unicode-math-input — Allow entering Unicode symbols in math formulas*

user202729

Released 2024/01/25

Abstract

Allow entering Unicode symbols in math formulas.

1 Introduction

This package allows entering Unicode symbols in math formulas.

1.1 Existing packages

There are several existing packages, but other than `unicode-math` (which also changes the output encoding) they does not cover a lot of characters and/or does not handle several issues well.

We compare the situation with several existing packages:

- commonunicode:
  - defines all characters to be active, which means it breaks usage of α in fancyvrb's `Verbatim` environment for example.
  - changes the behavior of e.g. ½ in text mode in PDFLaTeX.
  - does not always select best option, for example ∄ always get mapped to `\not\exists` even though the outcome is worse than `\nexists`.
  - fakes several symbols such as ≝ even when there's better option e.g. `\eqdef`,
  - uses `\ensuremath` extensively, which means no error message when it's used in text mode,
  - not as good symbol coverage.
- unixode:
  - defines ′ to be `\prime` which is big and not usable, it should be `^{\prime}` similar to `'`'s definition.
  - defines – (en dash) to be nothing, which breaks the character even in text mode.

*This file describes version v0.1.1, last revised 2024/01/25.

  - does not define × or ± (they're already valid in text mode in LaTeX, but will be silently omitted in math mode)
  - does not handle consecutive superscript/subscript characters.
  - you need to manually patch the source code a bit in order to make it work with PDFLaTeX. And even after that it will raise lots of warnings about redefining Unicode characters.

- utf8x:
  - incompatible with lots of packages.
  - does not define ⨁ (`\bigoplus`)
  - also does not handle consecutive superscript/subscript characters.

  See also https://tex.stackexchange.com/a/628285.

1.2 Features

LaTeX's implementation of input encoding and font encoding is *very* complicated, necessitated by the fact that non-Unicode TeX engines handles each UTF-8 character as multiple tokens and encTeX extension is not enabled in LaTeX.[1]

There's a few other issues that we don't really need to deal with, because they are in the next layer:

- What is the use of the command `\IeC`?
- https://tex.stackexchange.com/a/239575/250119

We don't need to deal with `\IeC` as since TeX Live 2019, the mechanism is no longer used and the Unicode character itself is written to auxiliary files.

We need to get the following things correct:

- `\left(`

  In LuaLaTeX in order to implement this we need to hard code the `\Udelcode` of the character, so if `\langle` is redefined, the change will not follow.

  An alternative is to overwrite the definition of `\left` built-in, but this is not used.

- `\big(` (in amsmath package or outside)

  In PDFLaTeX there's an issue of argument-grabbing (`\big` etc. is a macro so they will only grab the first octet of the `(` character), so the macro must be patched.

  Furthermore, the patching is done `\AtBeginDocument` in case amsmath etc. is loaded after this package.

  We handle `\big \Big \bigg \Bigg` and the `\bigl`, `\bigr` variants etc.

  Pass the option ignore-patch-delimiter-commands to disable the behavior in case of package clash.

- in unicode-math, `a`` renders as `a^{\backprime}` i.e. $a^{\backprime}$. We will not modify the default behavior i.e. $a^{`}$ in this package.

[1] Refer to https://tex.stackexchange.com/a/266282/250119 for a way to force-enable encTeX extension in LaTeX if you're interested.

- `\section{$1 × 2$}` (for writing to auxiliary file in table of contents) – as mentioned above, since TeX Live 2019 this is correct by default.
- Some characters such as × or ½ in PDFLaTeX are already usable outside math mode, we try to not break the compatibility.
- The symbol should work correctly when appear at the start of an alignment entry, e.g., the start of an `align*` cell.
- `$2³⁴$` (consecutive Unicode characters for superscript/subscript, refer to https://tex.stackexchange.com/q/344160/250119.) Also need to handle `'` similarly.
- This packages does modify the default definition of `'` to allow `G'²` to work however. Pass the option `ignore-patch-prime` to disable the behavior in case of package clash.
- The original implementation of `'` is somewhat interesting that it allows sequences such as `G'^\bgroup 123\egroup` to work, we will not emulate it here.
- Also need to handle Unicode prime symbols ′, ″ etc.
- To minimize errors, we make ≢ default to `\nequiv`, but fallback to `\not\equiv` if the former is not available.

  We should also take care of aliases – for example, ≰ should check `\nle` and `\nleq` before fallback to `\not\le` or `\not\leq`.

  Note that by default (or with amsmath or amssymb), `\not` does not smartly check the following symbol, however with some packages such as unicode-math, txfonts the `\not` does do that – in particular, it checks for the presence of control sequences named `\notXXX` and `\nXXX` where `XXX` stands for the original control sequence/character.

  It would be beneficial for amssymb to make `\not` smart, as for example `\not\exists` looks worse than `\nexists`, however the package does not touch `\not`.
- Similarly, `''` default to `^{\dprime}` if available, else fallback to `^{\prime\prime}`.
- Whenever possible, we do not make the symbols have active catcode, only change the mathcode, that way usage of the symbols in places such as `fancyvrb` environment is minimally affected. (see test files for an example)
- We try to make minimum assumptions about the internal implementation details of LaTeX packages; nevertheless this is not always possible.
- Combining modifiers (such as `U+00305 COMBINING OVERLINE` in a̅ , which corresponds to `\overbar`) are difficult to support (although with whole-file scanning + rescansync or LuaTeX's `process_input_buffer` callback it's not impossible; an alternative is to use LuaTeX callback to modify the math list after it's constructed, see https://github.com/wspr/unicode-math/issues/555#issuecomment-1045207378 for an example), plus `unicode-math` does not support them anyway, so they will not be supported.

  They're difficult to support because normally the modifier appear after the character that it modifies but TeX requires the command (e.g. `\overbar`) to appear *before* the character that it modifies.

As a special case, the 4 commands `\enclosecircle` `\enclosesquare` `\enclosediamond` and `\enclosetriangle` are supported (simply because the TEX command can appear after the character it modifies)

- The fraction slash `U+2044 FRACTION SLASH`, as in `1/2` rendering $\frac{1}{2}$, is also not implemented because of similar difficulty as above.
- Symbols such as √ or ∛ will be equivalent to `\sqrt` command (taking an argument to draw a square root) instead of `\surd` (the symbol itself), unlike unicode-math.

  While sequences such as `⁵√{67}` may feasibly be supported without breaking too many things, implementation is difficult and we don't see much use for it.
- Similarly, one might expect that ⏟ `U+23DF BOTTOM CURLY BRACKET` get mapped to `\underbrace`, but the behavior of such command would be a bit unexpected (you need to write `⏟{123}_{456}` to get $\underbrace{123}_{456}$), so this will not be the default.
- the Unicode character is mapped indirectly to the control sequence, so that when the user/some package redefines a control sequence such as `\pi`, the corresponding Unicode character (π) will also change. This will incur a small loss in efficiency however.

  (modulo the issue with `\Udelcode` mentioned above)
- The character ⋯ is mapped to `\cdots` and … is mapped to `\ldots`. Note that `\dots` behaves the same as `\ldots` without amsmath package loaded, but with it it smartly detect which variant to use depends on the following character, for example `$\dots +$` prints $\cdots +$ but `$\dots ,$` prints $\ldots,$.

  There's another discrepancy with the spacing around these 2 characters, see https://github.com/wspr/unicode-math/issues/571.

There are some issues however:

- 𝟘 `U+1D7D8 MATHEMATICAL DOUBLE-STRUCK DIGIT ZERO` gets translated to `\mathbb{0}`, but this is incorrect by default unless the blackboard bold font happens to have such a character.

  (nevertheless, it's difficult to change math font in the middle of the document anyway. Refer to https://tex.stackexchange.com/q/30049/250119.)
- In the unicode-math source code there's this remark:

  > The catcode setting is to work around (strange?) behaviour in LuaTEX in which catcode 11 characters don't have italic correction for maths. We don't adjust ascii chars, however, because certain punctuation should not have their catcodes changed.

  This feature is currently unimplemented.
- At the moment, following a Unicode superscript character, double superscript will not be defined – that is, `G²^3^4` will just display as `G^{234}` – while this is fixable, we don't see much point in detecting this error.

2 Usage

Simply include the package.

```
1 \usepackage{unicode-math-input}
```

Because by default the unicode-math package will already allow entering Unicode symbols in math formulas, this package will raise an error if unicode-math is already loaded.

3 Advanced commands and options

`\umiMathbf` `\umiMathbf {...}`
`\umiMathit` `\umiMathit {...}`
`\umiMathbfit`
`\umiMathscr`
`\umiMathbfscr`
`\umiMathfrak`
`\umiMathbb`
`\umiMathbbit`
`\umiMathbffrak`
`\umiMathsf`
`\umiMathsfbf`
`\umiMathsfit`
`\umiMathsfbfit`
`\umiMathtt`

These functions are not to be used directly. But you can redefine them to customized behavior of bold/italic/etc. Unicode characters.

For example you can `\renewcommand\umiMathbf[1]{\mathbf{#1}}` which is the default behavior.

Or you can execute, for example, `\renewcommand\umiMathscr[1]{\mathcal{#1}}` to use the calligraphic instead of the script alphabet for script characters.

More usefully, you may want to `\renewcommand\umiMathbf{\bm}` to make entered characters such as $\boldsymbol{a}$ appear bold italic in the output, remember to load package bm if you want to do so (which is `unicode-math` behavior with `[bold-style=ISO]` package option).

`\umiFrac` `\umiFrac {1} {2}`

Not to be used directly, but you can redefine it such as `\let\umiFrac\tfrac` (or more clearly, `\renewcommand\umiFrac[2]{\tfrac{#1}{#2}}`) to customize the appearance of Unicode characters like ½.

If you want to customize the appearance of individual symbols, consider using `\umiDeclareMathChar`.

`\umiDeclareMathChar` `\umiDeclareMathChar {α} {\alpha}`

Does what it says. Will override existing definitions, if any.

Note that the Unicode character must be braced.

(You may choose to call `\umiPatchCmdUnicodeArg` `\umiDeclareMathChar` beforehand so bracing is not necessary, but this is not really recommended)

This might or might not destroy the existing text-mode definition. For now, one way to preserve it is `\umiDeclareMathChar {²} {\TextOrMath{\texttwosuperior}{^2}}`.

`\umiDeclareMathDelimiter` `\umiDeclareMathDelimiter {(} \langle`

You must use this in order to use the Unicode character with `\left`, `\big`, `\bigl` etc. (because of the internal detail being that in XeLaTeX and LuaLaTeX, as this package does not change the character catcode to be active, it's necessary to set the `delcode` as mentioned before)

In that case the second argument must be a single token.

Unfortunately, the command does not always work – it must detect the second argument to be a delimiter, but if the detection fails it may not work.

Note: There's no need to provide `\umiDeclareMathAlphabet`, `\umiDeclareMathAccent` or `\umiDeclareMathRadical`, for `\umiDeclareMathChar` suffices. It's not supported to define *control sequences*, for that the typical `\RenewDocumentCommand` or `\RenewCommandCopy` suffices.

`\umiRefreshDelimiterList` `\umiRefreshDelimiterList`

You should normally not need this command.

As mentioned before, in LuaLaTeX once a command is redefined, the Unicode character does not automatically update.

This command will check all the normal delimiter Unicode characters. In PDFLaTeX this command does nothing.

Another way is to use `\umiDeclareMathDelimiter` to manually refresh individual Unicode characters, this is also useful if you define an Unicode character that is not "normally" a delimiter.

`ignore-refresh-delimiter-list` Package option.

`\umiRefreshDelimiterList` will be run `\AtBeginDocument`. Pass this to disable it running.

Only needed if there's some package clash or if there's spurious warning on "not determined to be a delimiter" etc.

`\umiPatchCmdUnicodeArg` `\umiPatchCmdUnicodeArg \sqrt`
`\umiUnpatchCmdUnicodeArg` `\umiUnpatchCmdUnicodeArg \sqrt`

After executing this command, the command specified in the argument (`\sqrt` in this example) can be called with one argument being an Unicode character without needing a brace.

(i.e. you can write `\sqrt α` instead of `\sqrt{α}`.)

Because of implementation detail,

- `\sqrtα` (without the space between `\sqrt` and `α`) works in PDFLaTeX but not LuaLaTeX. (so this form is not recommended.)
- `\sqrt α` works in LuaLaTeX without needing the patch. In other words, the patch does nothing in Unicode engines.

The command being patched must take at least one mandatory argument as the first argument, and it only affect that first argument. In other words, `\sqrt[3]α` cannot be patched this way unless you do e.g. `\newcommand\cbrt[1]{\sqrt[3]{#1}}` then `\umiPatchCmdUnicodeArg\cbrt`, then `\cbrt α` works (but `\sqrt[3]α` still doesn't).

`\umiPatchCmdUnicodeTwoArgs` `\umiPatchCmdUnicodeTwoArgs \frac`
`\umiUnpatchCmdUnicodeArg \frac`

Similar to above, but for commands with (at least) two mandatory arguments.

Only affects these 2 mandatory arguments.

`\umiPatchCmdUnicodeArgExtraGroup` `\umiPatchCmdUnicodeArgExtraGroup \Big`

Don't use this command unless you know exactly what you're doing.

Similar to `\umiPatchCmdUnicodeArg`, but open an implicit group before executing anything and close the group after.

The command being patched must take exactly one argument.

This is useful because some TEX primitives such as `^` or `\mathopen` requires either a single "character" or a group braced with `{...}` / `\bgroup...\egroup` – in particular, `\Big`'s original definition is such that `\Bigl` being defined as `\mathopen \Big` can work, and we must ensure it still work after the patch.

`ignore-patch-delimiter-commands`

Package option.

Pass this to avoid patching `\Big` etc. with the command above (only needed if there's some package clash).

`\umiBraceNext` `\umiBraceNext {abc...} αxyz...`

In the example above, after some steps of execution of TEX, the state will be `abc... {α}xyz...`.

Formally: if the character following the first argument to `\umiBraceNext` is not representable in a single byte and the engine is not Unicode, the character will be braced, otherwise nothing happens. Then the argument is put back in the input stream.

This is an internal command mainly useful for defining the command above, for example after

```
\let\oldbig\big
\def\big{\umiBraceNext{\oldbig}}
```

then `\big(` will eventually execute `\oldbig{(}` which is the desired behavior (that `\oldbig` expects one braced argument).

`ignore-patch-prime` Package option.

Do not patch the default definition of `'` in math mode.

By default it's patched to allow `G'²` and `G²'` to work. Only use this when there's some package clash.

`\umiPatchPrime` `\umiPatchPrime`
`\umiUnpatchPrime` `\umiUnpatchPrime`

As mentioned above, by default `\umiPatchPrime` is run `\AtBeginDocument`. But it can be patched and unpatched manually.

Note that it's undefined behavior if some package modifies the definition of active `'` while it's patched. To resolve conflict, unpatch `'`, load the package, then patch again.

4 Compatibility

This package should have tested with various TEX distribution versions on Overleaf.

5 Advanced remarks

As mentioned before, by design this package defines the Unicode character in math mode to do whatever the corresponding LATEX command does *at the time of use*, so if you redefine the meaning of `\alpha`, then the Unicode character `α` will change as well.

The other "standard" way to define commands in LATEX is to assign the mathcode to the character/control sequence directly, using `\DeclareMathSymbol` etc. which is used to define almost all the standard control sequences. For efficiency reasons or other reasons, you may want to *copy* the definition of an existing control sequence (this way the definition of the Unicode character is not changed when the control sequence changes), you can do that by:

`\umiDeclareMathCharCopy` `\umiDeclareMathCharCopy {±} \pm`

Does what it says.

The second argument must be a single control sequence.

`\umiDeclareMathDelimiterCopy` `\umiDeclareMathDelimiterCopy {‖} \Vert`

Does what it says. Refer to `\umiDeclareMathDelimiter` for difference between this command and `\umiDeclareMathCharCopy`.

In case you want to explicitly specify a font/slot pair for an Unicode character, you can use `\DeclareMathSymbol` etc. directly, then use one of the commands above to copy it to the Unicode character.

Useful resources:

- https://tex.stackexchange.com/questions/98781/create-a-font-table-for-all-available-characters-for-a-particular-font
- https://tex.stackexchange.com/questions/380775/font-table-for-opentype-truetype-fonts
- https://ctan.org/pkg/fonttable (need double quotes if font name has spaces: https://tex.stackexchange.com/a/506246/250119)
- Although there's always `texdoc encguide` for the default (non-Unicode) encodings.

Index

The italic numbers denote the pages where the corresponding entry is described, numbers underlined point to the definition, all others indicate the places where it is used.

U