` typeset the ordinary text letters. The protected macros are not expanded in bookmarks, therefore collect the code that lets them stand for the text commands there, see section A.9. Macros that have been redefined since are left to their new meaning. Also collect the names of the macros.

```
\tl_new:N \g__hep_greek_pdfstring_tl
\clist_new:N \g__hep_greek_clist
\cs_new_protected:Npn \hep_math_greek_record:nnn #1#2#3 {
  \clist_gput_right:Nn \g__hep_greek_clist {#2}
  \tl_gput_right:Nn \g__hep_greek_setup_tl {
    \tl_if_empty:nTF {#3}
      {\cs_gset_protected:cpn {#2}{\use:c {hep_math_char_#1:}}}
      {
        \cs_gset_protected:cpn {#2}
          {\TextOrMath{\use:c {text#3}}{\use:c {hep_math_char_#1:}}}
      }
  }
  \tl_if_empty:nF {#3} {
    \tl_gput_right:Nn \g__hep_greek_pdfstring_tl {
      \cs_if_eq:ccT {#2}{__hep_greek_#2:}
        {\cs_set_eq:cc {#2}{text#3}}
    }
  }
}
```

`ep_math_glyph:nnnnnn` Pass the next slot, the names `#1` and `#2` and the evaluated characters `#3` to `#6` on to the backend.

```
\int_new:N \g__hep_math_slot_int
\cs_new_protected:Npn \__hep_math_glyph:nnnnnn #1#2#3#4#5#6 {
  \use:e {
    \exp_not:N \hep_math_glyph:nnnnnnn
      {\int_use:N \g__hep_math_slot_int} {#1} {#2}
      {\int_eval:n {#3}} {\int_eval:n {#4}}
      {\int_eval:n {#5}} {\int_eval:n {#6}}
  }
  \int_gincr:N \g__hep_math_slot_int
}
```

`hep_math_range:nnnnn` Register `#1` consecutive characters without macro names. The hexadecimal Unicode characters `#2` to `#5` of the first character are the upright, italic, bold and bold italic shapes.

```
\cs_new_protected:Npn \__hep_math_range:nnnnn #1#2#3#4#5 {
  \int_step_inline:nnn {0}{#1-1} {
    \__hep_math_glyph:nnnnnn {}{}
```

```
      {\int_from_hex:n {#2}+##1} {\int_from_hex:n {#3}+##1}
      {\int_from_hex:n {#4}+##1} {\int_from_hex:n {#5}+##1}
  }
}
```

`hep_math_named:nnnnn` Register consecutive characters with the macro names given in the list `#1`. An entry ⟨*name*⟩`/`⟨*text*⟩ uses the text command `\text`⟨*text*⟩, the entry ⟨*name*⟩ uses `\text`⟨*name*⟩ and the entry ⟨*name*⟩`/` has no text command.

```
\seq_new:N \l__hep_math_name_seq
\int_new:N \l__hep_math_step_int
\cs_new_protected:Npn \__hep_math_named:nnnnn #1#2#3#4#5 {
  \int_zero:N \l__hep_math_step_int
  \clist_map_inline:nn {#1} {
    \seq_set_split:Nnn \l__hep_math_name_seq {/} {##1}
    \__hep_math_glyph:nnnnnn
      {\seq_item:Nn \l__hep_math_name_seq {1}}
      {\seq_item:Nn \l__hep_math_name_seq {-1}}
      {\int_from_hex:n {#2}+\l__hep_math_step_int}
      {\int_from_hex:n {#3}+\l__hep_math_step_int}
      {\int_from_hex:n {#4}+\l__hep_math_step_int}
      {\int_from_hex:n {#5}+\l__hep_math_step_int}
    \int_incr:N \l__hep_math_step_int
  }
}
```

The glyph table determines the slots of the virtual fonts and is also read by their generator script. Hence the slots of the virtual fonts have to be generated again whenever the table changes. Latin capitals and small letters, where the italic h is a letterlike symbol in Unicode.

```
\__hep_math_range:nnnnn {26}{0041}{1D434}{1D400}{1D468}
\__hep_math_range:nnnnn {7}{0061}{1D44E}{1D41A}{1D482}
\__hep_math_range:nnnnn {1}{0068}{210E}{1D421}{1D489}
\__hep_math_range:nnnnn {18}{0069}{1D456}{1D422}{1D48A}
```

Digits, whose italic shapes are upright.

```
\__hep_math_range:nnnnn {10}{0030}{0030}{1D7CE}{1D7CE}
```

Greek capitals, skipping the position of the capital theta symbol after Rho.

```
\__hep_math_named:nnnnn {
  Alpha, Beta, Gamma, Delta, Epsilon, Zeta, Eta, Theta, Iota, Kappa,
  Lambda, Mu, Nu, Xi, Omicron, Pi, Rho
}{0391}{1D6E2}{1D6A8}{1D71C}
\__hep_math_named:nnnnn {Sigma, Tau, Upsilon, Phi, Chi, Psi, Omega}
  {03A3}{1D6F4}{1D6BA}{1D72E}
```

Greek small letters, skipping the position of the final sigma after rho. The macros `\varepsilon` and `\varphi` select the ordinary letters, while `\epsilon` and `\phi` select the symbols, as usual in TEX.

```
\__hep_math_named:nnnnn {
```

```
  alpha, beta, gamma, delta, varepsilon/epsilon, zeta, eta, theta, iota,
  kappa, lambda, mu, nu, xi, omicron, pi, rho
}{03B1}{1D6FC}{1D6C2}{1D736}
\__hep_math_named:nnnnn {sigma, tau, upsilon, varphi/phi, chi, psi, omega}
  {03C3}{1D70E}{1D6D4}{1D748}
\__hep_math_named:nnnnn {varsigma/finalsigma}{03C2}{1D70D}{1D6D3}{1D747}
```

Greek symbols, derivatives, the dotless letters, the ell and the h-bar. The last four have no bold Unicode characters, and the ell has no italic one. The upright h-bar is the Latin small letter h with stroke.

```
\__hep_math_named:nnnnn {epsilon}{03F5}{1D716}{1D6DC}{1D750}
\__hep_math_named:nnnnn {vartheta/thetasymbol}{03D1}{1D717}{1D6DD}{1D751}
\__hep_math_named:nnnnn {varkappa/kappasymbol}{03F0}{1D718}{1D6DE}{1D752}
\__hep_math_named:nnnnn {phi}{03D5}{1D719}{1D6DF}{1D753}
\__hep_math_named:nnnnn {varrho/rhosymbol}{03F1}{1D71A}{1D6E0}{1D754}
\__hep_math_named:nnnnn {varpi/pisymbol}{03D6}{1D71B}{1D6E1}{1D755}
\__hep_math_named:nnnnn {varTheta/Theta}{03F4}{1D6F3}{1D6B9}{1D72D}
\__hep_math_named:nnnnn {partial/}{2202}{1D715}{1D6DB}{1D74F}
\__hep_math_named:nnnnn {nabla/}{2207}{1D6FB}{1D6C1}{1D735}
\__hep_math_named:nnnnn {imath/}{0131}{1D6A4}{0131}{1D6A4}
\__hep_math_named:nnnnn {jmath/}{0237}{1D6A5}{0237}{1D6A5}
\__hep_math_named:nnnnn {ell/}{2113}{2113}{2113}{2113}
\__hep_math_named:nnnnn {hbar/}{0127}{210F}{0127}{210F}
```

`g__hep_greek_user_tl` Define the Greek letter macros already now, so that copies such as `\let\vareps\varepsilon` in the preamble select the glyphs of this package, and remember these definitions. The macros are defined again at the beginning of the document, after UNICODE-MATH and ALPHABETA have defined their own. A letter that the preamble or a package loaded later has redefined, e.g. with `\renewcommand{\epsilon}{\varepsilon}`, keeps its new meaning, so collect the code that restores it.

```
\tl_use:N \g__hep_greek_setup_tl
\clist_map_inline:Nn \g__hep_greek_clist
  {\cs_gset_eq:cc {__hep_greek_#1:}{#1}}
\tl_new:N \g__hep_greek_user_tl
\AddToHook{begindocument/before}{
  \clist_map_inline:Nn \g__hep_greek_clist {
    \cs_if_eq:ccF {#1}{__hep_greek_#1:} {
      \cs_gset_eq:cc {__hep_greek_user_#1:}{#1}
      \tl_gput_right:Nn \g__hep_greek_user_tl
        {\cs_gset_eq:cc {#1}{__hep_greek_user_#1:}}
    }
  }
}
```

`_hep_math_accent:nnn` Define the accent `#1` whose shaft has the pattern `#2` in the next slot of the virtual fonts. The hexadecimal character `#3` is the first of its three private use characters for the serif, sans-serif and bold NCM shapes in the accent font of the Unicode engines. The accents follow the glyph table, so that their rows have to be the last ones.

```
\cs_new_protected:Npn \__hep_math_accent:nnn #1#2#3 {
  \exp_args:Nnee \hep_math_accent:nnn {#1}
    {\int_use:N \g__hep_math_slot_int} {\int_from_hex:n {#3}}
  \int_gincr:N \g__hep_math_slot_int
}
```

`\dashedvec` `\dottedvec` The vector accents with a dashed and a dotted shaft.

```
\__hep_math_accent:nnn {dashedvec}{dashed}{E000}
\__hep_math_accent:nnn {dottedvec}{dotted}{E003}
```

`\__hep_math_bb:nnn` Register `#2` consecutive blackboard characters starting at the hexadecimal input character `#1` and the hexadecimal Unicode character `#3`. The row order determines the slots within each blackboard alphabet of the virtual font.

```
\int_new:N \g__hep_math_bb_int
\cs_new_protected:Npn \__hep_math_bb:nnn #1#2#3 {
  \int_step_inline:nnn {0}{#2-1} {
    \hep_math_bb_glyph:nn
      {\int_from_hex:n {#1}+##1} {\int_from_hex:n {#3}+##1}
    \tl_gput_right:Nx \g__hep_bb_setup_tl {
      \exp_not:N \hep_math_bb_map:nnn
        {\int_use:N \g__hep_math_bb_int}
        {\int_eval:n {\int_from_hex:n {#1}+##1}}
        {\int_eval:n {\int_from_hex:n {#3}+##1}}
    }
    \int_gincr:N \g__hep_math_bb_int
  }
}
```

Blackboard capitals, of which C, H, N, P, Q, R and Z are letterlike symbols in Unicode, small letters and digits.

```
\__hep_math_bb:nnn {0041}{2}{1D538}
\__hep_math_bb:nnn {0043}{1}{2102}
\__hep_math_bb:nnn {0044}{4}{1D53B}
\__hep_math_bb:nnn {0048}{1}{210D}
\__hep_math_bb:nnn {0049}{5}{1D540}
\__hep_math_bb:nnn {004E}{1}{2115}
\__hep_math_bb:nnn {004F}{1}{1D546}
\__hep_math_bb:nnn {0050}{2}{2119}
\__hep_math_bb:nnn {0052}{1}{211D}
\__hep_math_bb:nnn {0053}{7}{1D54A}
\__hep_math_bb:nnn {005A}{1}{2124}
\__hep_math_bb:nnn {0061}{26}{1D552}
\__hep_math_bb:nnn {0030}{10}{1D7D8}
```

`\__hep_math_cal:nnn` Register `#2` consecutive calligraphic characters starting at the hexadecimal input character `#1` and the hexadecimal Unicode script character `#3`. Unicode does not distinguish calligraphic and script characters, so the same rows serve both alphabets. The row order determines the slots within each calligraphic and script alphabet of the virtual fonts.

```
\int_new:N \g__hep_math_cal_int
\cs_new_protected:Npn \__hep_math_cal:nnn #1#2#3 {
  \int_step_inline:nnn {0}{#2-1} {
    \tl_gput_right:Nx \g__hep_cal_setup_tl {
      \exp_not:N \hep_math_cal_map:nnn
        {\int_use:N \g__hep_math_cal_int}
        {\int_eval:n {\int_from_hex:n {#1}+##1}}
        {\int_eval:n {\int_from_hex:n {#3}+##1}}
    }
    \int_gincr:N \g__hep_math_cal_int
  }
}
```

Calligraphic capitals, of which B, E, F, H, I, L, M and R are letterlike symbols in Unicode. The CM symbol fonts and RSFS contain no small calligraphic and script letters.

```
\__hep_math_cal:nnn {0041}{1}{1D49C}
\__hep_math_cal:nnn {0042}{1}{212C}
\__hep_math_cal:nnn {0043}{2}{1D49E}
\__hep_math_cal:nnn {0045}{2}{2130}
\__hep_math_cal:nnn {0047}{1}{1D4A2}
\__hep_math_cal:nnn {0048}{1}{210B}
\__hep_math_cal:nnn {0049}{1}{2110}
\__hep_math_cal:nnn {004A}{2}{1D4A5}
\__hep_math_cal:nnn {004C}{1}{2112}
\__hep_math_cal:nnn {004D}{1}{2133}
\__hep_math_cal:nnn {004E}{4}{1D4A9}
\__hep_math_cal:nnn {0052}{1}{211B}
\__hep_math_cal:nnn {0053}{8}{1D4AE}
```

A.6 Modifiers

`\__hep_math_map:` Map all characters of the glyph table to the current state. The blackboard alphabet then replaces the Latin letters and digits, and the calligraphic or script alphabet the capitals.

```
\cs_new_protected:Npn \__hep_math_map: {
  \tl_use:N \g__hep_glyph_setup_tl
  \bool_if:NT \l__hep_bb_bool {
    \hep_math_bb_select:
    \tl_use:N \g__hep_bb_setup_tl
  }
  \bool_lazy_or:nnT {\l__hep_cal_bool}{\l__hep_scr_bool} {
    \hep_math_cal_select:
    \tl_use:N \g__hep_cal_setup_tl
  }
}
```

`\l__hep_state_tl` The name of the current state, and the name of the state whose math codes are in effect, which
`\l__hep_applied_tl` is empty if other code has changed the math codes since. Both are local like the math codes.
`\g__hep_cache_bool` Math codes are only cached in the document body, after all math families have been allocated.

```
\tl_new:N \l__hep_state_tl
\tl_new:N \l__hep_applied_tl
\bool_new:N \g__hep_cache_bool
```

`\hep_math_apply:` Reset the math family set by `\fam` and apply the current state to all characters of the glyph table. The math codes are local, so the formula is parsed as usual. Mapping a character requires several decisions, which only depend on the state, so the resulting math codes of each state are cached the first time it is used. Nothing has to be done if the math codes of the state are already in effect, which is the case for most formulas. The backend selection is cheap and also sets the variables used by the Greek letter macros.

```
\cs_new_protected:Npn \hep_math_apply: {
  \tex_fam:D -1\scan_stop:
  \hep_math_backend_select:
  \tl_set:Ne \l__hep_state_tl {
    \l__hep_family_tl
    \bool_if:NT \l__hep_bold_bool {-bold}
    \bool_if:NT \l__hep_up_bool {-up}
    \bool_if:NT \l__hep_bb_bool {-bb}
    \bool_if:NT \l__hep_cal_bool {-cal}
    \bool_if:NT \l__hep_scr_bool {-scr}
  }
  \tl_if_eq:NNF \l__hep_state_tl \l__hep_applied_tl {
    \bool_if:NTF \g__hep_cache_bool
      {\__hep_math_cached_map:}
      {\__hep_math_map:}
    \tl_set_eq:NN \l__hep_applied_tl \l__hep_state_tl
  }
}
```

`hep_math_cached_map:` Use the cached math codes of the current state, or map the characters and save their math codes.

```
\cs_new_protected:Npn \__hep_math_cached_map: {
  \tl_if_exist:cTF {g__hep_state_ \l__hep_state_tl _tl}
    {\tl_use:c {g__hep_state_ \l__hep_state_tl _tl}}
    {
      \__hep_math_map:
      \tl_gset:ce {g__hep_state_ \l__hep_state_tl _tl} {
        \tl_map_function:NN \g__hep_math_codes_tl
          \__hep_math_saved_code:n
      }
    }
}
```

`\__hep_math_native:` Restore the original math codes of the characters, which are no longer those of a state.

```
\cs_new_protected:Npn \__hep_math_native: {
  \tl_use:N \g__hep_native_codes_tl
  \tl_clear:N \l__hep_applied_tl
}
```

_hep_math_alphabet:n Set the flag #1 of the blackboard, calligraphic and script alphabets and clear the other two, or all three for an empty argument.

```
\cs_new_protected:Npn \__hep_math_alphabet:n #1 {
  \clist_map_inline:nn {bb, cal, scr} {
    \str_if_eq:nnTF {##1}{#1}
      {\bool_set_true:c {l__hep_##1_bool}}
      {\bool_set_false:c {l__hep_##1_bool}}
  }
}
```

\hep_math_modify:nn Apply the attributes in the list #1 to the math #2 within a group. Bold and upright accumulate, and the innermost family and the innermost of blackboard, calligraphic and script win. With mathsf=contrast sf selects the family opposite to the main family of the document, independent of the current family.

```
\cs_new_protected:Npn \hep_math_modify:nn #1#2 {
  \group_begin:
  \clist_map_inline:nn {#1} {
    \str_case:nn {##1} {
      {bold}{\bool_set_true:N \l__hep_bold_bool}
      {bb}{\__hep_math_alphabet:n {bb}}
      {cal}{\__hep_math_alphabet:n {cal}}
      {scr}{\__hep_math_alphabet:n {scr}}
      {up}{\bool_set_true:N \l__hep_up_bool}
      {it}{\bool_set_false:N \l__hep_up_bool}
      {mono}{\tl_set:Nn \l__hep_family_tl {mono}}
      {serif}{\tl_set:Nn \l__hep_family_tl {serif}}
      {sf}{
        \tl_set:Nn \l__hep_family_tl {sans}
        \bool_lazy_and:nnT
          {\str_if_eq_p:Vn \hepmathfont@mathsf {contrast}}
          {\str_if_eq_p:Vn \g__hep_main_family_tl {sans}}
          {\tl_set:Nn \l__hep_family_tl {serif}}
      }
    }
  }
  \hep_math_apply:
  #2
  \group_end:
}
```

\l__hep_native_bool Some packages typeset text in math mode and rely on the original math codes. This flag disables the initialisation of the math alphabets there.

```
\bool_new:N \l__hep_native_bool
```

hep_math_initialize: Each formula starts italic and without blackboard, calligraphic and script in the family and weight of the surrounding text. The bold math version also selects bold.

```
\cs_new_protected:Npn \hep_math_initialize: {
```

```
  \bool_if:NF \l__hep_native_bool {
    \tl_set_eq:NN \l__hep_family_tl \l__hep_text_family_tl
    \bool_set_eq:NN \l__hep_bold_bool \l__hep_text_bold_bool
    \str_if_eq:eeT {\math@version}{bold}
      {\bool_set_true:N \l__hep_bold_bool}
    \bool_set_false:N \l__hep_up_bool
    \__hep_math_alphabet:n {}
    \hep_math_apply:
  }
}
```

A.7 User commands

`__hep_math_commands:` Collect the definitions of the alphabet commands in a function. It is used when the package is loaded, so that copies made with `\let` in the preamble select the composable commands. UNICODE-MATH defines its own commands at the beginning of the document, therefore the function is used again afterwards, see below.

`\mathbf` `\mathup` `\mathrm` `\mathit` `\mathnormal` `\mathsf` `\mathtt` `\mathserif` Define the composable commands.

```
\cs_new_protected:Npn \__hep_math_commands: {
  \cs_gset_protected:Npn \mathbf ##1 {\hep_math_modify:nn {bold}{##1}}
  \cs_gset_protected:Npn \mathup ##1 {\hep_math_modify:nn {up}{##1}}
  \cs_gset_eq:NN \mathrm \mathup
  \cs_gset_protected:Npn \mathit ##1 {\hep_math_modify:nn {it}{##1}}
  \cs_gset_protected:Npn \mathnormal ##1
    {\group_begin:\hep_math_initialize: ##1\group_end:}
  \cs_gset_protected:Npn \mathsf ##1 {\hep_math_modify:nn {sf}{##1}}
  \cs_gset_protected:Npn \mathtt ##1 {\hep_math_modify:nn {mono}{##1}}
  \cs_gset_protected:Npn \mathserif ##1
    {\hep_math_modify:nn {serif}{##1}}
```

`\mathbfrm` `\mathbfup` `\mathbfsf` `\mathsfup` `\mathsftext` `\mathbfsfup` `\mathbftt` `\mathttup` `\mathbfttup` Define the combined commands.

```
  \cs_gset_protected:Npn \mathbfrm ##1
    {\hep_math_modify:nn {bold,up}{##1}}
  \cs_gset_eq:NN \mathbfup \mathbfrm
  \cs_gset_protected:Npn \mathbfsf ##1
    {\hep_math_modify:nn {bold,sf}{##1}}
  \cs_gset_protected:Npn \mathsfup ##1
    {\hep_math_modify:nn {sf,up}{##1}}
  \cs_gset_eq:NN \mathsftext \mathsfup
  \cs_gset_protected:Npn \mathbfsfup ##1
    {\hep_math_modify:nn {bold,sf,up}{##1}}
  \cs_gset_protected:Npn \mathbftt ##1
    {\hep_math_modify:nn {bold,mono}{##1}}
  \cs_gset_protected:Npn \mathttup ##1
    {\hep_math_modify:nn {mono,up}{##1}}
  \cs_gset_protected:Npn \mathbfttup ##1
    {\hep_math_modify:nn {bold,mono,up}{##1}}
}
```

Define the commands now. Without the virtual fonts of pdfTeX the standard commands defined above are kept.

```
\ifhepmathfont@vf
  \__hep_math_commands:
\else
  \cs_gset_eq:NN \__hep_math_commands: \prg_do_nothing:
\fi
```

Define the decorative alphabets, the Greek letter macros and again the alphabet commands at the beginning of the document, after UNICODE-MATH has defined its own commands. The code of this package for the hook `begindocument` runs before the code of packages loaded later and of the preamble, e.g. the code added with `\AtBeginDocument`. Copies of the commands made there, such as `\let\mat\mathbf`, therefore also select the composable commands. With the kernel alphabet commands they would select a font of the LaTeX kernel with the math codes of this package and therefore wrong glyphs. The ALPHABETA package also defines the Greek letter macros at the beginning of the document, so the code of this package follows its code and that of UNICODE-MATH.

```
\DeclareHookRule{begindocument}{hep-math-font}{after}{unicode-math}
\DeclareHookRule{begindocument}{hep-math-font}{after}{alphabeta}
\AddToHook{begindocument}{
  \tl_use:N \g__hep_greek_setup_tl
  \tl_use:N \g__hep_greek_user_tl
```

`\mathcal` `\mathscr` `\mathbb` `\mathfrak` The decorative alphabets restore the original math codes of the input characters within their argument. For `symbols=true` the blackboard alphabet and for `symbols=true` and `ams` the calligraphic and script alphabets are composable instead. Without the virtual fonts of pdfTeX they are kept.

```
  \ifhepmathfont@vf
    \clist_map_inline:nn {mathcal, mathscr, mathbb, mathfrak} {
      \cs_if_exist:cT {#1} {
        \cs_gset_eq:cc {hep_native_#1:}{#1}
        \cs_gset_protected:cpn {#1} ##1 {
          \group_begin:
          \__hep_math_native:
          \use:c {hep_native_#1:}{##1}
          \group_end:
        }
      }
    }
    \str_if_eq:VnT \hepmathfont@symbols {true}
      {\cs_gset_protected:Npn \mathbb #1 {\hep_math_modify:nn {bb}{#1}}}
    \exp_args:NnV \clist_if_in:nnT {true, ams} \hepmathfont@symbols {
      \cs_gset_protected:Npn \mathcal #1 {\hep_math_modify:nn {cal}{#1}}
      \cs_gset_protected:Npn \mathscr #1 {\hep_math_modify:nn {scr}{#1}}
    }
  \fi
```

Define the alphabet commands.

```
  \__hep_math_commands:
}
```

Determine the main family again at the end of the beginning of the document. The math fonts were selected for the main family detected when the package was loaded, so warn if it has changed since. The Greek letter macros are not defined again here, so that redefinitions with `\AtBeginDocument` are kept; HEP-FONT does not wrap them when this package is loaded.

```
\AddToHook{begindocument/end}{
  \tl_set_eq:NN \l_tmpa_tl \g__hep_main_family_tl
  \hep_math_detect_family:
  \tl_if_eq:NNF \l_tmpa_tl \g__hep_main_family_tl {
    \PackageWarningNoLine{hep-math-font}{
      The~main~family~changed~from~\l_tmpa_tl\space
      to~\g__hep_main_family_tl\space
      after~loading~the~package.\MessageBreak
      Load~hep-font~and~set~\protect\familydefault\space
      before~hep-math-font
    }
  }
  \tl_set_eq:NN \l__hep_text_family_tl \g__hep_main_family_tl
```

Initialise every inline and display formula. Also initialise the document itself, so that the state is defined when a document replaces `\everymath` or uses an alphabet command outside math. Without the virtual fonts of pdfTEX the formulas keep their math codes.

```
  \ifhepmathfont@vf
    \everymath\expandafter{\the\everymath\hep_math_initialize:}
    \everydisplay\expandafter{\the\everydisplay\hep_math_initialize:}
    \bool_gset_true:N \g__hep_cache_bool
    \hep_math_initialize:
  \fi
}
```

A.8 Text state

Follow the family and weight declarations of the text, but not within math. The assignments are local, so that headings and groups do not change later formulas.

```
\AddToHook{cmd/rmfamily/after}
  {\mode_if_math:F{\tl_set:Nn\l__hep_text_family_tl{serif}}}
\AddToHook{cmd/sffamily/after}
  {\mode_if_math:F{\tl_set:Nn\l__hep_text_family_tl{sans}}}
\AddToHook{cmd/ttfamily/after}
  {\mode_if_math:F{\tl_set:Nn\l__hep_text_family_tl{mono}}}
\AddToHook{cmd/bfseries/after}
  {\mode_if_math:F{\bool_set_true:N\l__hep_text_bold_bool\boldmath}}
\AddToHook{cmd/mdseries/after}
  {\mode_if_math:F{\bool_set_false:N\l__hep_text_bold_bool\unboldmath}}
\AddToHook{cmd/normalfont/after}{
  \mode_if_math:F{
    \tl_set_eq:NN\l__hep_text_family_tl\g__hep_main_family_tl
```

```
    \bool_set_false:N\l__hep_text_bold_bool\unboldmath
  }
}
```

The AMSMATH package groups operator names such as `\sin` before it selects the operator font. Restore the original math codes there, so that operator names use the operator font.

```
\AddToHook{cmd/operator@font/after}{\__hep_math_native:}
```

The URL package typesets its argument in math mode with the current text font as family 0. Its letters and digits select this family through their original math codes. Restore these codes and keep the formula from initialising the math alphabets. With the Unicode engines the list of original math codes is empty, and UNICODE-MATH switches to its literal math codes before the formula starts. The flag is local and ends with the group of the URL.

```
\AddToHook{cmd/Url@MathSetup/after}{
  \bool_set_true:N \l__hep_native_bool
  \__hep_math_native:
}
```

A.9 Bookmarks

`\hep_math_pdfstring:` The HYPERREF package [5] turns sectioning titles into bookmarks by expansion and removes the math shifts. It does not expand the protected alphabet commands and removes them with a warning. Keep their argument instead, and also that of the dashed and dotted vector accents.

```
\cs_new_protected:Npn \hep_math_pdfstring: {
  \clist_map_inline:nn {
    mathbf, mathup, mathrm, mathit, mathnormal, mathsf, mathtt,
    mathserif, mathbfrm, mathbfup, mathbfsf, mathsfup, mathsftext,
    mathbfsfup, mathbftt, mathttup, mathbfttup, mathbb, mathcal,
    mathscr, mathfrak, bm, dashedvec, dottedvec
  }{\cs_set_eq:cN {##1}\use:n}
```

Let the Greek letters, the derivatives, the dotless letters, the ell and the h-bar stand for text commands, which HYPERREF turns into Unicode characters.

```
  \tl_use:N \g__hep_greek_pdfstring_tl
  \cs_set_eq:NN \partial \textpartial
  \cs_set_eq:NN \nabla \textnabla
  \cs_set_eq:NN \imath \i
  \cs_set_eq:NN \jmath \j
  \cs_set_eq:NN \ell \textell
  \cs_set_eq:NN \hbar \texthbar
}
```

Register the definitions with HYPERREF, whether it is loaded before or after this package.

```
\cs_new_protected:Npn \hep_math_pdfstring_setup:
  {\pdfstringdefDisableCommands{\hep_math_pdfstring:}}
\IfPackageLoadedTF{hyperref}{\hep_math_pdfstring_setup:}
  {\AddToHook{package/hyperref/after}{\hep_math_pdfstring_setup:}}
```

_pdfstring_variants: The Unicode strings distinguish the epsilon and phi symbols and the theta symbol from the
dfstring_variant:nnn ordinary letters. Other packages let these macros stand for the ordinary letters, so register the definitions at the beginning of the document after theirs. Only letters that stand for the ordinary text letters are changed, so letters that have been redefined keep their new meaning.

```
\cs_new_protected:Npn \hep_math_pdfstring_variants: {
  \legacy_if:nT {Hy@unicode} {
    \__hep_math_pdfstring_variant:nnn {epsilon}{epsilon}{epsilonsymbol}
    \__hep_math_pdfstring_variant:nnn {phi}{phi}{phisymbol}
    \__hep_math_pdfstring_variant:nnn {varTheta}{Theta}{Thetasymbol}
  }
}
\cs_new_protected:Npn \__hep_math_pdfstring_variant:nnn #1#2#3 {
  \cs_if_eq:ccT {#1}{text#2}
    {\cs_set_eq:cc {#1}{text#3}}
}
\AddToHook{begindocument/before}{
  \IfPackageLoadedT{hyperref}
    {\pdfstringdefDisableCommands{\hep_math_pdfstring_variants:}}
}
\ExplSyntaxOff
```

</package>

B Virtual font generator

The TFM and VF files of the virtual fonts are generated by a Python script, which uses the fontTools library to read the metrics of the installed fonts.

<*generator>

Describe the script and import the required modules.

```
# Copyright (C) 2019-2026 by Jan Hajer
# This file may be distributed and/or modified under the conditions of
# the LaTeX Project Public License, either version 1.3c of this license
# or (at your option) any later version.
"""Build the pdfTeX virtual fonts of the hep-math-font package.

Run this script in the directory containing hep-math-font.sty.
It reads the glyph tables from that file and writes the hepalpha,
hepdecor, hepscript and hepbb TFM and VF files there, with the TFM files
of their re-encoded and slanted base fonts, the TrueType font of the
accents, two encodings, two map files and hep-math-font-vf.def, which
records the version of the package file.
Requires Python 3 with fontTools, kpsewhich, pltotf and vptovf.
"""
from functools import lru_cache
from pathlib import Path
import copy
import math
```

```
import re
import shutil
import statistics
import subprocess
import tempfile
from fontTools.fontBuilder import FontBuilder
from fontTools.misc.bezierTools import splitCubic, splitLine
from fontTools.pens.boundsPen import BoundsPen
from fontTools.pens.cu2quPen import Cu2QuPen
from fontTools.pens.recordingPen import RecordingPen
from fontTools.pens.transformPen import TransformPen
from fontTools.pens.ttGlyphPen import TTGlyphPen
from fontTools.t1Lib import T1Font
from fontTools.tfmLib import TFM
from fontTools.ttLib import TTFont, newTable
from fontTools.ttLib.tables import otTables

ROOT = Path.cwd()
TEMP = Path(tempfile.mkdtemp())
```

Locate installed files with KPSEWHICH and read the metrics of the generated and installed fonts and the outlines of the installed Type 1 fonts.

```
def kpse(name):
    """Return the path of an installed file."""
    path = subprocess.run(['kpsewhich', name], text=True,
                          stdout=subprocess.PIPE).stdout.strip()
    if not path:
        raise SystemExit('Missing installed file: ' + name)
    return path


@lru_cache(None)
def tfm(name):
    """Return the metrics of a generated or installed TFM file."""
    path = ROOT / (name + '.tfm')
    return TFM(str(path) if path.exists() else kpse(name + '.tfm'))


@lru_cache(None)
def type1(name):
    """Return the parsed Type 1 font of an installed PFB file."""
    font = T1Font(kpse(name + '.pfb'))
    font.parse()
    return font
```

The map file of pdfTEX names the Type 1 font and the encoding of each installed font. Without an encoding file the font uses its built-in encoding. The fonts generated by this script use their own encoding.

```
@lru_cache(None)
def maplines():
```

```
    """Return the lines of the pdfTeX map file by font name."""
    lines = Path(kpse('pdftex.map')).read_text().splitlines()
    return {line.split()[0]: line for line in lines
            if line.strip() and not line.startswith(('%', '#'))}


def mapfiles(name):
    """Return the font and encoding files of a font."""
    if name in REENCODED:
        return REENCODED[name], None
    if name not in maplines():
        raise SystemExit('Font missing from pdftex.map: ' + name)
    files = re.findall(r'<[<\[]?(\S+)', maplines()[name])
    pfb = next(file for file in files if file.endswith('.pfb'))
    enc = [file for file in files if file.endswith('.enc')]
    return pfb[:-4], enc[0] if enc else None


@lru_cache(None)
def encoding(name):
    """Return the glyph names of the slots of a font."""
    if name in REENCODED:
        return GLYPHS
    pfb, enc = mapfiles(name)
    if enc is None:
        return list(type1(pfb).font['Encoding'])
    text = re.sub(r'%.*', '', Path(kpse(enc)).read_text())
    return re.findall(r'/([^\s/\[\]]+)', text.split('[', 1)[1])
```

The horizontal extent of a glyph, optionally slanted by a shear, determines the metrics of the slanted glyphs.

```
def bounds(pfb, glyph, slant=0):
    """Return the horizontal extent of a glyph, optionally slanted."""
    glyphs = type1(pfb).getGlyphSet()
    pen = BoundsPen(glyphs)
    glyphs[glyph].draw(TransformPen(pen, (1, 0, slant, 1, 0, 0)))
    box = pen.bounds or (0, 0, 0, 0)
    return box[0] / 1000, box[2] / 1000
```

B.1 Glyph tables

Read the glyph tables from the package, so that the slots are defined only once. The rows are expanded in the same way as by `\__hep_math_range:nnnnn`, `\__hep_math_named:nnnnn`, `\__hep_math_decor:nnn`, `\__hep_math_bb:nnn` and `\__hep_math_cal:nnn`. The accents of the rows `\__hep_math_accent:nnn` take the slots after the glyph table. Every call of a row macro must be parsed, otherwise the slots of the package and the virtual fonts would differ.

```
ARG = r'\s*\{([^}]*)\}'
ROW = re.compile(r'\\__hep_math_(range|named):nnnnn' + 5 * ARG)
DECOR = re.compile(r'\\__hep_math_decor:nnn' + 3 * ARG)
```

```
BB = re.compile(r'\\__hep_math_bb:nnn' + 3 * ARG)
CAL = re.compile(r'\\__hep_math_cal:nnn' + 3 * ARG)
ACCENT = re.compile(r'\\__hep_math_accent:nnn' + 3 * ARG)


def table_rows(pattern, sty, *names):
    """Return the parsed rows, which must be all calls of the macros."""
    found = pattern.findall(sty)
    calls = sum(len(re.findall(re.escape(name) + r'\s*\{', sty))
                for name in names)
    if len(found) != calls:
        raise SystemExit(f'Parsed {len(found)} of {calls} rows of '
                         + ', '.join(names))
    return found


def glyph_table(sty):
    """Expand the glyph rows exactly as the package does."""
    glyphs = []
    table = table_rows(ROW, sty, r'\__hep_math_range:nnnnn',
                       r'\__hep_math_named:nnnnn')
    for kind, names, *columns in table:
        start = [int(column, 16) for column in columns]
        if kind == 'range':
            items = [('', '')] * int(names)
        else:
            items = []
            for item in names.split(','):
                parts = [part.strip() for part in item.split('/')]
                items.append((parts[0], parts[-1]))
        for step, (name, text) in enumerate(items):
            glyphs.append(dict(index=len(glyphs), name=name, text=text,
                               unicode=[cp + step for cp in start]))
    assert len(glyphs) < 127, 'slot 127 + 128 = 255 is the skewchar'
    return glyphs


def decor_table(sty):
    """Expand the decorative rows to (alphabet, character, slot)."""
    rows = []
    table = table_rows(DECOR, sty, r'\__hep_math_decor:nnn')
    for alphabet, first, count in table:
        for step in range(int(count)):
            rows.append((alphabet, int(first, 16) + step, len(rows)))
    return rows


def bb_table(sty):
    """Expand the blackboard rows to input and Unicode characters."""
    chars = []
```

```
    table = table_rows(BB, sty, r'\__hep_math_bb:nnn')
    for first, count, unicode in table:
        chars += [(int(first, 16) + step, int(unicode, 16) + step)
                  for step in range(int(count))]
    assert 4 * len(chars) < 256, 'slot 255 is the skewchar'
    return chars

def cal_table(sty):
    """Expand the calligraphic rows to input characters."""
    table = table_rows(CAL, sty, r'\__hep_math_cal:nnn')
    return [int(first, 16) + step for first, count, _ in table
            for step in range(int(count))]


def accent_table(sty, glyphs):
    """Return the accents with their slot, pattern and characters."""
    table = table_rows(ACCENT, sty, r'\__hep_math_accent:nnn')
    accents = [dict(slot=len(glyphs) + index, pattern=pattern,
                    unicode=int(first, 16))
               for index, (_, pattern, first) in enumerate(table)]
    assert len(glyphs) + len(accents) <= 127, 'slot 128 is upright'
    return accents
```

B.2 Source slots

The local Greek (LGR) encoding addresses each Greek letter by an ASCII letter. The OML encoding only contains the Greek capitals which differ from the Latin ones, the others use the Latin capitals of the same font. The letters are identified by the names of their text commands.

```
GREEK = ('Alpha Beta Gamma Delta Epsilon Zeta Eta Theta Iota Kappa '
         'Lambda Mu Nu Xi Omicron Pi Rho Sigma Tau Upsilon Phi Chi Psi '
         'Omega')
LGR = dict(zip(GREEK.split(), 'ABGDEZHJIKLMNXOPRSTUFQYW'))
LGR.update((name.lower(), letter.lower()) for name, letter in
           list(LGR.items()))
LGR['finalsigma'] = 'c'
OML = dict(zip('Gamma Delta Theta Lambda Xi Pi Sigma Upsilon Phi Psi '
               'Omega'.split(), range(11)))
OML.update(zip('Alpha Beta Epsilon Zeta Eta Iota Kappa Mu Nu Omicron '
               'Rho Tau Chi'.split(), map(ord, 'ABEZHIKMNOPTX')))
OML.update(zip('alpha beta gamma delta varepsilon zeta eta theta iota '
               'kappa lambda mu nu xi pi rho sigma tau upsilon varphi '
               'chi psi omega'.split(), range(11, 34)))
OML.update(epsilon=34, phi=39, omicron=111, finalsigma=38)
```

The variant letters, derivatives, dotless letters, the ell and the h-bar are identified by their macro names. Their kind determines where the upright, typewriter and bold shapes come from.

```
SPECIAL = dict(epsilon=('cmu', 15), vartheta=('serifcmu', 35),
               phi=('serifcmu', 30), varrho=('serifcmu', 37),
               varpi=('variant', 36), partial=('variant', 64),
```

```
1207                varkappa=('kappa', 123), nabla=('nabla', 114),
1208                imath=('dotless', 123), jmath=('dotless', 124),
1209                ell=('ell', 96), hbar=('hbar', None))
1210
1211
1212 def classify(glyph):
1213     """Return the kind of a glyph and its source slot(s)."""
1214     if glyph['name'] in SPECIAL:
1215         return SPECIAL[glyph['name']]
1216     if glyph['name']:
1217         text = glyph['text']
1218         return 'greek', dict(lgr=ord(LGR[text]), oml=OML[text])
1219     up = glyph['unicode'][0]
1220     return ('digit' if up < 65 else 'latin'), up
```

B.3 Base fonts

The CM Unicode fonts provide the bold typewriter Greek, the upright and typewriter epsilon symbol, the upright serif theta, theta symbol, phi symbol and rho symbol, which are not part of the LGR encoding. They distinguish the Greek Delta, Omega and mu from the symbols of the same name by a suffix. The LM text fonts provide the h-bar. These glyphs form the encoding of the base fonts generated by this script.

```
1221 CMU_GLYPHS = {name: name for name in GREEK.split() + GREEK.lower().split()}
1222 CMU_GLYPHS.update(Delta='Delta.greek', Omega='Omega.greek',
1223                   mu='mu.greek', finalsigma='sigma1')
1224 CMU_VARIANTS = dict(epsilon='uni03F5', vartheta='theta1', phi='phi1',
1225                     varrho='uni03F1')
1226 GLYPHS = [*CMU_GLYPHS.values(), *CMU_VARIANTS.values(), 'theta.ipa',
1227           'hbar']
1228 ENCODING = 'hep-math-font'
1229 MAPS = {}
1230 REENCODED = {}
```

Write the TFM file of a base font, which takes a CM Unicode or LM text font in this encoding, and record its map line. The metrics are computed from the outlines and the italic correction of an italic font covers the overhang of its glyphs. The CM Unicode fonts have the design size 10 pt, the LM fonts state it in their names.

```
1231 def pl_value(value):
1232     """Return a property list real number."""
1233     return f'R {value:.9f}'
1234
1235
1236 @lru_cache(None)
1237 def reencoded(pfb):
1238     """Return the name of a base font of a PFB file in our encoding."""
1239     name = 'hep-' + pfb
1240     font = type1(pfb)
1241     glyphs = font.getGlyphSet()
1242     size = re.search(r'\d*$', pfb).group()
```

```
    italic = font.font['FontInfo'].get('ItalicAngle', 0) != 0
    pl = [f'(DESIGNSIZE R {size or 10})',
          f'(CODINGSCHEME {ENCODING.upper()})',
          '(FONTDIMEN (SLANT R 0.0) (QUAD R 1.0))']
    for slot, glyph in enumerate(GLYPHS):
        if glyph not in glyphs:
            continue
        box = BoundsPen(glyphs)
        glyphs[glyph].draw(box)
        x0, y0, x1, y1 = [value / 1000 for value in
                          box.bounds or (0, 0, 0, 0)]
        width = glyphs[glyph].width / 1000
        pl.append(f'(CHARACTER D {slot} (CHARWD {pl_value(width)}) '
                  f'(CHARHT {pl_value(max(y1, 0))}) '
                  f'(CHARDP {pl_value(max(-y0, 0))}) '
                  f'(CHARIC {pl_value(max(x1 - width, 0) * italic)}))')
    path = TEMP / (name + '.pl')
    path.write_text('\n'.join(pl) + '\n')
    subprocess.run(['pltotf', str(path), str(ROOT / (name + '.tfm'))],
                   check=True, stdout=subprocess.DEVNULL)
    line = (f'{name} {font.font["FontName"]} " {ENCODING.upper()}'
            f' ReEncodeFont " <{ENCODING}.enc <{pfb}.pfb')
    MAPS[name] = (line, line)
    REENCODED[name] = pfb
    return name
```

Synthetic shapes are slanted copies of installed or generated fonts. Their TFM file is a copy of the original one and their map line slants the font by the `SlantFont` operator. The map line of the fallback map file uses the font unchanged.

```
def slant_of(name):
    """Return the slant of a font."""
    return tfm(name).fontdimens['SLANT']


SLANTED = {}


def slanted(base, slant):
    """Return the name of a slanted copy of a font."""
    name = 'hepslant-' + base
    if name not in SLANTED:
        SLANTED[name] = (base, slant)
        path = ROOT / (base + '.tfm')
        if base not in REENCODED:
            path = kpse(base + '.tfm')
        shutil.copyfile(path, ROOT / (name + '.tfm'))
        line = (MAPS[base][0] if base in REENCODED
                else maplines()[base]).split(None, 1)[1]
        if '"' in line:
            slant_line = line.replace('"', f'" {slant:.4f} SlantFont', 1)
```

```
        else:
            psname, files = line.split(None, 1)
            slant_line = f'{psname} " {slant:.4f} SlantFont " {files}'
        MAPS[name] = (f'{name} {slant_line}', f'{name} {line}')
    assert SLANTED[name] == (base, slant), name
    return name
```

The LM text fonts exist at several optical sizes, choose the closest one.

```
def optical(stem, size):
    """Return the installed optical size of a font closest to size."""
    for candidate in [*range(size, 4, -1), *range(size + 1, 18)]:
        if subprocess.run(['kpsewhich', f'{stem}{candidate}.pfb'],
                          stdout=subprocess.DEVNULL).returncode == 0:
            return stem + str(candidate)
    raise SystemExit('Missing installed font: ' + stem)
```

B.4 Source fonts

The CM Unicode and LM text fonts for each family, weight and shape, where the shape is 1 for upright. CM Unicode has no italic serif and sans-serif fonts, the math italic fonts are used instead. The family 3 is the proportional typewriter.

```
CMU = {(0, 0, 1): 'cmunrm', (0, 1, 1): 'cmunbx', (1, 0, 1): 'cmunss',
       (1, 1, 1): 'cmunsx', (2, 0, 1): 'cmuntt', (2, 1, 1): 'cmuntb',
       (2, 0, 0): 'cmunit', (2, 1, 0): 'cmuntx'}
HBAR = {(0, 0, 1): 'lmr', (0, 1, 1): 'lmbx', (0, 0, 0): 'lmri',
        (0, 1, 0): 'lmbxi', (1, 0, 1): 'lmss', (1, 1, 1): 'lmssbx',
        (1, 0, 0): 'lmsso', (1, 1, 0): 'lmssbo', (2, 0, 1): 'lmtt',
        (2, 1, 1): 'lmtk', (2, 0, 0): 'lmtti', (2, 1, 0): 'lmtko',
        (3, 0, 1): 'lmvtt', (3, 1, 1): 'lmvtk', (3, 0, 0): 'lmvtto',
        (3, 1, 0): 'lmvtko'}
```

Return the source font and slot for a glyph in the family 0, 1 or 2 for serif, sans-serif or typewriter, with the given weight, shape and size. The family 3 for the proportional typewriter only differs in its Latin letters, digits and h-bar, and is otherwise treated as the typewriter family. The bold math italic fonts only exist at 5 pt, 7 pt and 10 pt. Most variant Greek letters of the CM Unicode fonts only exist in serif.

```
def source(kind, slot, glyph, family, bold, up, size):
    """Return the source font and slot of a glyph."""
    mathsize = size if not bold else (5 if size == 5 else
                                      7 if size == 7 else 10)
    mi = ('lmmib' if bold else 'lmmi') + str(mathsize)
    sfmi = ('cmssmib' if bold else 'cmssmi') + str(size)
    italic = sfmi if family == 1 else mi
    prop = family == 3
    family = min(family, 2)

    def cmu(family, up):
        font = reencoded(CMU[family, bold, up])
        if glyph['name'] in CMU_VARIANTS:
```

```
            return font, GLYPHS.index(CMU_VARIANTS[glyph['name']])
        if glyph['name'] == 'theta' and family == 0:
            return font, GLYPHS.index('theta.ipa')
        return font, GLYPHS.index(CMU_GLYPHS[glyph['text']])
```

CM has a single upright nabla for each family and weight, the typewriter family uses the serif one. The italic nabla is slanted by the slant of the math italic font.

```
    if kind == 'nabla':
        if family == 1:
            font = ('cmssbsy' if bold else 'cmsssy') + str(size)
        else:
            font = ('lmbsy' if bold else 'lmsy') + str(mathsize)
        return (font if up else slanted(font, slant_of(italic))), slot
```

The math italic fonts provide the italic variant letters and derivatives. The upright varpi and partial and the upright sans-serif phi, rho and theta symbols are the italic shapes slanted back. The typewriter family uses the serif shapes. The other upright and typewriter shapes come from CM Unicode.

```
    if kind == 'variant' or (kind == 'serifcmu'
                             and (family == 1 or not up)):
        font = sfmi if family == 1 else mi
        return (slanted(font, -slant_of(font)) if up else font), slot
    if kind == 'serifcmu':
        return cmu(0, 1)
    if kind == 'cmu':
        if up or family == 2:
            return cmu(family, up)
        return italic, slot
```

The kappa symbol comes from the AMS blackboard fonts, whose sans-serif version is part of the SANSMATHFONTS package. There is no bold kappa symbol. The ell of the math italic fonts serves both shapes and the typewriter family uses the serif one. The h-bar comes from the LM text fonts.

```
    if kind == 'kappa':
        font = ('ssmsbm' if family == 1 else 'msbm') + str(size)
        return (slanted(font, -slant_of(italic)) if up else font), slot
    if kind == 'ell':
        return italic, slot
    if kind == 'hbar':
        stem = HBAR[family + prop, bold, up]
        return reencoded(optical(stem, size)), GLYPHS.index('hbar')
```

Italic serif and sans-serif Greek is taken from the OML math fonts. Upright and typewriter Greek is taken from the CB Greek fonts. Their upright serif theta has the shape of the theta symbol, so it comes from CM Unicode instead. Their italic typewriter epsilon has the shape of the epsilon symbol, so it is the upright epsilon slanted. The bold typewriter Greek comes from CM Unicode as the CB Greek typewriter fonts have no bold weight.

```
    if kind == 'greek':
        if family != 2 and not up:
```

```
            return italic, slot['oml']
        if family == 0 and glyph['name'] == 'theta':
            return cmu(0, 1)
        if family == 2 and not up and glyph['name'] == 'varepsilon':
            font, cp = source(kind, slot, glyph, family, bold, 1, size)
            return slanted(font, slant_of(f'gtti{size:02d}00')), cp
        if family == 2:
            if bold:
                return cmu(2, up)
            font = ('gttn' if up else 'gtti') + f'{size:02d}00'
            return font, slot['lgr']
        if family == 1:
            prefix = 'gsxn' if bold else 'gsmn'
        else:
            prefix = 'grxn' if bold else 'grmn'
        return prefix + f'{size:02d}00', slot['lgr']
```

Latin letters and digits are taken from the LM text and math fonts. The proportional typewriter has no italic, its oblique fonts are used instead. Digits are always upright. The upright dotless letters are taken from the text fonts.

```
    if kind == 'digit':
        up = True
    if family == 0:
        lm = 'rm-lmbx' if bold else 'rm-lmr'
        font = lm + str(size) if up else mi
    elif family == 1:
        lm = 'rm-lmssbx10' if bold else 'rm-lmss' + str(max(8, size))
        font = lm if up else sfmi
    elif prop:
        lm = 'rm-lmvtk' if bold else 'rm-lmvtt'
        font = lm + ('10' if up else 'o10')
    elif bold:
        font = 'rm-lmtk10' if up else 'rm-lmtko10'
    else:
        font = 'rm-lmtt' + str(max(8, size)) if up else 'rm-lmtti10'
    if kind == 'dotless' and (up or family == 2):
        slot = 16 if glyph['name'] == 'imath' else 17
    return font, slot
```

B.5 Virtual property lists

A character copies the width, height, depth and italic correction of its source. A slanted glyph keeps its left side bearing. Slanted back to upright it keeps the right side bearing including the italic correction, slanted to italic the overhang becomes its italic correction.

```
METRICS = [('CHARWD', 'width'), ('CHARHT', 'height'),
           ('CHARDP', 'depth'), ('CHARIC', 'italic')]


def slant_metrics(name, cp, metrics):
```

```
    """Return the metrics and offset of a slanted glyph."""
    base, slant = SLANTED[name]
    pfb = mapfiles(base)[0]
    glyph = encoding(base)[cp]
    x0, x1 = bounds(pfb, glyph)
    s0, s1 = bounds(pfb, glyph, slant)
    move = x0 - s0
    right = metrics['width'] + metrics.get('italic', 0) - x1
    metrics = dict(metrics, italic=0)
    if slant < 0:
        metrics['width'] = s1 + move + right
    else:
        metrics['italic'] = max(s1 + move + right - metrics['width'], 0)
    return metrics, move
```

Synthesise a bold glyph by overprinting copies of the glyph, which are shifted around a circle of the radius `BOLD` in units of the font size. The copies are a marked-content sequence with an empty actual text, so that the text extracted from the PDF file contains the glyph once. The sequence ends the text object before it begins and before it ends, so that both are properly nested.

```
BOLD = 0.012


def overprint(glyph, radius):
    """Return the VPL commands which embolden a glyph."""
    copies = ' '.join(
        f'(PUSH) (MOVERIGHT R {radius * (1 + math.cos(angle)):.9f}) '
        f'(MOVEUP R {radius * math.sin(angle):.9f}) {glyph} (POP)'
        for angle in (math.pi * step / 8 for step in range(16)))
    return ('(SPECIAL pdf:page:/Span<</ActualText()>>BDC) '
            + copies + ' (SPECIAL pdf:page:EMC) '
            f'(MOVERIGHT R {radius:.9f}) {glyph}')
```

Write a character of the virtual font, which moves a slanted glyph to keep its left side bearing. Source fonts listed in `SCALE` are scaled together with their metrics, and the radius `bold` widens a glyph on both sides and raises its height and depth. Source fonts listed in `PROXY` take their metrics from another font.

```
SCALE = {}
PROXY = {}


def character(slot, fonts, name, cp, bold=0):
    """Return a VPL character copying the metrics of its source."""
    metrics = tfm(PROXY.get(name, name)).chars.get(cp)
    assert metrics is not None, (name, cp)
    move = 0
    if name in SLANTED:
        metrics, move = slant_metrics(name, cp, metrics)
    metrics = {key: value * SCALE.get(name, 1)
               for key, value in metrics.items()}
    for key in ('width', 'height', 'depth'):
```

```
        metrics[key] = metrics.get(key, 0) + bold * (2 if key == 'width'
                                                     else 1)
    fields = ' '.join(f'({key} R {metrics.get(value, 0):.9f})'
                      for key, value in METRICS)
    shift = f'(MOVERIGHT R {move:.9f}) ' if move else ''
    glyph = f'(SELECTFONT D {fonts.index(name)}) (SETCHAR D {cp})'
    if bold:
        glyph = overprint(glyph, bold)
    return f'(CHARACTER D {slot} {fields} (MAP {shift}{glyph}))'
```

The source fonts are used at their design sizes, scaled to the size of the virtual font and by their factor in `SCALE`.

```
def mapfonts(fonts):
    """Return the VPL font references at their design sizes."""
    return [f'(MAPFONT D {i} (FONTNAME {name}) '
            f'(FONTAT R {SCALE.get(name, 1.0)}) '
            f'(FONTDSIZE R {tfm(name).designsize}))'
            for i, name in enumerate(fonts)]
```

Math accents are placed using the kerns with the skew character. Redirect them to the empty slot 255, which the package declares as the skew character of the virtual fonts. Glyphs slanted to italic place the accent above the middle of their height.

```
def skew_kerns(rows):
    """Redirect accent skew kerns to the empty slot 255."""
    kerns = [f'(LABEL D {slot}) (KRN D 255 R {skew:.9f}) (STOP)'
             for slot, skew in rows if skew]
    return ['(LIGTABLE ' + ' '.join(kerns) + ')'] if kerns else []


def skew(name, cp):
    """Return the accent skew of a character of an italic source."""
    if name in SLANTED:
        base, slant = SLANTED[name]
        return max(slant, 0) * tfm(base).chars[cp]['height'] / 2
    if name.startswith(('lmmi', 'cmssmi')):
        return tfm(name).kerning.get(cp, {}).get(127, 0)
    return 0
```

Convert the property list with VPTOVF, keeping only the TFM and VF files.

```
def write(vpl, name):
    """Convert the VPL lines to TFM and VF files next to the package."""
    path = TEMP / (name + '.vpl')
    path.write_text('\n'.join(vpl) + '\n')
    subprocess.run(['vptovf', str(path), str(ROOT / (name + '.vf')),
                    str(ROOT / (name + '.tfm'))], check=True,
                   stdout=subprocess.DEVNULL)
```

B.6 Accents

The dashed and dotted vector accents are copies of the vector accents of the math italic fonts for pdfTEX and of the NCM math fonts for the Unicode engines, whose shafts are cut into dashes or dots. The arrowhead, the width, the height and the accent attachment are those of the original glyph, so that the accents are placed exactly like `\vec`. The glyphs are collected in the TrueType font hepaccent.ttf, which pdfTEX uses through the TFM file hepaccent.tfm and the `hepalpha` virtual fonts, and the Unicode engines through its private use characters and its math table. Each source font provides one glyph per accent. The bold and sans-serif math italic fonts only exist at the three sizes of the virtual fonts.

```
ACCENT_FONT = 'hepaccent'
ACCENT_SOURCES = [stem + str(size) for stem in
                  ('lmmi', 'lmmib', 'cmssmi', 'cmssmib')
                  for size in (5, 7, 10)]
ACCENT_UNICODE = ['NewCMMath-Regular', 'NewCMSansMath-Regular',
                  'NewCMMath-Bold']


def accent_source(family, bold, size):
    """Return the math italic font of a bank, whose accent is used."""
    return ('cmssmi' if family == 1 else 'lmmi') + 'b' * bold + str(size)
```

A glyph is a list of contours, each a list of line and curve segments given by their two or four points. The outlines are read with their original cubic curves.

```
def contours(glyph):
    """Return the contours of a glyph as lists of segments."""
    pen = RecordingPen()
    glyph.draw(pen)
    result = []
    for operator, points in pen.value:
        if operator == 'moveTo':
            start = current = points[0]
            result.append([])
        elif operator in ('lineTo', 'curveTo'):
            result[-1].append((current, *points))
            current = points[-1]
        elif operator in ('closePath', 'endPath'):
            if current != start:
                result[-1].append((current, start))
        else:
            raise SystemExit('Unexpected outline: ' + operator)
    return result


def at(segment, t):
    """Return the point at the parameter t of a segment."""
    if len(segment) == 2:
        weights = (1 - t, t)
    else:
        weights = ((1 - t) ** 3, 3 * t * (1 - t) ** 2,
```

```
                    3 * t * t * (1 - t), t ** 3)
    return tuple(sum(w * p[i] for w, p in zip(weights, segment))
                 for i in (0, 1))


def samples(contour):
    """Return points along a contour."""
    return [at(segment, step / 32) for segment in contour
            for step in range(32)]
```

The vector accents consist of a single contour: a horizontal shaft with a round left end and an arrowhead, whose barbs reach back over the shaft. The shaft is measured in the left part of the glyph and the arrowhead starts where the outline leaves the shaft.

```
def shaft(contour):
    """Return the left end, top, bottom and head start of a shaft."""
    points = samples(contour)
    left = min(x for x, _ in points)
    right = max(x for x, _ in points)
    middle = [y for x, y in points
              if left + .15 * (right - left) < x
              < left + .35 * (right - left)]
    top, bottom = max(middle), min(middle)
    margin = (top - bottom) / 4
    head = min(x for x, y in points
               if y > top + margin or y < bottom - margin)
    return left, top, bottom, head
```

Cut a contour at the vertical line `cut`, keep its right part and close it by a round end on the left. The line has to cross the contour twice, at the top and the bottom of the shaft. A circular arc is drawn by two cubic curves per quarter.

```
KAPPA = 4 * (math.sqrt(2) - 1) / 3


def quarter(start, end, horizontal):
    """Return a quarter circle leaving start horizontally or not."""
    (x0, y0), (x1, y1) = start, end
    if horizontal:
        return (start, (x0 + KAPPA * (x1 - x0), y0),
                (x1, y1 - KAPPA * (y1 - y0)), end)
    return (start, (x0, y0 + KAPPA * (y1 - y0)),
            (x1 - KAPPA * (x1 - x0), y1), end)


def split(segment, cut):
    """Return the pieces of a segment on both sides of x = cut."""
    if len(segment) == 2:
        pieces = splitLine(*segment, cut, False)
    else:
        pieces = splitCubic(*segment, cut, False)
    return [tuple(piece) for piece in pieces]
```

```
1549
1550
1551 def cut_shaft(contour, cut):
1552     """Return the part right of cut, closed by a round end."""
1553     pieces = [piece for segment in contour
1554               for piece in split(segment, cut)]
1555     right = [at(piece, .5)[0] > cut for piece in pieces]
1556     starts = [i for i in range(len(pieces))
1557               if right[i] and not right[i - 1]]
1558     assert len(starts) == 1, 'the cut must cross the shaft twice'
1559     order = pieces[starts[0]:] + pieces[:starts[0]]
1560     kept = [piece for piece in order if at(piece, .5)[0] > cut]
1561     exit, entry = kept[-1][-1], kept[0][0]
1562     radius = abs(exit[1] - entry[1]) / 2
1563     middle = (cut - radius, (exit[1] + entry[1]) / 2)
1564     return kept + [quarter(exit, middle, True),
1565                    quarter(middle, entry, False)]
```

The dashes are capsules and the dots are circles, both with the orientation of the original contour.

```
1566 def capsule(left, right, y, radius):
1567     """Return a counterclockwise capsule between left and right."""
1568     a, b = left + radius, right - radius
1569     return [*([((a, y - radius), (b, y - radius))] if b > a else []),
1570             quarter((b, y - radius), (right, y), True),
1571             quarter((right, y), (b, y + radius), False),
1572             *([((b, y + radius), (a, y + radius))] if b > a else []),
1573             quarter((a, y + radius), (left, y), True),
1574             quarter((left, y), (a, y - radius), False)]
1575
1576
1577 def area(contour):
1578     """Return the signed area of a contour."""
1579     points = samples(contour)
1580     return sum(x0 * y1 - x1 * y0 for (x0, y0), (x1, y1)
1581                in zip(points, points[1:] + points[:1])) / 2
1582
1583
1584 def reverse(contour):
1585     """Return a contour with the opposite orientation."""
1586     return [segment[::-1] for segment in contour[::-1]]
```

DASH
DOT

The visible shaft reaches from its left end to the start of the arrowhead. The lengths below are given in units of the thickness of the shaft. Both shafts end in a stub of the length `STUB` before the arrowhead, which is preceded by a gap. The dashed shaft consists of two dashes before the stub, separated by gaps of the length `GAP`. If the dashes would be shorter than `DASH`, there is only one. The dotted shaft consists of three dots of the diameter `DOT`. If the gaps between them would be smaller than `SPACE`, the dots shrink down to `MINDOT` and then their number decreases. Thus the two patterns differ in the number and the length of their elements. The lengths are adjusted, so that the pattern fills the shaft exactly. The left end of the shaft and thus the bounding box

are kept.

```
DASH, GAP, STUB = 1.3, 0.75, 0.6
DOT, MINDOT, SPACE = 1.3, 0.85, 0.5


def dashes(left, stub, thick):
    """Return the left ends of the dashes and their length."""
    for count in (2, 1):
        pitch = (stub - left) / count
        dash = pitch - GAP * thick
        if dash >= DASH * thick or count == 1:
            return [left + step * pitch for step in range(count)], dash


def dots(left, stub, thick):
    """Return the left ends of the dots and their diameter."""
    for count in (3, 2, 1):
        pitch = (stub - left) / count
        diameter = min(DOT * thick, pitch - SPACE * thick)
        if diameter >= MINDOT * thick or count == 1:
            return [left + step * pitch for step in range(count)], diameter


def pattern(glyph, style):
    """Return the contours of a glyph with a dashed or dotted shaft."""
    original, = contours(glyph)
    left, top, bottom, head = shaft(original)
    thick, y = top - bottom, (top + bottom) / 2
    stub = head - STUB * thick
    if style == 'dashed':
        starts, length = dashes(left, stub, thick)
        radius = thick / 2
    else:
        starts, length = dots(left, stub, thick)
        radius = length / 2
    shapes = [capsule(start, start + length, y, radius)
              for start in starts]
    if area(original) < 0:
        shapes = [reverse(shape) for shape in shapes]
    return shapes + [cut_shaft(original, stub + thick / 2)]
```

Draw the contours into a TrueType glyph, approximating the cubic curves by quadratic ones within 0.5 units. The PostScript fonts orient their outer contours counterclockwise, TrueType fonts clockwise.

```
def truetype(shapes):
    """Return a TrueType glyph of the contours."""
    pen = TTGlyphPen(None)
    cubic = Cu2QuPen(pen, 0.5, reverse_direction=True)
    for contour in shapes:
```

```
        cubic.moveTo(contour[0][0])
        for segment in contour:
            if len(segment) == 2:
                cubic.lineTo(segment[1])
            else:
                cubic.curveTo(*segment[1:])
        cubic.closePath()
    return pen.glyph()
```

The vector accent is the slot 126 of the math italic fonts and the combining right arrow above in the NCM math fonts. The width of a Type 1 glyph is only known after its outline has been drawn.

```
def accent_glyph(name):
    """Return the glyph set, glyph name and width of a vector accent."""
    if name in ACCENT_UNICODE:
        font = TTFont(kpse(name + '.otf'))
        glyph = font.getBestCmap()[0x20D7]
        return font.getGlyphSet(), glyph, font['hmtx'][glyph][0]
    pfb, _ = mapfiles(name)
    glyphs = type1(pfb).getGlyphSet()
    glyph = encoding(name)[126]
    glyphs[glyph].draw(RecordingPen())
    return glyphs, glyph, glyphs[glyph].width
```

The math table of the font contains the accent attachments of the original glyphs and the constants of the NCM math font. Without it the Unicode engines would treat the font as a traditional font and centre the accents.

```
def math_table(order, attachments):
    """Return a math table with the given accent attachments."""
    source = TTFont(kpse(ACCENT_UNICODE[0] + '.otf'))['MATH'].table
    table = otTables.MATH()
    table.Version = 0x00010000
    table.MathConstants = copy.deepcopy(source.MathConstants)
    names = [name for name in order if name in attachments]
    top = otTables.MathTopAccentAttachment()
    top.TopAccentCoverage = otTables.Coverage()
    top.TopAccentCoverage.glyphs = names
    top.TopAccentAttachmentCount = len(names)
    top.TopAccentAttachment = []
    for name in names:
        record = otTables.MathValueRecord()
        record.Value = attachments[name]
        record.DeviceTable = None
        top.TopAccentAttachment.append(record)
    info = otTables.MathGlyphInfo()
    info.MathItalicsCorrectionInfo = None
    info.MathTopAccentAttachment = top
    info.ExtendedShapeCoverage = None
    info.MathKernInfo = None
    table.MathGlyphInfo = info
```

```
    variants = otTables.MathVariants()
    variants.MinConnectorOverlap = source.MathVariants.MinConnectorOverlap
    variants.VertGlyphCoverage = variants.HorizGlyphCoverage = None
    variants.VertGlyphCount = variants.HorizGlyphCount = 0
    variants.VertGlyphConstruction = []
    variants.HorizGlyphConstruction = []
    table.MathVariants = variants
    math_font = newTable('MATH')
    math_font.table = table
    return math_font


def top_accent(name, glyph):
    """Return the accent attachment of a glyph of a Unicode font."""
    font = TTFont(kpse(name + '.otf'))
    top = font['MATH'].table.MathGlyphInfo.MathTopAccentAttachment
    return top.TopAccentAttachment[
        top.TopAccentCoverage.glyphs.index(glyph)].Value
```

Write the TrueType font with a glyph for each accent and source font. The glyphs of the Unicode fonts get consecutive private use characters, starting at the character of their accent. The glyphs of the math italic fonts are only addressed by their names through the encoding hepaccent.enc, which the TFM file hepaccent.tfm and the map line use. The TFM file copies the metrics of the original glyphs. XeTeX expects every font to contain a space. Return the slots of the TFM file by source font and accent pattern.

```
def accent_font(accents):
    """Write the accent font with its TFM, encoding and map line."""
    names, glyphs, widths, attachments = ['.notdef', 'space'], {}, {}, {}
    for name, width in (('.notdef', 0), ('space', 333)):
        glyphs[name], widths[name] = TTGlyphPen(None).glyph(), width
    cmap = {0x20: 'space'}
    slots, pl = {}, ['(DESIGNSIZE R 10)', '(CODINGSCHEME HEPACCENT)',
                     '(FONTDIMEN (SLANT R 0.0) (QUAD R 1.0))']
    for source in ACCENT_SOURCES + ACCENT_UNICODE:
        glyphset, glyph, width = accent_glyph(source)
        for accent in accents:
            name = f'uni20D7.{accent["pattern"]}.{source}'
            names.append(name)
            glyphs[name] = truetype(pattern(glyphset[glyph],
                                            accent['pattern']))
            widths[name] = width
            if source in ACCENT_UNICODE:
                cp = accent['unicode'] + ACCENT_UNICODE.index(source)
                cmap[cp] = name
                attachments[name] = top_accent(source, glyph)
                continue
            slot = len(slots)
            slots[source, accent['pattern']] = slot
            metrics = tfm(source).chars[126]
            pl.append(f'(CHARACTER D {slot} ' + ' '.join(
```

```
                f'({key} {pl_value(metrics.get(value, 0))})'
                for key, value in METRICS) + ')')
    builder = FontBuilder(1000, isTTF=True)
    builder.setupGlyphOrder(names)
    builder.setupCharacterMap(cmap)
    builder.setupGlyf(glyphs)
    table = builder.font['glyf']
    for name in names:
        table[name].recalcBounds(table)
    builder.setupHorizontalMetrics(
        {name: (round(widths[name]), getattr(table[name], 'xMin', 0))
         for name in names})
    builder.setupHorizontalHeader(ascent=800, descent=-200)
    builder.setupNameTable(dict(
        familyName='HEP Math Accents', styleName='Regular',
        psName='HEPMathAccents', version='Version 1.0',
        copyright='Derived from Latin Modern, sansmathfonts and New '
                  'Computer Modern under the GUST Font License'))
    builder.setupOS2(fsType=0, sTypoAscender=800, sTypoDescender=-200,
                     usWinAscent=800, usWinDescent=200)
    builder.setupPost()
    builder.font['MATH'] = math_table(names, attachments)
    builder.save(str(ROOT / (ACCENT_FONT + '.ttf')))
    vector = [name for name in names
              if name.split('.')[-1] in ACCENT_SOURCES]
    vector += ['.notdef'] * (256 - len(vector))
    (ROOT / (ACCENT_FONT + '.enc')).write_text(
        f'/{ACCENT_FONT.upper()} [\n' +
        '\n'.join('/' + glyph for glyph in vector) + '\n] def\n')
    path = TEMP / (ACCENT_FONT + '.pl')
    path.write_text('\n'.join(pl) + '\n')
    subprocess.run(['pltotf', str(path),
                    str(ROOT / (ACCENT_FONT + '.tfm'))],
                   check=True, stdout=subprocess.DEVNULL)
    line = (f'{ACCENT_FONT} <{ACCENT_FONT}.enc <{ACCENT_FONT}.ttf')
    MAPS[ACCENT_FONT] = (line, line)
    return slots
```

B.7 Fonts

Write the eight `hepalpha` banks for the three families and the proportional typewriter in medium and bold weight at three sizes. The upright glyphs are placed 128 slots above the italic ones. The accents follow the italic glyphs. They are copied from the math italic font of the family and weight, whose x-height the bank adopts, because TEX lowers an accent by the x-height of its font.

```
def alphabets(glyphs, accents, accent_slots):
    """Write the banks: serif, sans, mono, proportional times weight."""
    for family in range(4):
        for bold in range(2):
            bank = 2 * family + bold
            for size in (5, 7, 10):
```

```
                italic = accent_source(family, bold, size)
                rows = [(accent['slot'], ACCENT_FONT,
                         accent_slots[italic, accent['pattern']])
                        for accent in accents]
                for up in range(2):
                    for glyph in glyphs:
                        kind, slot = classify(glyph)
                        font, cp = source(kind, slot, glyph, family, bold,
                                          up, size)
                        rows.append((glyph['index'] + 128 * up, font, cp))
                fonts = list(dict.fromkeys(font for _, font, _ in rows))
                vpl = [f'(VTITLE HEP CM alphabets bank {bank})',
                       '(FAMILY HEPALPHA)', f'(DESIGNSIZE R {size})',
                       '(CODINGSCHEME HEP MATH ALPHABETS)',
                       '(FONTDIMEN (SLANT R 0.0) (SPACE R 0.0) '
                       '(STRETCH R 0.0) (SHRINK R 0.0) (XHEIGHT '
                       f'{pl_value(tfm(italic).fontdimens["XHEIGHT"])}) '
                       '(QUAD R 1.0))']
                vpl += mapfonts(fonts)
                vpl += [character(slot, fonts, font, cp)
                        for slot, font, cp in rows]
                vpl.append('(CHARACTER D 255 (CHARWD R 0.0) (MAP))')
                vpl += skew_kerns((slot, skew(font, cp))
                                  for slot, font, cp in rows)
                write(vpl, f'hepalpha{bank}-{size}')
```

Write the `hepdecor` banks. For `symbols=minion` the banks 0 to 3 contain all four decorative alphabets for the serif and sans-serif blackboard in medium and bold weight. Otherwise the banks 4 and 5 contain the fraktur alphabet in medium and bold weight, followed by the four composable italic calligraphic alphabets. The serif alphabets use the LM calligraphic capitals, as the NCM math font of the Unicode engines does. The calligraphic capitals of the CM sans symbol fonts only retrace them, so the sans-serif alphabets use the Zapf Chancery capitals of the MATHALPHA package instead, which exist in a single weight and design size. They are scaled to the median capital height of the LM capitals, and the bold alphabet overprints them. The RSFS and double-stroke fonts have no bold weight. MnSymbol uses the skew character 0, RSFS uses 127, the LM symbol fonts use 48 and Zapf Chancery uses 45.

```
CAL_FONTS = (('lmsy', True, 48, 0), ('lmbsy', True, 48, 0),
             ('urwchancal', False, 45, 0),
             ('urwchancal', False, 45, BOLD))


def capitals(name):
    """Return the median height of the capitals of a font."""
    chars = tfm(name).chars
    return statistics.median(chars[cp]['height'] for cp in range(65, 91))


SCALE['urwchancal'] = round(capitals('lmsy10')
                            / capitals('urwchancal'), 3)
```

Each calligraphic or script alphabet of a list of sources occupies the same number of consecutive slots, starting at `first`.

```
def capital_rows(sources, cal, size, first=0):
    """Return the rows of consecutive capital alphabets."""
    rows = []
    for index, (name, sized, char, bold) in enumerate(sources):
        font = name + str(size) if sized else name
        rows += [(first + len(cal) * index + slot, font, cp, char, bold)
                 for slot, cp in enumerate(cal)]
    return rows
```

Write a font with the given rows of slot, source font, source slot, skew character and bold radius.

```
def write_rows(rows, size, title, name):
    """Write a virtual font of decorative glyphs."""
    fonts = list(dict.fromkeys(row[1] for row in rows))
    vpl = [f'(VTITLE HEP {title})', f'(DESIGNSIZE R {size})',
           '(FONTDIMEN (SLANT R 0.0) (SPACE R 0.0) '
           '(XHEIGHT R 0.430555) (QUAD R 1.0))']
    vpl += mapfonts(fonts)
    vpl += [character(slot, fonts, font, cp, bold)
            for slot, font, cp, _, bold in rows]
    vpl.append('(CHARACTER D 255 (CHARWD R 0.0) (MAP))')
    vpl += skew_kerns(
        (slot, SCALE.get(font, 1) * tfm(PROXY.get(font, font))
         .kerning.get(cp, {}).get(skew, 0) + bold
         if skew is not None else 0)
        for slot, font, cp, skew, bold in rows)
    write(vpl, name)


def decorations(rows, cal):
    """Write the decorative banks of symbols=minion, true and ams."""
    assert len(rows) + len(CAL_FONTS) * len(cal) < 255
    banks = [(2 * sans + bold, sans, bold) for sans in range(2)
             for bold in range(2)] + [(4, None, 0), (5, None, 1)]
    skewchar = dict(mathcal=0, mathscr=127)
    for bank, sans, bold in banks:
        for size in (5, 7, 10):
            sources = {'mathfrak': ('eufb' if bold else 'eufm')
                                   + str(size)}
            if sans is not None:
                sources['mathscr'] = 'rsfs' + str(size)
                sources['mathcal'] = ('MnSymbolS' + ('-Bold' if bold
                                                     else '') + str(size))
                sources['mathbb'] = ('dsss' if sans else 'dsrom') + '10'
            used = [(slot, sources[alphabet], cp,
                     skewchar.get(alphabet), 0)
                    for alphabet, cp, slot in rows if alphabet in sources]
            if sans is None:
```

```
1842                 used += capital_rows(CAL_FONTS, cal, size, len(rows))
1843             write_rows(used, size, 'decorative alphabets',
1844                        f'hepdecor{bank}-{size}')
```

Write the `hepscript` fonts with the italic and upright script and the upright calligraphic alphabets in medium and bold weight. The italic script uses the RSFS capitals and the bold alphabet overprints them. The upright script uses the less slanted RSFS capitals of the RSFSO package, whose virtual fonts only add the metrics and accent kerns for the skew character 45 to the slanted base fonts. Their glyphs are taken from the base fonts and their metrics from the virtual fonts. The upright calligraphic alphabets use the Euler script, which has the skew character 48.

```
1845 SCRIPT_FONTS = (('rsfs', True, 127, 0), ('rsfs', True, 127, BOLD),
1846                 ('rrsfso', True, 45, 0), ('rrsfso', True, 45, BOLD),
1847                 ('eusm', True, 48, 0), ('eusb', True, 48, 0))
1848 PROXY.update(('rrsfso' + str(size), 'rsfso' + str(size))
1849              for size in (5, 7, 10))
1850
1851
1852 def scripts(cal):
1853     """Write the script fonts used by symbols=true and ams."""
1854     assert len(SCRIPT_FONTS) * len(cal) < 255
1855     for size in (5, 7, 10):
1856         write_rows(capital_rows(SCRIPT_FONTS, cal, size), size,
1857                    'script alphabets', f'hepscript-{size}')
```

Write the `hepbb` fonts with the serif and sans-serif blackboard in medium and bold weight. Each alphabet occupies the same number of consecutive slots. The NCM blackboard of the MATHALPHA package is addressed by the glyph names of its Unicode characters, the BBOLDX fonts by the input characters. The NCM blackboard uses the skew character 45, BBOLDX has no accent kerns.

```
1858 def blackboard(chars):
1859     """Write the blackboard fonts used by symbols=true."""
1860     sources = ['mathalpha-ncmbb-regular', 'mathalpha-ncmbb-bold',
1861                'BBOLDX-Regular', 'BBOLDX-Bold']
1862     rows = []
1863     for bank, name in enumerate(sources):
1864         for index, (char, cp) in enumerate(chars):
1865             if name.startswith('mathalpha'):
1866                 glyph = f'uni{cp:04X}' if cp < 0x10000 else f'u{cp:X}'
1867                 char = encoding(name).index(glyph)
1868             rows.append((len(chars) * bank + index, name, char))
1869     for size in (5, 7, 10):
1870         vpl = ['(VTITLE HEP blackboard alphabets)',
1871                f'(DESIGNSIZE R {size})',
1872                '(FONTDIMEN (SLANT R 0.0) (SPACE R 0.0) '
1873                '(XHEIGHT R 0.430555) (QUAD R 1.0))']
1874         vpl += mapfonts(sources)
1875         vpl += [character(slot, sources, name, char)
1876                 for slot, name, char in rows]
1877         vpl.append('(CHARACTER D 255 (CHARWD R 0.0) (MAP))')
1878         vpl += skew_kerns(
```

```
            (slot, tfm(name).kerning.get(char, {}).get(45, 0)
             if name.startswith('mathalpha') else 0)
            for slot, name, char in rows)
        write(vpl, f'hepbb-{size}')
```

Read the package, write all fonts, the encoding of the generated base fonts and the two map files. Finally record the date, version and description of the package file, which the package compares with its own, see section A.4.1.

```
def main():
    sty = (ROOT / 'hep-math-font.sty').read_text()
    glyphs = glyph_table(sty)
    accents = accent_table(sty, glyphs)
    alphabets(glyphs, accents, accent_font(accents))
    decorations(decor_table(sty), cal_table(sty))
    scripts(cal_table(sty))
    blackboard(bb_table(sty))
    vector = GLYPHS + ['.notdef'] * (256 - len(GLYPHS))
    (ROOT / (ENCODING + '.enc')).write_text(
        f'/{ENCODING.upper()} [\n' +
        '\n'.join('/' + glyph for glyph in vector) + '\n] def\n')
    for suffix, index in (('', 0), ('-fallback', 1)):
        (ROOT / f'{ENCODING}{suffix}.map').write_text(
            ''.join(lines[index] + '\n' for lines in MAPS.values()))
    version = re.search(r'\\ProvidesPackage\{hep-math-font\}\[([^]]*)\]',
                        sty)
    if not version:
        raise SystemExit('Missing \\ProvidesPackage in hep-math-font.sty')
    (ROOT / 'hep-math-font-vf.def').write_text(
        '% Generated by hep-math-font-vf.py: the version of the\n'
        '% hep-math-font.sty the virtual fonts were generated from.\n'
        '\\def\\hepmathfont@vfversion{' + version.group(1) + '}\n')
    shutil.rmtree(TEMP)
    print('Built the hepalpha, hepdecor, hepscript and hepbb fonts '
          'and the accent font')


if __name__ == '__main__':
    main()
```

</generator>

C Test

<*testserif|testsans>

```
\documentclass{article}

%<testserif>\usepackage[oldstyle,serif]{hep-font}
%<testsans>\usepackage[oldstyle,sans]{hep-font}
\usepackage{hep-math-font}
```

```
1918
1919 \begin{document}
1920
1921 \section{Composable math $a\alpha 123$}
1922
1923 Text 0123456789 and math $aA\alpha\Gamma\partial\nabla 123$.
1924
1925 \[
1926   \mathbf{\mathsf{\mathrm{a\alpha}}}
1927   = \mathrm{\mathbf{\mathsf{a\alpha}}}
1928   = \mathbfsfup{a\alpha},
1929   \qquad
1930   \mathtt{\mathbf{\mathup{x\Gamma}}} = \mathbfttup{x\Gamma}
1931 \]
1932
1933 \[
1934   \mathbfsf{\frac{\partial f}{\partial x} + \hat{x}_\alpha^\beta + \sin x}
1935   \qquad
1936   \mathcal{ABC} \quad \mathscr{ABC} \quad \mathbb{ABC} \quad
1937   \mathfrak{abc}
1938 \]
1939
1940 \[
1941   \mathup{\mathcal{L}} \quad \mathbfsf{\mathcal{L}} \quad
1942   \mathbf{\mathscr{L}} \quad \mathup{\mathscr{L}} \quad
1943   \mathscr{\hat{L}_\mu}
1944 \]
1945
1946 \[
1947   \mathbb{\mathbf{R}} = \mathbf{\mathbb{R}}
1948   \qquad
1949   \mathbb{\mathsf{Rk1}} \quad \mathbfsf{\mathbb{Rk1}} \quad
1950   \mathbb{\hat{x}_n}
1951 \]
1952
1953 \[
1954   \vec{x} \quad \dashedvec{x} \quad \dottedvec{x} \quad
1955   \dashedvec{p}_{\dottedvec{k}} \quad \dottedvec{\hat{A}}
1956 \]
1957
1958 {\bfseries Bold text and math $a\alpha 123$.}
1959
1960 {\ttfamily Typewriter text and math $a\alpha 123$.}
1961
1962 Text Greek \alpha\beta\Gamma\varepsilon\phi\varphi.
1963
1964 \end{document}
```

</testserif|testsans>

D Readme

<*readme>

```
# The ‘hep-math-font‘ package

Composable math alphabets that follow the text font, and dashed and
dotted vector accents

## Introduction

The ‘hep-math-font‘ package provides math fonts matching the text fonts
of the ‘hep-font‘ package. Its math alphabet commands are composable:
bold, upright and the serif, sans-serif and typewriter families can be
combined in any order, e.g. ‘\mathbf{\mathsf{x}}‘ and
‘\mathsf{\mathbf{x}}‘ give the same bold sans-serif x. Math follows the
surrounding text, so that math in a sans-serif or bold heading is
sans-serif or bold as well. The package also provides vector accents
with a dashed and a dotted shaft. With LuaLaTeX and XeLaTeX the package
uses the New Computer Modern math fonts, with pdfLaTeX it uses Latin
Modern and Computer Modern fonts combined into virtual fonts.

The package is loaded using ‘\usepackage{hep-math-font}‘.

## Author

Jan Hajer

## License

This file may be distributed and/or modified under the conditions of the
‘LaTeX‘ Project Public License, either version 1.3c of this license or
(at your option) any later version. The latest version of this license is
in ‘http://www.latex-project.org/lppl.txt‘ and version 1.3c or later is
part of all distributions of LaTeX version 2005/12/01 or later.
```

</readme>

References

[1] *LaTeX Team.* ‘The bm package: Access bold symbols in maths mode’ (1993). CTAN: bm.

[2] *American Mathematical Society.* ‘The amsfonts package: TEX fonts from the American Mathematical Society’ (1995). CTAN: amsfonts. URL: ams.org/tex/amsfonts.

[3] J. Knappen. ‘The mathrsfs package: Support for using RSFS fonts in maths’ (1995). CTAN: mathrsfs.

[4] M. Sharpe. ‘The rsfso package: A mathematical calligraphic font based on rsfs’ (2013). CTAN: rsfso.

[5] *LaTeX3 Project.* ‘Hypertext marks in LaTeX: a manual for hyperref: Extensive support for hypertext in LaTeX’ (1995). CTAN: hyperref. GitHub: latex3/hyperref.

[6] *American Mathematical Society.* ‘The amssymb package’ (1995). CTAN: amsfonts.

[7] A. Blumensath. 'The MnSymbol package: Mathematical symbol font for Adobe MinionPro' (2005). CTAN: MnSymbol.
[8] O. Kummer. 'The doublestroke font: Typeset mathematical double stroke symbols' (1995). CTAN: doublestroke.
[9] K. Hosny, W. Robertson, P. Stephani, and J. Wright. 'The unicode-math package: Unicode mathematics support forXeTeX and LuaTeX' (2006). CTAN: unicode-math.
[10] *LaTeX Team.* 'The iftex package: Am I running under pdfTeX, XeTeX or LuaTeX?' (2006). CTAN: ifxetex. GitHub: latex3/iftex.
[11] H. Oberdiek. 'The kvoptions package: Key value format for package options' (2004). CTAN: kvoptions. GitHub: ho-tex/kvoptions.
[12] *LaTeX Team.* 'The amsmath package: AMS mathematical facilities for LaTeX' (1994). CTAN: amsmath. URL: ams.org/tex/amslatex.
[13] A. Barton. 'The sansmathfonts package: Correct placement of accents in sans-serif maths' (2013). CTAN: sansmathfonts.
[14] G. Milde. 'The alphabeta package: LICR macros and encoding definition files for Greek' (2013). CTAN: greek-fontenc.

Change History

v1.0
General: First release on CTAN: serif and sans-serif math, Greek letters in text and math, symbol package selection, and math alphabets. 1
v1.1
General: Protected the Greek letter macros and refined sans-serif and MinionPro handling. 1
v1.2
General: Added \mathsftext and fixed the processing of the symbols option. 1
v1.3
General: Reworked the Greek letters with fixmath and alphabeta, substituted the LGR fonts of nfssext-cfr, and loaded symbol fonts and math alphabets only when needed. 1
v1.4
General: Simplified the Greek letters and applied starred command definitions. . 1
v1.5
General: Raised the LaTeX format requirement, made math digits lining, and set \mathsf digits in a matching lining font. 1