A true plain text format for math expressions (with markdown-type equivalent variant) and its XML compatible equivalent format

Kent Karlsson
Stockholm, Sweden
2025-11-10

1 Introduction

This proposal is a two-for-one, or indeed three-for-one counting the variant proposal in Appendix A. We will give several (three) computer character codes representations for math expressions, as traditionally displayed in a 2-dimensional layout, in two, actually three (counting Appendix A) different formats, suitable for different representation contexts. One that is suitable for "plain text" contexts, it is a plain text protocol (a true plain text protocol, using C1 control characters), and one that fits with HTML, SVG, Office Open XML, but is still equivalent in what math expressions can be represented, to the C1 version. Appendix A gives a "mark-down" style variant, based on the C1 version, and is equivalent in what math expressions can be represented to the two other formats proposed; though one may want to extend it with some mnemonic character references for common math characters (like Greek letters, some common symbols).

All the proposed formats are succinct, avoiding the verbosity of MathML and OMML (Office Math Markup Language, part of Office Open XML). The proposed formats are often even more succinct than (La)TeX math expressions. Furthermore, the format(s) are not tied to (essentially) one implementation, unlike, for instance, (La)TeX math expressions. The formats presented here are also fully capable of handling RTL (right-to-left) math expressions which is sometimes used in RTL script contexts (Arabic script, Hebrew script, …), without getting into ambiguities or display errors due to bidi processing, neither when it comes to argument order nor mirroring, nor a plain mess (which can easily be the result of bidi processing of text that is not very plain prose).

There are already several (computerised) methods for formatting math expressions, while keeping close to traditional math typographical conventions. One of the older ones is *eqn* for use with troff/groff (https://en.wikipedia.org/wiki/Troff, https://troff.org/, https://www.gnu.org/software/groff/, https://en.wikipedia.org/wiki/Eqn_(software)), somewhat newer is (La)TeX, and even newer is MathML (which is based on XML) and OMML (also XML based, used in MS Office). There are a few other ones, like AsciiMath (https://en.wikipedia.org/wiki/AsciiMath, http://asciimath.org/) and Murray Sargent's "UnicodeMath" (which is actually a misnomer, Unicode does not specify any math expression representation). Most of them require using circumventions for using Greek letters (e.g., \alpha) and various math operators (like \sum) and other math symbols.

So why a new format for representing math expressions? Or, really, a set of new formats, catering for different document type contexts. Here we will give the formats for plain text contexts and for HTML/SVG/OOXML/XML contexts. Others could be devised for other contexts, if need be, and in appendix A we present a markdown type of alternative. Then why? There is a need for more generality than is available in the existing systems. There is also a need for something less verbose than several of the newer systems, especially those that are XML-based; the current ones are very verbose, which makes them impractical; indeed, impossible for manual input/editing, and hard to needlessly handle for computer programs too (for WYSIWYG editing, display, formula manipulation, etc.).

Also, there is a need for better integration with the contextual document formatting system; like TeX math is well integrated with the rest of TeX (though TeX math has integration with web pages, it is a crude integration, with only default sizes and default colour). One should not need to have a separate system for giving size, colour, shadow, strike-through (assuming such style settings are available in the contextual document formatting system), like done in MathML and OMML. Using the same formatting system inside of a math expression as outside of it one can, for instance, set an emphasis colour on a subexpression, or use a math expression in a heading or figure caption, both of which usually has a different font size, and sometimes different colour, relative to the running text. This is often hard to do, when at all possible, in existing systems for math expression layout. That is, math expressions should be more like stylable “ordinary” text, especially for embedded text snippets, and have that styling done like it is done in non-math-expressions parts of the document.

Some of the existing formats are well integrated with a text formatting system for other document formatting, such as text styling, headings, image/figure captions, etc. For instance, in TeX math expressions are very well integrated with the rest of TeX, while others (eqn, OMML, AsciiMath, UnicodeMath, TeX math inside web pages) have next to no integration beyond allowing embeddability of math expressions. The formats we will define here integrates well with ECMA-48 text styling (the C1 variant) and with HTML/SVG text styling (the HTML/SVG compatible variant). We thus will not need “math special” ways of doing styling, so no need for constructs like *<mstyle>* (from MathML). Indeed, we will try to keep the controls and markup to a minimum, without losing expressive power. Note, however, that certain styling of identifiers and numerals in math expressions are semantically significant and must be controlled by an internal math expression mechanism; the proposed formats here have such a mechanism.

Furthermore, in the context of Unicode, one would like to be able to reliably represent math expressions in various parts of the world. While much of higher mathematics is very oriented to using letters that are basic Latin (a to z, A to Z), some Greek letters, and less commonly some Hebrew letters, in other contexts math is not necessarily limited to that. For instance, one may use multi-letter identifiers (though sometimes used also in “international math”, such as ‘cos’, ‘𝔑𝔢’), it is more common in, e.g., computer science, or in school textbooks; or use identifiers in the Arabic script (and then be very careful when it comes to bidi reordering and mirroring so that intended meaning is not distorted, something no other present day system handles well), or in any other script, especially in school books.

Unicode encodes several mathematical operators/symbols, as well as all kinds of letters and digits. Why not use these directly? Instead of circumventions like *\alpha*, *\sum*, *<msqrt>* etc. as is often used in many older formats. (Some of these are still handy, but we leave that to the markdown version only, Appendix A.) And in addition, Unicode assumes support for RTL (right-to-left) math expressions, a possibility not properly and reliably covered by any hitherto existing math expression representation system. RTL math expressions must be reliable in which operators are shown, so that not the wrong mirror form is shown. Displaying the wrong mirror form would entirely change the semantics of a math expression. So, in a math expression as we will define them here (all variants), bidi controls are *not* interpreted in the math expression, and bidi processing (and cursive joining) applies only if a) enabled outside of the math expression and then b) only for numerals and identifiers *each in isolation* plus embedded text blocks *each in isolation*. Bidi mirroring is *never* applied in a math expression, *not even* in embedded text blocks, *not even* to symbol sequences which are always LTR (in isolation). Symbol mirroring is an author edit operation *only*.

The math layout system must not "undisplay" some symbols (like parentheses) in some circumstances (with one exception: explicitly "phantom" elements; plus, markup controls must not be displayed, except when editing "source code"). Further, nor require some kind of "fence" nesting (again not counting markup controls). Any such handling prevents certain common mathematical notations, like]*a*, *b*] (or (a, b]) for left open, right closed interval, or big { only on the left side (for vertical grouping of a number equalities for instance).

We *also* focus on the proper use of Unicode characters, e.g., *never* use combining characters to denote an *above-an-expression* mathematical operator (some systems mistakenly do so), as well as *never* using parentheses or symbols to carry formatting significance (we only use controls, whether C1, XML tags or markdown commands; however a few "symbols" in Unicode we will interpret as control codes with graphic, like for big slash division). Formatting is done via formatting controls, not symbols, but formatting controls can affect certain symbols, e.g., their length or height, like when creating big brackets. How that stretching is to be implemented is not specified here but could range from using font mechanisms to assembling old-fashioned glyph pieces, or plain zoom (often will not look good…; but used in moderation it could work).

All pre-existing formats, as well as the HTML/SVG/XML variant and the markdown variant here, are higher-level protocols. *Only* the plaint text variant presented here is a true plain text protocol, and neither proposal presented here *need* circumventions for representing various math symbols, since they are all based on Unicode, which has many math symbols encoded. However, it is possible to use ECMA-48 references, both "traditional" (ESC<*char*>) for C1 controls (in particular **SCI**: **ESC Z**), as well as extended (new) character references, **CSI**<*Unicode scalar value in decimal*>**_**, if available in the implementation. The HTML/SVG/XML compatible variant can use HTML/SVG/XML character references (**&**<*name*>**;**, **&#**<*decimal*>**;**, **&#x**<*hexadecimal*>**;**). The markdown variant can use similar character references (****<*name*>**;**, ****<*decimal*>**;**, **\§**<*hexadecimal*>**;**), though a list of names have not yet been specified, but should be limited to very common math letters and symbols. It is preferable to use the character in question directly, which requires using a Unicode encoding.

All variants proposed here can handle RTL math expressions. Also, the formats proposed here do not "silence" printable characters (outside of the HTML/SVG/XML markup itself and ignoring "phantom" spacing) nor require that open and close "fences" match in any way, as opposed to some other preexisting formats. Nor are "MATHEMATICAL" characters (invented for MathML) used at all. Note that there is no requirement, from the Unicode standard nor anywhere else, to use "MATHEMATICAL" characters at all in a math expression format, and for the formats here we will not use them, indeed they will be forbidden, but instead a more general mechanism is given. And further, combining characters are used correctly in all the proposed formats.

Our first variant of the proposed format here is a true text level protocol (as opposed to higher-level protocols) for math expression formatting. Bidi controls, and text formatting controls (if available) must be ignored within a math expression (text formatting controls are allowed in embedded text blocks). This format uses hitherto unused control codes in the C1 area, and the format is compatible with plain text.

Our second variant of the proposed format is a higher-level protocol compatible with HTML, SVG and other XML-based/like formats (like Office Open XML). While, technically, the first format (as proposed here) would be workable with HTML/SVG/XML, those formats are blanket allergic to C0/C1 control characters (except certain whitespace characters, like HT and LF). So, we propose a variant format that is compatible with HTML/SVG/XML, using tags and attributes. However, CSS does *not* apply to math expressions even for this alternative format, except *partially* in embedded <txt>…</txt> blocks, and tags that are not math tags (as defined here) must be ignored or even regarded as errors (except partially in <txt>…</txt> blocks) in math expressions (as defined here).

Our third variant (Appendix A) of the proposed format is also a higher-level protocol. It is based on the first variant, which uses SCI and other control chars, and map these controls more directly to "printable" characters. Similar to what is done for the second variant, but no need to make it compatible with HTML/SVG/XML using so-called tags etc. It (intentionally) looks a bit like TeX math (but just superficially). As for the HTML/SVG/XML variant, we then must have some character references, especially for the symbols we use to make the command language. There are general character reference mechanisms specified but following tradition (and for ease of use) we also define some shorter character references, like \\ for \ (we use \ as one of the command introduction characters, so for an actual \ we need a character reference).

The proposed formats are parallel, as near as can be, and conversion between them is straight-forward. All are economic in their amount of markup needed. Thus, it is manageable to type in math expressions "by hand" (though it may be a bit hard to convince a text editor program to insert a C1 control character). However, editors that show the final form during editing of math expressions (like done in MS Word "formula editor" and other tools) can be used, while the program manages the file representation form of the math expressions in the background. The large number of markup tags that must be used when using MathML or OMML is avoided in the formats here, which make the proposed formats manageable for manual code edit as well as simplifies the background management of the file representation in a WISYWIG math editor.

The math styling and layout controls only give style and layout. There is no semantics implied by those controls, nor is a semantics implied by the symbols used in the math expressions. Giving semantics to a math expression, even a simple one like "1 + 1 = 2" is out of scope for this proposal. From the point of view of this proposal, semantics is "in the eye of the beholder" or, slightly more technical, "lies in a higher-level agreement". Even so, that does not prevent certain (for a particular system) well-formed math expressions to be interpreted in a more semantic manner, like for formula manipulation systems or even theorem provers. Note that this approach is no different from the approach in, e.g., TeX, which also only gives style and layout, while certain "well-formed" math expressions expressed via TeX code can be used as basis for formula manipulation, theorem provers and the like. (Compare the IANA registration https://www.iana.org/assignments/media-types/text/vnd.latex-z which is for a .sty variant of LaTeX as layout notation and Z notation (https://en.wikipedia.org/wiki/Z_notation) as semantics.)

For doing any kind of semantic handling of a math expression, an extra layer of parsing is needed, to handle fences (parentheses, brackets, etc.), visible operator precedences (e.g., + "binds harder" than =) and associativities, "distfix" parsing (e.g., for "$\{x \in Y \mid x < 4\}$"). This extra layer of parsing is *out of scope* for this proposal and should be the subject of other parsing specifications for this *additional* layer of parsing, directed at different conventions and different math subject areas.

2 Goals and non-goals for this proposal

2.1 Goals for this proposal

a) Enable proper traditional math expression layout and styling for at least basic math expressions, preferably more; but include computer science, engineering, and school math, as well as logic (of various kinds), and other non-mainstream “math”.

b) Use C0/C1 control characters to control the math expression layout and formatting succinctly. Ordinary printable characters stand for themselves only (the C0/C1 variant).

c) Give an alternative format in HTML/SVG/XML style with the same expressive power as in b), but still be succinct (in contrast to, e.g., MathML and OMML which are very verbose).

d) Handle “fences” (parentheses, brackets, etc.) correctly. Every such character should be displayed if occurring in a properly formed math expression (except if part of a “phantom” or “pragma” part), and there must be no nesting/pairing requirement of any kind for such characters (that would violate common notations).

e) Handle letter and digit styling properly. No “MATHEMATICAL” letters and digits (to be disallowed for use with the styling and layout controls proposed here), but instead use math specific styling controls. Plus, for convenience, math style defaults for some common names (like ‘cos’, ‘Z’), including multiletter (SI) units and chemical elements.

f) Properly allow for multi-letter names for functions, variables, chemical elements, units, and more.

g) Handle RTL (right to left) math expressions properly. RTL math expressions are sometimes used in conjunction with RTL scripts, like Arabic or Hebrew. That includes mirroring symbols (when the author of the expression wants it) also for characters that do not have a mirrored “twin” character in *BidiMirroring.txt* or *ExtraMirroring.txt*, but also *not* to mirror when that would be wrong to do, i.e., *no automatic* mirroring anywhere in a math expression (not even in an embedded text block). Plus, *no automatic* rearrangement when displaying math expression parts.

h) No use of Unicode’s “non-character” characters in the formats. (“Non-characters” in Unicode are intended for purely internal temporary use (like referencing a substructure that has been parsed out, e.g. a math expression in a paragraph), and must not be used for data interchange, like in a file format that, here, may include math expressions.)

i) Make a system that is well integrable with the rest of the text formatting system.

2.2 Non-goals for this proposal

a) Mimic every functionality of, e.g., (La)TeX, MathML, or any other math layout system. Needed missing features *may* be added in a revised version of this proposal.

b) Support strange or new-fangled notations (other than which symbols to use), e.g., overlays or positioning at a “far” distance or looking like a graph-based notation. Graph like notations as such are fine, of course, just not in scope for this proposal.

c) Interpret embedded TeX math expressions or any other pre-existing math layout system.

d) Handle “display form” vs. “inline form” for math expressions. That is an “outside of the math expression” formatting issue. Line breaks, centring, and numbering are all outside of the math expression itself, and therefore out of scope for this proposal.

e) Make a system where the expressions can double as program source code in a programming language.

f) Provide an implementation of the proposed math expression layout systems.

2.3 Examples

For example, as a teaser for the rest of the proposal, the famous math expression $E = mc^2$, is in

- eqn: #E=mc **sup** 2#, having set the inline delimiters to ##.
- TeX: **$**E=mc**^**2**$**. In TeX (and eqn) there is no need for a space between m and c. TeX has a special "style" for multi-letter variables (\mathit).
- Our format, C1 variant: **SME** E=m c **SCI^**2 **EME**, where the bold parts represent C1 control codes. The space between m and c is needed to make two variable names, rather than one multi-letter one.
- Our format, XML variant: **<me>**E=m c**<rsp/>**2**</me>**, where the bold parts are XML tags.
- Our format, appendix A: **${**E=m c**^**2**}**, where the bold parts represent mark-down control codes.
- MathML, OMML: Both have way too long XML markup to give here even for this small example.

3 Style inherited from the start of a math expression

Math expressions have a particular kind of styling for a special kind of math style selection. Most aspects of styling are not changeable in "normal" ways within a math expression. E.g., CSS must be disabled within a math expression (except for in <txt>…</txt>). Instead, font size, background/high-light and foreground colours, as well as shadows and shadow colour, and in addition strike-through, are inherited from the style in effect at the start of the math expression. These inherited styles cannot be changed within the math expression, apart from embedded "text" in a math expression, which, in turn may contain a math expression. So, the settings for typeface, boldness and italic/slanted, underlining/overlining and other emphasis marks are *not* inherited (though some are recorded for being inherited to embedded texts).

Strike-trough is often used for signifying either that something should be deleted (for whatever reason), is marked as wrong or marked as already been dealt with. It would be a bit strange if an embedded math expression was skipped in the strike-through. The strike-trough should go through each letter/digit/symbol of the math expression, wherever they are placed, and the strike-through line keeps its style (thickness, single/double, waviness, …) as well as colour (which may be different from the colour of the math expression) also inside the math expressions.

Further, tab settings are *not* inherited to a math expression at all (not even to embedded text blocks, see below). Indeed, tabs (HT, VT), are mostly ignored in math expressions, *except* in matrix layouts (using the C1 variant of the proposed system) where they mark matrix element *termination*. But the tab stops as set outside of the math expression, if there are any such, are still completely ignored within a math expression, including in embedded text blocks.

Writing directions setting is also ignored (like for bidi or vertical CJK). Math expressions are *always* left-to-right. Indeed, also RTL math expressions are stored in "visual order", i.e., left-to-right; but a math expression editor may help giving input in RTL order. However, within an identifier with multiple letters as well as in an embedded text block (see below) bidi (and Arabic letter joining) is enabled if and only if the writing directions setting outside of the math expression has bidi processing enabled. But even if bidi processing is enabled, symbols (including punctuation) are *never ever* mirrored within a math expression. Indeed, symbol *sequences* must be LTR, regardless of bidi, *also* in embedded text (**STX…ETX/<txt>…</txt>**) blocks.

Mirroring of symbols (including punctuation, like parentheses and commas) is done purely as an *editing* operation for *individual* symbols only and should *not* be limited to characters with *BidiMirrored=Yes* (see http://www.unicode.org/Public/UNIDATA/BidiMirroring.txt), but should be possible also for arrow symbols and more (see *ExtraMirroring.txt* in Appendix A or in https://www.unicode.org/L2/L2022/22026r-non-bidi-mirroring.pdf). Note: mirroring is an *edit* operation to be done by the *author* of the math expression, *never* mirroring as an automatic display operation or mirroring done by a program as part of the display process or as an "overall" edit. Mirroring is done either by character replacement or by applying the math mirror control to a symbol. The latter should be used only when there is no mirror character encoded. Regarding the mirror control: math symbols (including punctuation) are never italicised/slanted in a math expression, so mirroring can be done by flipping the glyph along a vertical mid-axis.

Note also that case mapping, even as an editing operation (some text editing programs allow for case-changing swats of text), is *not* allowed within a math expression. That includes not allowing small caps styling, and similar font styling, which may imply hidden case mapping; such styling should not be allowed even in embedded text blocks, there should not be any need for that.

4 Math expression formatting and math styling controls

As mentioned, we will give three variants of the math layout system proposed here, one based on C1 control characters, one that is compatible with HTML/SVG/XML using control tags, and one that is more in the markdown style (actually, and intentionally, a bit similar to TeX math).

We will use four C1 control codes that are undefined in ECMA-48 5th edition: U+0080, U+0081, U+0084, U+0099. We will refer to these control codes as **SME** (START OF MATH EXPRESSION), **EME** (END OF MATH EXPRESSION), **MHD** (MATH HORIZOTAL DIVISION) and **MMS** (MATH MIRROR SYMBOL), respectively. They will be detailed below, but **SME**…**EME** will be used as invisible math expression format bracketing; compare TeX/LaTeX's $...$ and {…}.

We will use **SCI** (SINGLE CHARACTER INTRODUCER, U+009A) with a following single character (note: character, not byte) for math controls that signify layout and styling controls. The single follow-on character will be a symbol (by which we here include punctuation), digit or letter, but is (here) not limited to ASCII, and sometimes will even be a C0 control character (e.g., **SCI HT**). In ECMA-48 5th edition this kind of sequences are not defined, just reserved for future standardisation, and thus not even given any name. But we will use these sequences as *math expression styling and layout controls*, and refer to them as SCI-sequences.

Math styling and layout controls, **SCI** *x*, SCI-sequences, should be ignored outside of outermost **SME**…**EME** brackets. Note that **SME** … **EME** nest, and nest with **STX**…**ETX** (text blocks, see below, using U+0002 (START OF TEXT, **STX**) and U+0003 (END OF TEXT, **ETX**)) as well (but **STX**…**ETX** do not directly nest, they only nest indirectly via **SME** … **EME**). **SME**…**EME** are used also as grouping brackets inside the outermost **SME**…**EME**. Note that **SME**…**EME** *never* generate visible parentheses, brackets or braces (not counting a "show invisibles" mode). All proper visible parentheses, brackets or braces ("fences") *must* be given explicitly.

On the other hand, use of visible parentheses, brackets etc. *never* imply an **SME** or **EME** control (though symbol sequences in text blocks do get that *around* them, making them math styled and LTR). Visible and control bracketing are completely independent of each other. There is no requirement, from the viewpoint of this system of controls for math layout, for visible fences, to nest or be paired in any way at all. This is important for such math expressions as "]*x*,*y*]", "]*x*,*y*[", "[*x*,*y*[", "(*x*,*y*]" (etc.), which are common notations for open intervals, as well as having a (big) left curly brace on the left a matrix layout, but no brace on the right side. It also allows being able to write student assignments of the type "spot the (syntactic) error" using an expression where, e.g., a left parenthesis is left out. Those things can be written using (La)TeX as well as the system we will propose here but that may cause problems in some other existing math layout systems.

On an additional level of parsing (not in scope for this proposal), fence nesting, operator precedence, etc. may be parsed and checked, according to the conventions implemented for that additional level of parsing. But this varies both by math (physics, chemistry, computer science, logic, engineering, …) area and local conventions. This additional level is not covered in the proposal (with variants) given here.

4.1 Math expression styling and layout controls

We will here use four hitherto unused control characters in the C1 area for certain "math controls". Invisible math open and close brackets: The brackets can be used both outside of a math expression, starting/ending a math expression embedded in the text, as well as inside of a math expression for grouping. And we will use two other C1 control characters for two other operations, only for use inside of math expressions. We will introduce several other mathematical layout operations, only for use within math expressions, using **SCI** (SINGLE CHARACTER INTRODUCER), which is currently not used for anything else (it is reserved for future standardisation in ECMA-48 5th edition).

We will assign the currently unassigned C1 control codes, for the C1 math expression representation variant, as follows:

- U+0080, START OF MATH EXPRESSION, **SME**. See section 4.2.1.
- U+0081, END OF MATH EXPRESSION, **EME**. See section 4.2.1.
- U+0084, MATH HORIZONTAL DIVISION, **MHD**. See section 4.8.1. (We will use four other Unicode characters for solidus-based division notation.)
- U+0099, MATH MIRROR SYMBOL, **MMS**. See section 7.3. The Unicode bidi algorithm is not permitted to do any symbol mirroring in a math expression. But some "mirrorable" symbols do not have a mirror character allocated. Hence the need for a control character for requesting mirroring of a symbol.
- **SCI** (U+009A) sequences for math styling of variables, numerals and letter-like symbols (other symbols cannot be styled in math expressions). See section 4.3.1. In this case the character after the **SCI** is a letter, mostly A-Z and a-z, but for Arabic math styles some Arabic letters are used. This styling is math specific, not related to styling of running text. This styling cannot be used in embedded text blocks, except in math expressions embedded in text blocks.
- **SCI** sequences for math positioning (like subscripts). See section 4.6.1 and 4.7.1. In this case the character after the **SCI** is a punctuation/symbol, mostly in the ASCII range, but some other symbols are used as well.
- **SCI** sequences for math expression matrix layout. See section 4.10.1. For these we also use some ASCII punctuation/symbols/digits to follow the **SCI**, as well as a few control characters (**HT**, **VT**). For matrix cell *termination* we also use **HT**, **HTJ, SCI HT, SCI VT**. The termination determines which column alignment the preceding cell has.

An **SCI** sequence is **SCI** followed by a character, even some C0 control characters are allowed, but no “device controls”, and surrogates, non-characters, and general category Cf characters should be considered as excluded, but for brevity not expressed here:

```
sci-sequence ::=  SCI  ( [\u0001-\u0003\u0009-\u000D\u0020-\u007E] |
                         [\u00A0-\U10FFFF]  |  CSI-char-reference  ) // should skip ‘surrogates’
```

The follow-on character should be an *assigned* character position, but that changes over time, as more characters are encoded; for our purposes here, we of course use assigned character positions, mostly in the ASCII range. In some contexts, **SCI** may be represented by an ECMA-48 *control character reference*: **ESC** \u005A, that is **ESC Z**, to represent **SCI**.

We will also consider a few Unicode math characters, general category Sm, to be control characters (no **SCI**). Namely for fractions or what looks like fractions; semantics is out of scope for this proposal. Why? Because these characters are not just glyphs from a font, but inherently stretch, just like **MHD**, MATH HORIZONTAL DIVISION, which we will define as U+0084 here. This also means that we can have special syntax for them, unlike ordinary symbols; thus also, they are not permitted in the HTML/SVG/XML compatible variant defined here, where we instead use tags (with attributes) for them.

4.2 Math bracketing controls for layout

We need a way to mark which parts of a text is a math expression, as well as be able to group parts of a math expression. This is a grouping for layout, “unaware” of any kind of visible “fences” (parentheses, brackets, …) that can be used. This is just the same as is the case in TeX, so there is nothing unique or new about this approach.

4.2.1 Using C1 control characters

For this functionality, we will use two control codes (in C1) that hitherto have not been used/allocated to anything. One to start the math group, and one to end the math group.

- U+0080, START OF MATH EXPRESSION, **SME**. An invisible open parenthesis for grouping of math subexpressions. Compare (La)TeX’s $ and {. *Visible* parentheses, brackets, etc., imply ***no** grouping whatsoever* for the math expression formats we will specify. **SME** is the only “math control” that is interpreted *outside* of a math expression. All **SCI** sequences stay undefined outside of a math expression (as defined here; **SCI** still “gobbles up” the following character also outside of math expressions). Note that it is invisible *assuming that* 1) math expressions in this format are supported, *and* 2) there are no syntax errors in the math expression. Otherwise SME and other math controls are *visible* (like (visible!) SUB or better display) as an error indication, just like for any C0/C1/Cf control that has not been interpreted. Ideally, the error display should be informative, and should be the same as in a “show invisibles” mode.
- U+0081, END OF MATH EXPRESSION, **EME**. An invisible close parenthesis used for grouping. Compare (La)TeX’s $ and }. It closes the most recent (textually) still open **SME**. This control code should be ignored outside of a math expression.
- If an editor executes a copy-and-paste operation of a *part* of a math expression, it may need to insert **SME**…**EME** around the copied/pasted part of the math expression. Indeed, the editor may need to balance up **SME**…**EME**, in case the copied part was not well balanced. (The latter may result in strange parts being copied, though. Copy-

paste does not do any semantic interpretation at all, not even visible operator precedence or visible bracketing held together.) Copy-paste (or, rather, text selection) can be sensitive to the math expression structure as defined here, but need not be.

A math bracketed expression is **SME** *expr* **EME** where *expr* is a math expression as defined in section 0. The **EME** that closes the **SME** that started outside of the math expression closes the math expression and returns to “normal” text. The **SME** and **EME** never generate any visible bracketing (not counting a “show invisibles” mode, which a text editor may have). No styling changes, by whatever mechanism, are interpreted inside the outer **SME**…**EME** except partially within a **STX**…**ETX** block (see below), and apart from explicit math styling (as defined here). This assumes a syntactically well-formed math expression (as defined here). An implementation may implement some kind of syntax correction in the face of syntax errors. That is out of scope for this proposal.

4.2.2 Using HTML/SVG/XML compatible control tags

For the HTML/XML compatible variant we will use **<me>***expr***</me>** for the math expression bracketing, where *expr* is a math expression as defined in section 4.12.2.

Like for the C1 control code version, the **<me>** open and close tags never generate any visible fences (except for “show invisibles” mode in text editor programs, which in this case may show the tags as “source” text). Like for the C1 control code version, **<me>** elements can be used also inside a math expression for grouping. Other math related tags (per below) have no (layout/formatting) meaning outside of an outer **<me>**…**</me>** and are then ignored or even cause an error.

None of the math related tags can be styled via CSS (or other styling mechanism that are not the **<st>** tag below) in any way except for the partial style inheritance and **<txt>**…**</txt>** blocks (embedded “non-math” text), detailed below. All non-math related tags are ignored inside an **<me>**…**</me>** except for in embedded **<txt>**…**</txt>** (but these should only be short texts) or are handled as errors.

4.3 Math styling

In math expressions the style of a “name” (of a function, variable, standard function, set, etc.) can carry semantic meaning, or at least conventionally be set in a particular style. It is not just arbitrary font differences, but particular font ranges that are recognised. I.e., particular math styles, and certain “ranges” of such styles. Therefore, the usual styling by having different fonts, bold and italic, etc. do not quite apply for math expressions. We will therefore use a separate system for specifying which font and font style to use for a particular “name” and rule out the “ordinary” styling of text within math expressions. For convenience, we will use default math styles, similar to what is done in TeX, but more of it.

Use of so-called Unicode prestyled letters and digits in math expressions as defined here is undefined, or actually, not permitted. We will instead device another system for specifying the math style for a “name”. Actually, we will device two, one based on (otherwise undefined) C1 control codes, the other based on HTML/SVG/XML tags with attributes. In both cases we will use default styles, like done in TeX to cut down on the prevalence of the styling controls. Use of prestyled characters would conflict with the use of these math styling controls and defaults.

The style of letters and digits in math expressions (as defined here) is controlled by SCI-sequences for the C1 controls version, and the **<st>** tag (with attributes) for the HTML/SVG/XML compatible version.

Certain aspects of styling are inherited from before the math expression: font size, foreground and background colours, shadow (and its colour), and strikethrough. Other style settings (weight, slant/italics, current font, underlining, …) are *not* inherited.

Note that parentheses, brackets and braces, in math expressions, are never inclined/italic or bold, no matter which style is set, and that cannot be changed by a math style setting. The same applies to other punctuation and, importantly, most symbols (including math symbols, arrows and arrow-like symbols). However, *letter-like* symbols (like n-ary sum) can be math styled.

All ASCII digits, within a style and at a given size, have the same width (“tabular style” in font terminology) in a math expression.

The styled item can be one of:

1. A sequence of proper combining sequences where the base characters are letters (L*, Sk). Included as letters are CJK ideographs and Hangul syllables (from Hangul Jamo or precomposed). The letters in the sequence *shall* all have the same bidi category and should all be in the same script. However, an implementation need not support all scripts (far from it), but all implementations must support ASCII letters. Letters with a compatibility decomposition are *not permitted*, with some exceptions (see section 9.1), nor is U+2118 permitted (use instead **SCI W**p). (NO-BREAK) HYPHEN is also usable. Note that HYPHEN-MINUS is mapped to MINUS SIGN (see section 8.7), not HYPEN.
2. A sequence of decimal digits and certain punctuation (COMMA, FULL STOP, PSP, SCI SP) starting and ending with a decimal digit, indeed each instance of allowed punctuation must be preceded and succeeded by a digit. For convenience, a SCI SPACE *between* digits is interpreted as PSP. All the digits in the sequence shall be of the same script, and a script change breaks up the digit sequence. However, an implementation need not support all scripts (far from it), but all implementations must support ASCII digits. Digits with a compatibility decomposition are *not permitted*. Decimal digits are the characters that have Unicode general category Nd (decimal digits) that do not have compatibility decompositions.
3. A single combining sequence with a base character that is a symbol (including punctuation: S* except Sk, P*), possibly preceded by an **MMS** (**MMS** is mostly intended for RTL math expressions). Technically all symbols (including punctuation and arrows, S*, P*) can be math styled via **SCI** <*letter*>, but that has no effect except for letter-based symbols (integral, n-ary sum, …). Symbols with a compatibility decomposition to a single symbol are *not permitted* with a few exceptions (listed below), nor are “default-ignorable” ‘symbols’ (like INVISIBLE TIMES). Note that in section 8.7 we specify a few symbols to symbol mappings.

Note again that symbols cannot be math style changed, except for the ones that are letter-like. The math style for non-letter-like symbols is fixed: upright, sans-serif, normal weight, and “conventional math” look for arrows. This means that, e.g., RIGHTWARDS ARROW (→) should look exactly like RIGHTWARDS SANS-SERIF ARROW in a math expression (though we will have a mapping specified below), and that symbols (including fences, arrows, …) in a math expression are never bold nor italic/slanted.

However, some arrow symbols have effectively assumed a particular almost “dingbat” style, that is largely unsuitable for math expressions. We will therefore map these “new dingbat” arrows (despite Sm general category) to arrow characters more suitable for math expressions, not relying on fonts to handle this. The size of symbols may vary of course: inherited size, index size, stretched. An implementation need not support all symbols (in Unicode) but should support all symbols that are commonly used in math expressions.

4.3.1 Using C1 control characters

The style operators are prefix math layout operators. They are valid/interpreted inside of math layout expressions (as defined here) only.

Note also that styling operators *cannot* be applied to a math bracketed subexpression.

To write a multiplication (of single letter variables), commonly written as a juxtaposition without any operator (INVISIBLE TIMES is not permitted; no default-ignorable code point is allowed), use a SPACE (or **SYN**; todo: work that in more; or **HT**, or *NLF*) to separate the operands of the multiplication. This should give the proper spacing. SPACEs are not rendered (i.e. normally collapsed to zero-width space) inside of a math expression, except inside an **STX/<txt>** block, see below). When using multi-letter variables, which is uncommon in “regular” math, except for certain standard functions (but more common in engineering, for instance), it is better to use a visible operator also for multiplication.

Whitespace (**SP**, **HT**, *NLF*, one or more) between upright letters and upright digits, as well as between upright letters and italic/slanted letters (either order), and between two upright letter sequences is converted to a narrow space, suitable for, e.g., **SME**sin 5**EME** and **SME**50 kW**EME**. The spacing otherwise, like between two italic letter sequences, is tighter (but not as tight as inside an italic letter sequence, compare default math style (or \mathnormal) vs. \mathit in TeX).

Using a corresponding lowercase letter instead of the uppercase letter after the **SCI** (per below) says to use *lowercase digits* but otherwise the same style as for when using uppercase letters in these **SCI** styling operators. However, lowercase digits are uncommon in math expressions. And zeroes do not have an interior slash or dot in any of the styles, though **STX/<txt>** blocks can, in principle, have such zeroes with interior slash or dot. Note that we will use **STX/<txt>** blocks for hexadecimal numerals, since only that allows for mixing digits and letters in a numeral; whether some decoration (like 0x, or $_{16}$) is used for hexadecimal numerals is up to the expression author, and therefore not specified here. For identifiers, **SCI**<*lowercase letter*> says to use small caps for capital letters in the identifier.

Not all styles need to be supported even for all ASCII letters/digits. For instance, **SCI W** should have support for “p” (Weierstraß p) but need not have support for any other letter/digit. (In this document we use a sans-serif font even when describing style variants that are serifed.)

- **SCI A** MSU Math serifed upright prefix operator. (Compare TeX’s \mathrm and \\mathup.) Default for digit-based sequences and certain Latin script identifiers. This math style is often used for standard functions, such as “sin”, “cos”: “**SCI A**sin”, “**SCI A**cos”, …; and for unit names, e.g., “m”, “mm”, “s”, “kHz”, …
  Indeed, for convenience, the following letter sequences has this math style as default (in math expressions, as defined here): “sin”, “asin”, “arcsin”, “sec”, “asec”, “arcsec”, “cos”, “acos”, “arccos”, “csc”, “acsc”, “arccsc”, “tan”, “atan”, “arctan”, “cot”, “acot”, “arccot”, “sinh”, “asinh”, “arcsinh”, “sech”, “asech”, “arcsech”, “cosh”, “acosh”, “arccosh”, “csch”, “acsch”, “arccsch”, “tanh”, “atanh”, “arctanh”, “coth”, “acoth”, “arccoth”, “ln”, “lg”, “log”, “exp”, “lim”, “inf” (for

infimum, not infinity), “liminf”, “sup” (supremum), “limsup”, “Im”, “Re”, “mod”, “min”, “max”, “deg”, “gcd”, “det”, “hom”, “arg”, “dim”, “cis”, the Hebrew letters “א”, “ב”, “ג”, “ד”, as well as “∇” and other letter-like symbols, including all integral (∫) symbols (TeX exceptionally uses a very italic integral sign, here coded as **SCI C ∫**, but that is not recommended since an italic integral sign does not stretch well, and integral signs are otherwise vertically stretchable), “∂” (upright is default here), and “∑”, “∏”, as well as ∀ and other letter-like symbols.

Multi-letter unit symbols (including prefixes) has this math style by default. (In SI, short unit "names" are called "symbols" (they are *not* abbreviations, but are mnemonic), even though almost all consist of letters, except for °C, °, ′, ″ (the latter three are for angles, not imperial units, nor time); when prefixed, ″ is replaced by as (arc second), but only when prefixed since “as” otherwise means attosecond. ° and ′ as such cannot be prefixed. This includes units “allowed for use with” SI, not just the pure SI units. Specifically, identifiers matching this regular expression have the **SCI A** math style as default style, and (for convenience) need not have the **SCI A** styling control explicitly (generous with prefixes, but that does not cause any problems):
[QRYZEPTGMkhdcmμunpfazyrq][sgmlLNJKWAVCHFSΩT]|[dcmμunpfazyrq](B|Np|deg|rad|sr|as)|Np|
rad|sr|[QRYZEPTGMkhdcmμunpfazyrq]?(Hz|Wh|Ah|eV|Wb|Pa|bar|Bq|Sv|Gy|mol|kat|cd|lx|lm|ohm)|
[QRYZEPTGMk](t|Bd|b|bit|px)|Bd|bit|px|(Qi|Ri|Yi|Zi|Ei|Pi|Ti|Gi|Mi|Ki)[oB]|min|ha|kcal|°C|pH|pOH
While “pH” and “pOH” are not counted as “units” per se, they do have some similarities with bell (B) and neper (Np), and (which is what we consider here) are typeset in a units style. “Wh” is really a multiplication of two units, in the representation proposed here: “**SCI A**W **SCI A**h”, but for convenience it is included in this regular expression as if it were a single unit symbol, likewise for “Ah”; “ha” is technically a historical (from before it was called SI) SI unit (prefix “h”, *hecto*, to the unit “a”, *ar* or *are*), but is still in very common use, similarly for “kcal”; “µ” (U+00B5, U+03BC) refers to U+03BC, and U+00B5 is excluded as if it had a compatibility mapping to U+03BC; for “Ω” (U+2126, U+03A9), U+2126 has a canonical mapping to U+03A9, but here we regard that as a compatibility mapping. Wide characters and precomposed unit symbols in Unicode are *not* covered, since they have compatibility mappings (and thereby generally excluded from math expressions). Other unit symbols still need to be written with an explicit **SCI A** to get the correct style for unit designations. That includes, but is not limited to, “d” (=24h), “h”, “s”, “g”, “m”, “t”, “K”, “L”, “N”, “J”, “W”, “A”, “V”, “C”, “H”, “F”, “S”, “Ω” (whether the ohm unit or the mathematical omega constant), “T” (tesla), “B” (bel or byte; the B that take “small” prefixes is bel, B (for byte) with large SI prefixes should not be used, instead use binary prefixes), “b” (bit, SI prefixes commonly used with SI prefix semantics), “o” (octet) and “Å” (Ångström) as unit symbols (note: these are without prefix, just single-letter unit designations), as most of these single letters have, and should have, italic style, **SCI C**, by default (except N, C and H which, like Z and R, have double-struck style, **SCI I**, by default), as well as “ppm”, “ppb”, “au”, “eV” and others. Side note: SI recommends using notation such as µL/L or µg/g, mg/g, cL/L, etc. instead of using %, ‰, “ppm” in order to be clear about what is measured, rather than bylining with “by volume” or “by weight”. Note also that SI prefixes cannot be used in isolation nor combined. The prefix “da” (deca) is not included the regular expression above, since it is in practice unused. One can still use **SCI A** explicitly of course. The “h”, “d”, “c”, prefixes are commonly used with the unit symbols “g”, “m” and “B” (though not hB) and “h” with “a” (ar, giving “ha”), but basically never with other unit symbols.

Likewise this style is the default for two-letter chemical element symbols: “Ac”, “Ag”, “Al”, “Am”, “Ar”, “As”, “At”, “Au”, “Ba”, “Be”, “Bh”, “Bi”, “Bk”, “Br”, “Ca”, “Cd”, “Ce”, “Cf”, “Cl”, “Cm”, “Co”, “Cr”, “Cs”, “Cu”, “Db”, “Ds”, “Dy”, “Er”, “Es”, “Eu”, “Fe”, “Fm”, “Fr”, “Ga”, “Gd”, “Ge”, “He”, “Hf”, “Hg”, “Ho”, “Hs”, “In”, “Ir”, “Kr”, “La”, “Li”, “Lr”, “Lu”, “Md”, “Mg”, “Mn”, “Mo”, “Mt”, “Na”, “Nb”, “Nd”, “Ne”, “Ni”, “No”, “Np”, “Os”, “Pa”, “Pb”, “Pd”, “Pm”, “Po”, “Pr”, “Pt”, “Pu”, “Ra”, “Rb”, “Re”, “Rf”, “Rg”, “Rh”, “Rn”, “Ru”, “Sb”, “Sc”, “Se”, “Sg”, “Si”, “Sm”, “Sn”, “Sr”, “Ta”, “Tb”, “Tc”, “Te”, “Th”, “Ti”, “Tl”, “Tm”, “Xe”, “Yb”, “Zn”, “Zr”. Simple chemical formulas, though *not* “structure formulas” (2D structure), are in scope for this math format proposal. Single-letter chemical element symbols, “H”, “B”, “C”, “N”, “O”, “F”, “P”, “S”, “K”, “V”, “Y”, “I”, “W”, “U”, need to be written with an explicit **SCI A** to get the correct style in a math expression as defined here. (Context will need to be used to determine if K, for instance, denotes a temperature unit or denotes a chemical element; similarly for several other letters.) Compare *mhchem* and *chemformula* packages for TeX (though they also support 2D structure formulas which is not in scope for this proposal).

This style is also the *fixed* style (i.e., irrespective of any **SCI** styling) for all Sc, currency symbol, characters like “$”, “€”, “£”, “₪”, etc. in math expressions (as defined here).

- **SCI B** MSUB Math serifed bold upright prefix operator. (Cmp.TeX’s \mathbf and \mathbfup.)
- **SCI C** MIT Math italics prefix operator. This must be real italics, not just inclined, that is another math style (**SCI G**). This is the default style for letter-based sequences, except for certain letter sequences (see above and below). Compare TeX’s default math style (and \mathnormal (actually corresponds more to **SCI c**)) with \mathit. TeX default math style and \mathnormal results in bad typesetting for multi-letter variables, like $coefficient$ (presuming juxtaposition of single-letter variables), whereas \mathit (with proper bracketing) gives something like *coefficient*, a single multi-letter variable. In the formats presented here, the distinction is done given by using SPACE between each letter for the former (juxtaposition of single-letter variables, there is no need for INVISIBLE TIMES), and no spaces between letters in the latter (a single multi-letter variable). Using multi-letter variables is common in computing and engineering contexts.
- **SCI D** MBIT Math bold italics prefix operator. Cmp. \mathbfit.

- **SCI E** MUSS Math sans-serif upright style operator. *Always* used for symbol-based combining sequences in math expressions (including punctuation but excluding letter-like symbols and currency symbols). Such symbols in math expressions *cannot* have any other style. This holds for these symbols also in **STX** blocks (see below). (Cmp.TeX’s \mathsf and \mathsfup.) Trying to get another style for such symbols has no effect.
- **SCI F** MBUS Math bold sans-serif upright style prefix operator. Cmp. \mathbfsfup.
- **SCI G** MISS Math inclined sans-serif style prefix operator. (For now, this may be used as the default for other scripts than Latin, Greek, Cyrillic, and Arabic. Note that Arabic has its own set of math styles (see below).) Cmp. \mathsfit.
- **SCI H** MBIS Math bold inclined sans-serif style prefix operator. Cmp. \mathbfsfit.

- **SCI I** MUDD Math upright double-struck style prefix operator, a.k.a. “black-board bold”. Note that double-struck is different from outline. This style is default for “N”, “Z”, “Q”, “R”, “I”, “C” and “H” (unless the C is directly after a “°” symbol); for “natural numbers”, …, “quaternion numbers”. (Cmp.TeX’s \mathbb.)
- **SCI J** MBUD Math bold upright double-struck style prefix operator.
- **SCI K** MIDD Math inclined double-struck style prefix operator. Cmp. \mathbbit.
- **SCI L** MBID Math bold inclined double-struck style prefix operator.

- **SCI M** MSC Math script style prefix operator.(Cmp. TeX’s \mathcal.)
- **SCI N** MBSC Math bold script style prefix operator. Cmp. \mathbfcal.

- **SCI O** MSSC Math swash script prefix operator. (Cmp.TeX’s \mathscr.)
- **SCI P** MBSS Math bold swash script prefix operator. Cmp. \mathbfscr.

- **SCI Q** MFR Math Fraktur (Latin script only) style prefix operator. (Cmp.TeX’s \mathfrac.)

- **SCI R** MBFR Math bold Fraktur (Latin script only) style prefix operator. Cmp. \mathbffrak.

- **SCI S** MTT Math typewriter style prefix operator. Even though it is called “typewriter style”, there is no requirement for this to be a “fixed width” font. (Cmp. TeX’s \mathtt.)
- **SCI T** MBT Math bold typewriter style prefix operator.
- **SCI U** MITT Math inclined typewriter style prefix operator.
- **SCI V** MBITT Math bold inclined typewriter style prefix operator.

- **SCI W** MSCW Math open script style à la Weierstraß, in particular Weierstraß p.
- **SCI $** MDEL Reserved for delayed determination of default style (internal use).

In some contexts, styled Arabic letters are used (in Arabic math expressions, which usually also are RTL). They need to be taken from the *nominal* Arabic letters (i.e., not having a compatibility mapping). Note that there can be no bidi control characters in any math identifier; that is syntactically blocked: any non-letter character terminates the identifier (and bidi control characters are not permitted otherwise either). The following styles apply only to Arabic (nominal) letters, and there *shall* be no mix of LTR and RTL (e.g., Arabic) letters in one letter sequence. The bidi algorithm (just reordering, absolutely no bidi mirroring) is done on individual identifiers and numerals, never on a math expression as a whole. In addition, it is done if, and only if, bidi is enabled outside of the math expression. (The letters after **SCI** below are picked arbitrarily, there is no mnemonicity (beyond them being Arabic letters).) Of course, the Arabic (preshaped isolated) letters in the **SCI** control sequences here do not form part the identifier and does of course not participate at all in the bidi process nor the cursive shaping. (In this document we use a “normal” Arabic font for all the Arabic math style variants.)

- **SCI ﻯ** MAR (arabic letter alef maksura isolated form) Normal style (“upright”, filled, “non-calligraphic”). Default style for Arabic math identifiers and numerals.
- **SCI ﺍ** MARO (arabic letter alef isolated form) Outline style. Note that outline is different from double-struck style.

- **SCI ﺭ** MARB (arabic letter reh isolated form) Bobtail style. The final letter is shaped as initial or medial form, as if there was a ZWJ at the end.
- **SCI ﺩ** MABO (arabic letter dal isolated form) Outline bobtail style.

- **SCI ﺡ** MARL (arabic letter hah isolated form) Looped style. The final Arabic letter of the sequence gets a loop tail.
- **SCI ﻉ** MALO (arabic letter ain isolated form) Outline looped style.

- **SCI ﻝ** MART (arabic letter lam isolated form) Down-tailed style. The final Arabic letter of the sequence gets a swash tail.
- **SCI ﺹ** MATO (arabic letter sad isolated form) Outline down-tailed style.

- **SCI ﻡ** MARS (arabic letter meem isolated form) Up-tailed (“stretched”) style. The final Arabic letter of the sequence gets a tall up-tail.
- **SCI ﻁ** MASO (arabic letter tah isolated form) Outline up-tailed style.

These styles should be followed by an identifier (or numeral) in the Arabic script. If followed by an identifier in another script, the result is implementation defined.

4.3.2 Using HTML/SVG/XML compatible control tags

The math styled element: **<st s="***x***">***id***</st>** corresponds to **SCI** *x id* where *x* is in [**A**-**Za**-**z**] or is an Arabic letter per above, and *id* (which we refer to as an identifier) is a) a non-empty sequence of combining sequences *each with a letter as base character*, b) a letter-like symbol, or c) a non-empty digit sequence possibly with full stops, commas and PSPs. Note that non-letter-like symbols (including punctuation and arrows) cannot be styled this way or any other way (like CSS) in math expressions, nor can currency symbols (however often letter-like). Certain aspects of style (size, colour, strike-through, shadow) are inherited, though, from outside the math expression.

The styling defaults for digit strings and letter strings and symbols as per section 4.3.1 still apply, thus allowing to write just the *id*, rather than the full **<st s="***x***">***id***</st>** for many common cases. These defaults radically cut down on the markup needed. This is a bit “counter” to what is the norm in XML/HTML/SVG, but always having **<st>** tags for each numeral/identifier makes the markup way too heavy, like in MathML or OMML.

CSS styling does *not* apply in math expressions, except for the inherited partial styling, and in **<txt>** blocks (see below). In addition, bidi applies (in isolation) for the *content* of each **<st>** tag (explicit or derived from defaults), *individually*. Automatic bidi mirroring (via the bidi algorithm) *never* applies in a math expression, no exceptions.

4.4 Embedding text

Sometimes one may want to have small pieces of (almost) ordinary text inside of a math expression. Compare **\*text*{**…**}** in TeX math expressions, **<mtext>**…**</mtext>** in MathML. This is useful for true words (not identifiers) used inside a math expression, like “if” or “and”, “def” (could be put above an equals sign), or a (short) piece of explanatory text. It is also the mechanism we will use for such things as hexadecimal numerals.

Inside such an embedded text block, the text style starts out in the inherited (from outside of the math expression) text style, “normal” text styling may be used (like CSS), except that it does not affect symbols, including punctuation and fences, which stay upright, sans-serif and normal weight. Sequences of symbols (including any intermediary spaces) in a text block are implicitly enclosed in a math block (**SME**…**EME**, **<me>**…**</me>**). This implies that a symbol sequence in a text block get the math style for symbols, and they are never reordered or mirrored by bidi processing. Likewise, sequences of no-break spaces in a math block are implicitly enclosed in a text block. In the grammars below, these implicit enclosures are handled as if they were explicit.

Text blocks may contain nested math expressions, where one in turn could have a text block. While mostly intended for short comments or having “word” annotations on symbols, embedded text blocks can thus also be used to change the inherited size for math subexpressions, or change the colour of subexpressions, since embedded text blocks can have internal math expressions. This way one can get not only short pieces of text embedded, but also (e.g.) colour a variable inside an expression:

<me>x+**<txt style="color:red;"><me>**y**</me></txt></me>** $x + {\color{red}y}$

Any style changes affect only the current text block, and any (attempted) style changes in a math expression outside of a text block, are ignored or causes an error.

All line breaking and tab characters are interpreted (and multiple ones in sequence are collapsed to one) as a NO-BREAK SPACE in an embedded text block. Nor is there any automatic line breaking in a **<txt>** block.

As always in a math expression (as defined here) bidi mirroring does ***not*** apply, nor are any bidi controls interpreted in a math text block, not even when they are expressed as HTML attributes or via CSS.

4.4.1 Using C0 control characters

We will (re)use **STX** (START OF TEXT, U+0002) and **ETX** (END OF TEXT, U+0003) as brackets for text embedded in math expressions. (**STX** and **ETX** were originally part of a telegram-like protocol, outside of math expressions; but that protocol is to our knowledge no longer in use.) These brackets nest (in math expressions) *indirectly* via **SME**…**EME**; direct nesting of **STX**…**ETX** is not allowed. If there is a “normal” text styling mechanism available in the contextual system (for instance ECMA-48 text styling), then that may be used within the text block.

Each sequence of space characters, except **SP**, **HT** and *NLF*s, in a math expression is implicitly embraced by **STX**…**ETX** and are thus allowed by the grammar. There are some additional exceptions in matrix layout representation.

Symbols are still in math style in a text block, and sequences of them are never reordered by bidi, they are always left-to-right also in a math text block. Symbol sequence order may affect the semantics of the symbol sequence, and therefore must be under author control, not be bidi reordered. This is done by having each longest *sequence* of symbols, Unicode general categories a) P*, b) S*, plus c) Z* *between* P*/S* (and *NLF*s are counted as **SP**, and is thus Zs here), in an embedded text block is implicitly embraced by **SME**…**EME** and are thus insulated from bidi rearrangement (within the sequence) and automatic bidi mirroring as well as get the math style for symbols also in the text block.

4.4.2 Using HTML/SVG/XML compatible control tags

We use **<txt>**…**</txt>** for text blocks in math expressions. Only inside of **<txt>**…**</txt>** does CSS styling apply in a math expression, except, of course, in any nested **<me>**…**</me>** part(s). And inside of **<txt>**…**</txt>** inline HTML tags may be used (assuming this is embedded into HTML, rather than some other XML based document type). The CCS styling still cannot change that non-letter-like symbols (incl. punctuation and fences) are upright, sans-serif, and normal weight everywhere in a math expression (except embedded text). Recall that CSS styling (or any other outside styling mechanism) changes does not apply in a math expression, except partially in **<txt>**…**</txt>** blocks.

Each sequence of spaces, except **SP**, **HT** and *NLF*s, in a math expression is implicitly embraced by **<txt>**…**</txt>** and are thus allowed by the grammar.

Symbols are still in math style in a text block, and sequences of them are never reordered by bidi, they are always left-to-right also in a math text block. Symbol sequence order may affect the semantics of the symbol sequence, and therefore must be under author control, not be bidi reordered. This is done by having each sequence of symbols, P*, S*, and Z* *between* P*/S* (and *NLF*s are counted as **SP**, and is thus Zs here), in an embedded text block is implicitly embraced by **<me>**…**</me>** and are thus insulated from bidi rearrangement (within the sequence) and automatic bidi mirroring as well as get the math style for symbols also in the text block.

4.5 Positioning layout controls

This is an introduction to sections 4.6 through to 4.11.

Superscripting and subscripting are among the most common layouts for math expressions. They are used for exponentiation, indexing, for giving start and end of summations and integrals, and more. Superscripts/subscripts can be to the right of, to the left of, or above/below (which we will call above-script and below-script) what is superscripted or subscripted. Like for all other math layouts in this proposal, there is no inherent semantics in superscripting or subscripting. That is for a higher level of interpretation. This proposal is only about the (typo)graphical layout.

Some Unicode characters are already superscripted or subscripted. Neither of those should be used in a math expression as defined here. Their use in a math expression is undefined. We will give more general superscripting and subscripting mechanisms that can make any math expression a superscript or a subscript, and "pre-superscripting" (and "pre-subscripting"). Using pre-superscripted/pre-subscripted characters in a math expression would interfere with those more general mechanisms. That does not preclude the use of pre-superscript (<super>) digits in, e.g., unit denotation in "ordinary" text, *outside* of a math expression (as defined here); likewise for pre-subscript (<sub>) digits in say H_2O *outside* of a math expression (as defined here). Indeed, for those uses, pre-superscript/pre-subscript digits would be preferred to using superscripting/subscripting as a style (those are sometimes available as styles) outside of a math expression. Superscript letters are sometimes encoded for phonetic notations, or for very prevalent abbreviations. But that is something else than math expressions.

A math layout group (it need not have grouping brackets; it can be just a letter sequence or a symbol) has 6 positions for superscripts/subscripts. In the representation proposed here, the left side superscripts/subscripts must come before the superscripted/subscripted group. The right-side ones as well as the above/below ones come after the superscripted/subscripted group. For above/below-scripts, if the same position is targeted more than once, the above ones stack up (centred) and the below ones stack down (centred). If either operand starts with a combining character, the superscript/subscript expression is invalid, and should render as an error indication.

A math positioning argument (including the base) can be one of (formulated for the C1 variety):

1. A non-empty sequence of proper combining sequences where the base characters are letters. Included as letters are CJK ideographs and (properly composed) Hangul syllables (from Jamo or precomposed). The letters in the sequence *shall* all have the same bidi category and should all be in the same script. However, an implementation need not support all scripts, but all implementations must support ASCII letters for all math styles that have defaults for some letter sequences. Letters with a compatibility decomposition are not permitted but with some exceptions (see below), nor is U+2118 (use **SCI W**p). (NO-BREAK) HYPHEN is also usable; note that HYPHEN-MINUS is mapped (see below) to MINUS SIGN, not to HYPHEN.
2. A non-empty sequence of decimal digits and certain punctuation (COMMA, FULL STOP, SCI SPACE) starting and ending with a decimal digit, indeed each instance of allowed punctuation must be preceded and succeeded by a digit. A (single) SCI SPACE *between* digits is interpreted as PSP. All the digits shall be in the same script. However, an implementation need not support all scripts, but all implementations must support ASCII digits. Digits with a compatibility decomposition are not permitted.

3. A single letter-based symbol (including integral, partial differential, sum and product symbols), possibly preceded by an **MMS** (**MMS** is intended for RTL math expressions). Such symbols with a compatibility decomposition to a single symbol are not permitted.
4. One of the three above but include a starting math styling operator (see below).
5. A single combining sequence where the base character is a symbol, math or other, including arrows and arrow-like symbols, fences and other punctuation and punctuation-like symbols, but not letter-based nor having a compatibility decomposition, nor U+2044, U+215F, U+27CC, U+2215, U-29F5, U+29F8, U+29F9 (the latter are considered control characters here, more details in the grammar below). It may be preceded by an **MMS** (**MMS** is intended for RTL math expressions). To simplify the grammar a bit, we will allow these to be preceded by a styling control, but that will have no effect on these symbols.
6. A math bracketed (**SME**…**EME**) subexpression or an embedded text (**STX**…**ETX**) subexpression.

There is no other type of argument. Combining characters (without a base character) cannot be arguments to math positioning operators, even if math bracketed (since math bracketing does not allow combining characters at the start of an identifier), nor can a combining character directly follow an **EME**. For instance, placing three dots above a variable, *x*, say, use **STX…ETX** (and here … is not a placeholder, it is the actual text), and place that above the variable (using **AVB**, see below). One cannot use U+1AB4; just using <x, U+1AB4> is a single identifier (applying the U+1AB4 to the *letter*), not an *identifier x* with three dots on top, it is an identifier that includes the three dots.

The operand of one of these operations that is "moved" is usually also rendered slightly smaller (approx. 80 % of the nominal size), to a limit (for cases like subscript upon subscript upon subscript…); there may be just one level of smaller size.

In addition, for a horizontal arrow that gets an above- or below-script, it should be horizontally stretched, by means no specified in this proposal, to cover the horizontal size of the above- or below-script. For horizontal arrows and operators that get an above- or below-script, the vertical distance should be "suitable", so that the above-or below-script does not get too far off the arrow/operator glyph.

Some items that can be above-scripted or below-scripted are also of variable size ("stretch" in particular). For instance: many horizontal arrows, overbraces and underbraces, macron (U+00AF) and circumflex (U+005E); note that these are spacing characters, *never* similar non-spacing characters; the non-spacing characters apply to a base character (letter or symbol), never a math expression. The exact mechanism for this size adaptation (often some kind of stretch, rather than simply font size change which would often look clumsy), is not specified here. This stretch (or even overstretch) is specified by using different variants of the above-scripting or below-scripting operators.

Some targets of subscripting/superscripting are of variable vertical size. Common examples of that are parenthesis/brackets and other "fences", as well as integral signs, summation signs and similar. And, in principle, stretched vertical arrows. Their size can be adapted to the size of another, adjacent, math expression. The placement of the superscripts/subscripts need to adapt to that size adaptation. The exact mechanism for size adaptation (often some kind of stretch, rather than simply font size change which would often look clumsy), is not specified here.

For the HTML/SVG/XML compatible version, note that the tags (operators) need to have an additional parsing (very similar to the parsing done for the C1 version), which is like the parsing done by eqn, TeX, or done in the compilers of any ordinary programming language that has operators with precedences (basically, all but Lisp), not in the rigid and limited way XML tags are normally parsed. The same goes for 4.6.2 trough to 4.12.2, as well as for resolving the default styles for identifiers and numerals.

Since these "operator tags" will initially be read by the ordinary parsing of HTML/SVG/XML parsing, giving a DOM tree, the extra parsing will need to be done on the DOM tree, restructuring it. Note though that **<me>**…**</me>**, **<txt>**…**</txt>**, and **<mx>**…**</mx>** (see below) are already parsed into the proper structure, since they are already brackets, and no need for extra parsing (and DOM restructuring) that is needed for the "operator tags" and identifiers/numerals with default styling.

The extra parsing of the DOM tree will insert new nodes in the DOM tree. In the case of default styling, the extra nodes correspond to **<st>** elements. For the infix/distfix operator tags, the extra parsing will insert "virtual elements", corresponding to the "entities" as given below in the DTD declarations (see section 9.3.5) (these "virtual elements" can be reduced to **<me>** elements in the DOM tree). This change is idempotent, doing it again does not change anything. Not having default styles plus having the "entities" directly as "elements" or require that all operator arguments were explicitly **<me>** (or similar) bracketed, would remove the need for extra parsing of the DOM tree for math expressions. However, that would also make the math expression markup extraordinarily verbose (compare MathML and OMML, which are both quite verbose). A verbosity that is undesirable as well as unneeded with this extra parsing/DOM modification step.

4.6 Above-script/below-script layout

Often math expressions have an above-script or a below-script, indeed there may be several, e.g. and overbrace and above that maybe a text block. These are, layoutwise, not variants of "ordinary" superscripting/subscripting (even though we mixed them above). *Sometimes* above-scripting/below-scripting is, *semantically*, a variant of superscripting/subscripting, line for sums and integrals. But semantics is out of scope for the proposals here, only layout and styling are in scope.

The part that is put above or below another part, is set is a smaller font size. Usually, the same size decrease as for ordinary superscripts or subscripts. But if a (short) text part is put above or below a symbol (like "def" put above "=", "=**SCI\STX**def**ETX**", preferred over U+225D, ≝, since another abbreviation can be used) the (automatic) decrease in font size is greater, suitable for such a combination, the = sign may even be stretched a bit. If the base is a sequence of two or more Arabic letters, and bidi is enabled, the Arabic word may be kashida-stretched to the width of the above-script/below-script. These extra adjustments do not apply to ordinary superscripts/subscripts.

4.6.1 Using C1 control characters

- **SCI /** BLW (SOLIDUS; <U+009A, U+002F>) Binary infix layout operator for below-scripting. Below-scripting the right operand under the left operand (no stretching of the below-script), the base in the pseudo-grammar below, multiple below-scripts stacking downwards. There may be several below-scripts in succession (without needing math bracketing). The below-scripts must come before above-scripts. If the base is a “principally horizontal” symbol, such as a horizontal arrow or equals sign, the symbol may be stretched to (a little bit more than) the width of the below-script.

- **SCI \** ABV (REVERSE SOLIDUS; <U+009A, U+005C>) Binary layout operator for above-scripting. Above-scripting the right operand above the left operand (no stretching), the base in the pseudo-grammar below, multiple above-scripts stacking upwards. There may be several above-scripts in succession (without needing math bracketing). Must come after any below-scripts and before any (right) subscripts. If the base is a “principally horizontal” symbol, such as a horizontal arrow or equals sign, the symbol may be stretched to (a little bit more than) the width of the above-script.

4.6.1.1 Variants which stretch the below-script/above-script subexpression

The **SCI /** and **SCI \** imply no stretching of the below-script/above-script. But sometimes stretching the below-script/above-script to the horizontal width of the base math expression of the below-scripting/above-scripting is desired. In particular when it comes to top/bottom-braces, overbars, circumflexes (note: must use spacing characters, never combining characters, for this), as well as (horizontal) arrows and arrow-like characters. This stretching is not so great for embedded text, but we do not exclude that here (the exact stretch method is implementation defined, but spacing out characters is an option in this latter case).

So we define some variant operators that imply stretching. The means of stretching is not specified here. But for mainly horizontal symbols, an implementation can use for instance some font mechanism, buildup using segment glyphs, or dynamic glyph generation (given character code, font size, target width) or dynamic glyph control points modification (to create a stretch, similar to what is done in multiple master fonts).

- **SCI ⏝** BLWS (BOTTOM PARENTHESIS, <U+009A, U+23DD>) Like BLW, but horizontally stretch the below-script to cover what is above it. Useful for stretching horizontal arrows and similar (or “expressions” based on horizontal arrows) as well as “BELOW” fences like ⏟ which one likely want to span the expression which it is under. The method for the stretching is not specified here, but more than 2 em in width stretched below-parentheses (BOTTOM PARENTHESIS) are rounded only at the ends, and BOTTOM TORTOISE SHELL BRACKET and BOTTOM CURLY BRACKET are stretched only⏡on the horizontal parts of the glyph.
- **SCI** BLWT (BOTTOM TORTOISE SHELL BRACKET; <U+009A, U+23E1>) Like BLWS but with 5% extra stretch.
- **SCI ⏟** BLWU (BOTTOM CURLY BRACKET; <U+009A, U+23DF>) Like BLWS but with 10% extra stretch.

- **SCI ⏜** ABVS (TOP PARENTHESIS; <U+009A, U+23DC>) Like ABV, but horizontally stretch the above-script to cover what is below it. Useful for stretching horizontal arrows and similar (or “expressions” based on horizontal arrows) as well as “ABOVE” fences like ⏞ which one likely want to span the expression which it is over. The method for the stretching is not specified here, but more than 2 em in width stretched above-parentheses (TOP PARENTHESIS) are rounded only at the ends and TOP TORTOISE SHELL BRACKET and TOP CURLY BRACKET are stretched only at the horizontal parts of the glyph.
- **SCI ⏠** ABVT (TOP TORTOISE SHELL BRACKET; <U+009A, U+23E0>) Like ABVS but with 5% extra stretch.
- **SCI ⏞** ABVU (TOP CURLY BRACKET; <U+009A, U+23DE>) Like ABVS but with 10% extra stretch.

Above positioned expressions stack upwards, likewise for below positioned expressions which stack downwards. E.g., a base expression with a stretched above-brace, and comment above that: **SME SME** <*base expr*> **EME ABVS ⏞ ABV STX** <*remark above the above-brace*> **ETX EME**.

The structure of an above-scripted/below-scripted math expression can be given as the regular expression:
base((**BLW**|**BLWS**|**BLWT**|**BLWU**)*b*)*((**ABV**|**ABVS**|**ABVT**|**ABVU**)*a*)*
where *base*, *b*, *a* are math expressions as defined in section 4.2.1, 4.3.1 (including default styled elements), or 4.4.1. Above-scripts stack (upwards) and below-scripts stack (downwards). Note that either or both above-script stack and below-script stack may be empty, and that is indeed the most common case.

4.6.2 Using HTML/SVG/XML compatible control tags

We will use the following control tags (note that they are infix operators, not bracketing):

- **<blw r="***(**n**|s|t|u)***"/>**. Below-scripting binary operators. These tags (which are infix operators) with below-script math expression as second argument follow the base. The **r** attribute (default **n**) indicates stretch. **r="n"** means no stretch; and then **s**, **t**, **u**: stretch as described above for the C1 variant.
- **<abv r="***(**n**|s|t|u)***"/>**. Above-scripting binary operators. These tags (which are infix operators) with above-script math expression as second argument follow the **<blw>** components (if any). The **r** attribute indicates stretch (none, stretch, extra stretch).

An above-scripted/below-scripted math expression has the following structure:
base(**<blw r="***(**n**|s|t|u)***"/>***b*)*(**<abv r="***(**n**|s|t|u)***"/>***a*)*
where *base*, *b*, *a* are math expressions as defined in sections 4.3.2 (including default styled elements), 4.4.2, or 4.2.2. Note that there can be several below-scripts and several above-scripts without the need for math bracketing, implicitly bracketing all of the previous, as a convenience. For a different bracketing than this implicit one, explicit math bracketing must be used.

4.6.3 Note on combining characters

Note that some marks above (or below) math expressions may look like combining marks. But combining marks apply to base *characters* (usually letters, sometimes symbols), never to (math) expressions. So any mark above or below (stretched or not) over/under a math expression *shall* be a non-combining character. E.g., a circumflex above a math expression needs to use U+005E, CIRCUMFLEX ACCENT, *not* U+0302, COMBINING CIRCUMFLEX ACCENT: **SME SME** … **EME ABVS** <U+005E> **EME** (for readability, the circumflex usually needs to be stretched, hence **ABVS** rather than **ABV**). (For dot(s) above/below, use FULL STOPs, or ONE DOT LEADER, TWO DOT LEADER, HORIZONTAL ELLIPSIS; however, this works for single letter variables, it does not work well graphically for other math expressions.)

4.7 Superscript/subscript layout

Apart from getting “superscripts” and “subscripts” above (above-scripts) or below (below-scripts), which we dealt with above, superscripting and subscripting occur to the left of and to the right of an expression (keeping in mind readability, so it may need to be visibly parenthesised, which is *not* done automatically, i.e. must be done explicitly by the author). A superscript on the left side is commonly used with a root sign, and, together with subscripts on the left side, with names for chemical elements (for number of protons and total of number of protons and neutrons). Superscripts/subscripts on the left side of an expression is also useful for RTL math expressions.

4.7.1 Using C1 control characters

We will use the following superscripting/subscripting layout operators:

- **SCI ~** SUPL Left superscript operator. Superscripting the left operand to the upper left of the right operand (actually the *base*, see below).
- **SCI =** SUBL Left subscript operator. Subscripting the left operand to the lower left of the right operand.
- **SCI _** SUBR Right subscript operator (cmp. TeX's _, underscore operator). Subscripting the right operand to the lower right of the left operand.
- **SCI ^** SUPR Right superscript operator (cmp. TeX's ^). Superscripting the right operand to the upper right of the left operand (actually the *base*, see below).

Subscript/superscript corners:
(*a* **SUPL**)?(*b* **SUBL**)?*base*(**SUBR** *c*)?(**SUPR** *d*)?
where *base* is as defined in section 4.6.1, and *a*, *b*, *c*, *d* are as defined in 4.2.1, 4.3.1 or 4.4.1. Note that there is at most one of each of the superscripts/subscripts. If you want more than one of a kind of superscript/subscript, then math bracketing (**SME**…**EME**) must be used, and may also need visible parentheses for readability.

4.7.2 Using HTML/SVG/XML compatible control tags

We will use the following control tags, with no attributes (acting like "infix operators"; actually, they form a "distfix operator", just as for the C1 variant, no explicit bracketing):

- **<lsp/>** Left superscript operator.
- **<lsb/>** Left subscript operator.
- **<rsb/>** Right subscript operator.
- **<rsp/>** Right superscript operator.

Subscript/superscript corners:
(*a*<**lsp/**>)?(*b*<**lsb/**>)?*base*(<**rsb/**>*c*)?(<**rsp/**>*d*)?
where *base* is as defined in section 4.6.2, and *a*, *b*, *c*, *d* are as defined in 4.2.2, 4.3.2 or 4.4.2. Note that these tags are infix *operators*, there is no content for these tags. This is in contrast to the **<sub>** and **<sup>** styling tags in HTML (outside of math expressions) which may appear similar but are completely unrelated; indeed, they are effectively (though not actually) defined via CSS styling. The operators here are instead defined via math expression positioning.

4.8 Vertical (with horizontal line) or slanted (with slash) fractions

Division (or, rather, the notations commonly used for division) has a few slightly different displays in mathematics. One is to use an ordinary symbol, just as done for addition, equality or implication. This ordinary symbol is / (U+002F, SOLIDUS). Other varieties need a bit more work on this level of representation.

But in the math expressions as defined here, we need to be able to write also the other different forms, that use special layout. Note that while we call this "division" or "fraction", the system proposed here gives no semantics to any layout, beyond the layout itself. Assigning a mathematical semantics is for a higher level. However, the division layout operators given here will almost always be used to signify a mathematical division.

It is important to note that the division/fraction operators presented here cannot be mirrored by **MMS** (see below, but they do have mirror control characters), nor are they styleable, thus cannot be argument to an **SCI** [**A**-**Za**-**z**] (most symbols are not styleable either, but for a very different reason, and syntactically symbols can be styled but, for most symbols, is has no effect) or the content of an **<st>** element. Indeed, these operators should be seen as control characters and not as symbols, and for the HTML/SVG/XML version there is a special tag for them, they are not handled as symbols at all. Nit: while symbols can individually be given a colour (by using **STX**/**<txt>** blocks), that is not possible for the control characters for division layout, just because they are control characters. (There is a workaround, left as an exercise for the reader…)

Also the thickness of the line depends on font size only. (There is a workaround, in principle the same as hinted in the previous paragraph.) We will not have an extra attribute for that in the HTML/SVG/XML variant, because that would not be directly transferable to the other two representation variants.

4.8.1 Using C1 control characters and control-type graphic characters

We will use infix controls for fractions: the ones that Unicode already defines, plus one C1 control code (U+0084, which we here define as MATH HORIZONTAL DIVISION, **MHS**) as operators. The operands are as defined in 4.2.1, 4.3.1, 4.4.1, 4.7.1. Note that ordinary solidus can also be used and acts as an ordinary symbol. The following ones we all consider to be control characters, not symbols.

- **MHD** (U+0084, here proposed to be the MATH HORIZONTAL DIVISION control code, cmp. TeX:s \frac, but **MHD** is an infix operator, which is both control and graphic of dynamic length). Inside of a math expression, this is the vertical division infix operator, giving a *horizontal* division bar (of suitable thickness given the current point size) between the operands. The left operand is displayed above, and the right operand is displayed below, and a horizontal line between them the length of which is the max width of the display of the operands, which are horizontally centred above/below that line. For example: "**SME SME** a+b **EME MHD** c **EME**" should display approximately like this:

  $$\frac{a+b}{c}$$

  (but without line breaks, they are external to the math expression, and the arguments a little bit closer to the horizontal bar than they are (were…) here).

  If *both* operands are ***un**bracketed* (no **SME** ... **EME**) digit sequences (no punctuation, no spaces) in default style (explicitly or using the style default mechanism), then the digits are rendered small (approx. index-size). Note that this automatic size reduction only applies to pure digits arguments.

- **DIVISION SLASH** (U+2215). Inside of a math expression, this is the (possibly big) *slanted* division stroke, of dynamic length, with "super" nominator (left argument), and "sub" denominator (right argument) operator. For example, "**SME SME** a+b **EME** <U+2215> c **EME**" should display approximately like this:

  $$^{a+b}/_{c}$$

  (but without line breaks, they are external to the math expression).

  This means that we regard **DIVISION SLASH** as a control character, not an ordinary symbol. Likewise for its “mirror”: **REVERSE SOLIDUS OPERATOR**.

If *both* operands are ***un****bracketed* (no **SME** ... **EME**) digit sequences (no punctuation, no spaces) in default style (explicitly or using the style default mechanism), then the digits are rendered small (approx. index-size). Note that this automatic size reduction only applies to pure digits arguments.

- **BIG SOLIDUS** (U+29F8). Inside a math expression, this is the (possibly big) *slanted* stroke, of dynamic length (as opposed to SOLIDUS), with *level* nominator (dividend) and denominator (divisor) operator (i.e., their math centres align, not offset). Note that any visible parentheses must be given explicitly. Note that there is no automatic detection of when parentheses may be needed; and grouping brackets (**SME** ... **EME**) never generate visible parentheses or brackets of any kind (except perhaps in a “view invisible controls” mode in a text editor). “**SME SME** a+b **EME** / c **EME**” should display like this: $a + b/c$. Note that despite the **SME**…**EME** there are no parentheses. They need to be written explicitly, inside of the **SME**…**EME**, by the author of the math expression. “**SME SME** (a+b **MHD** c) **EME** <U+29F8> d **EME**” should display similar to this (except that parenthesis stretch is glossed over here, see below regarding parenthesis stretch):

  $$\left(a + \frac{b}{c}\right)/d$$

  Indeed “**SME** (a+ b **MHD** c)<U+29F8>d **EME**” should display similar to that too (still glossing over the sizes of the parentheses), but technically the left operand to the division operator (control) is then just “)”, since visible bracketing is not parsed at the layout level.

  This means that we regard **BIG SOLIDUS** as a control character, not an ordinary symbol. Likewise for its “mirror”: **BIG REVERSE SOLIDUS**.

  The special treatment for pure digits arguments does not apply for this one.

- U+2044 (FRACTION SLASH) and U+215F (FRACTION NUMERATOR ONE) are here also regarded as control characters, but we will not allow them in math expressions, as defined here, since a) they are not well defined (despite the name) whether they are for horizontal line or slash line division, b) they seem to be only for digits numerator and denominator, c) they have no mirrors (for RTL math expressions) and they cannot be mirrored via **MMS** since they are (here) considered to be controls, not symbols. These characters can still be used outside of math expressions (as defined here).

Remarks: as noted above, SOLIDUS works as an ordinary symbol, and can be used for division, without the slash being “big” or any other special layout. REVERSE SOLIDUS is often used for set minus, as is SET MINUS (which is actually SMALL SET MINUS, *smallsetminus* in TeX). The binary operator spacing heuristic, see below, give neither of these extra spacing, even if they otherwise appear as binary operators.

Note also that there are no other automatic size changes than those mentioned above (for super/sub/over/under-scripts, and the special handling of fractions with pure digits arguments. All other size changes must be done explicitly (via text blocks).

4.8.2 Using HTML/SVG/XML compatible control tags

Here we will use a new empty tag **<dv/>** (for **MHD**), and none of the controls we used for the C1 based version. Since HTML/SVG/XML is largely allergic to “control” characters (mostly C0, C1, but also bidi controls), why should be allow these controls?

Fractions (horizontal(0)/slanted(1)/slanted-level(2)): *a*<**dv t="***(t-2|t-1|****t0****|t1|t2)***"/>***b*. The operands, *a* and *b*, are as defined in 4.2.2, 4.4.2, 4.7.2. The variants correspond to **MHD** (t0, default), **DIVISION SLASH** (t1), and **BIG SOLIDUS** (t2). These (here) control codes cannot be used in the XML/SVG/HTML compatible variant. SOLIDUS can be used directly, but acts as an ordinary operator symbol, no special (and stretchy) layout. *t-2* and *t-1* are used for the mirrored variants.

As above, also here, when both arguments are pure digit sequences (no **<me>** brackets) with default style (explicit or using the defaulting style mechanism), for the “t0” and “t1” (and “t-1”) variants, the digits are rendered a bit smaller (about 80%) of the current font size.

4.9 Long division (of numbers), adding numbers, subtracting numbers, multiplying numbers

MathML covers so-called long division. Long division is not a mathematical expression per se. It is a notation for a particular *numerical computation* (though long division notation can also be used for division of polynomials). Likewise, the notations for “manually” doing numerical multiplication, or numerical addition or numerical subtraction, including carry and borrow notations where applicable, are not mathematical expressions per se.

These notations for *numerical computations* are of interest. But they are in a separate category and are not covered by the proposals here. The same goes for long division notation for calculating division of polynomials, where the math expression representation(s) proposed here may be used for each of the polynomials, but the long division itself is not (and should not) be in scope for the math expression representation.

4.10 Matrix layout

Using (small!) matrices, with math expressions as “elements” is common. Like: $\begin{matrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{matrix}$ or $\begin{bmatrix} a & \cdots & b \\ \vdots & \ddots & \vdots \\ c & \cdots & d \end{bmatrix}$ (note that the various ellipses here are symbols available in Unicode, and they must be given explicitly in the representations proposed here; there is no behind the scenes filling in of any symbol). They are a little bit like tables but incorporating a full-fledged table mechanism (like HTML tables) as math expressions seems too much. We need a simpler mechanism, tailored for math expression use. Something similar to LaTeX:s *begin*{*matrix*}…*end*{*matrix*}, but in the C1/XML frameworks we are building. Note that we will not automatically include any parentheses, brackets, etc. Those have to be given explicitly, and explicitly stretched vertically (see below) to cover the (vertical) size of the matrix.

The exact same representation mechanism is used also for such things as $|x| = \begin{cases} -x, & x < 0 \\ x, & x \geq 0' \end{cases}$ where the part right of the large brace is a matrix with specific alignments.

The matrix cell contents may be aligned within the cell in various ways. Horizontally, the content of a cell (once its width is determined for the column) may be flushed left, flushed right, centred, or have it aligned on the decimal mark (for numerals, at least two in the column) within the column (this last one may change the width of the column, once). Vertically, the content of the cells of a row are aligned on their respective “math vertical centre”. However, one can in some cases (matrices and fractions) select a subcomponent's "math vertical centre" (to a limit) for that alignment (this may change the height of a row, once).

4.10.1 Using C0 and C1 control characters

Within **SME** … **EME** bracketing **SP**, **HT**, **LF** (and more) are used as separators and code layout. They are not rendered. But math expressions also need to have matrix layouts. It need not semantically be a matrix of any kind; it is just the layout. HT and LF (and more) will then be used not just as "whitespace", but have structural significance.

A math matrix layout is started with **SCI ,** (SCI, COMMA), **SCI .** (SCI, FULL STOP), **SCI ;** or **SCI :**. It is terminated with **SCI #**. For **SCI ,** and **SCI .** the default column spacing is zero width (suitable for laying out a set of equalities with vertical alignment; how to space out things like "if" clauses, without needing to use no-break spaces, we will get back to below). For **SCI ;** and **SCI :** the default column spacing is about 1.5 *em*, as in the examples above. The (default) row spacing is about 1 *en* (0.5 *em*), as in the examples above.

Note that while cell sizes are adjusted while forming rows and columns, expanding "around" each cell's contents, there are no cell margins, but there is row and column spacing between rows and between columns.

We will use **HT**, **HTJ**, **SCI HT** and **SCI VT** to *terminate* the elements (cells) in a row of the matrix layout. ~~A rightmost, in a row, element/cell terminating **HT** can be omitted~~. Note that there are no tab stops within a math expression. Compare TeX's "&" (which, however, is a separator). The different terminators specify different *horizontal* alignment for the preceding cell's content.

For instance, $\begin{matrix}1 & 0 & 0\\ 0 & 1 & 0\\ 0 & 0 & 1\end{matrix}$ can be represented as **SCI;**1**HT**0**HT**0**HT LF**0**HT**1**HT**0**HT LF**0**HT**0**HT**1**HT SCI#**.

- Terminating **HT** ~~(may be omitted after the cell content specification of a rightmost cell)~~: the content is left adjusted in the cell space (*after* the cell space is adjusted into rows and columns, i.e., equal height within the row and equal width within the column).
- Terminating **HTJ**: the content is right adjusted in the cell's space.
- Terminating **SCI HT**: the content is horizontally centred in the cell's space.
- Terminating **SCI VT** (with **SCI LS** as equivalent): the content, which should be a single decimal numeral that may be preceded by a minus sign, but *not* in so-called scientific notation, is decimal aligned with other cells that are **SCI VT** terminated in that *same* column. If there is "excess space" to the right (for *all* those cells), the contents are moved to right (to the left for the rare scripts that have RTL decimal digits) while maintaining the decimal alignment. The alignment is on COMMA if the math matrix layout is started with **SCI ,** or **SCI ;** (or where the comma would be, in case there is no comma in the numeral). The alignment is on FULL STOP if the math matrix layout is started with a **SCI .** or **SCI :** (or where the full stop would be, in case there is no full stop in the numeral). If the content is not a decimal numeral, then the content is centred.

Elements in a row are vertically aligned along the *vertical math centre line* over the entire row. Note, not centre, but *math centre*. For instance, a horizontal division has its vertical math centre at the division line no matter how large the nominator or denominator parts are, and superscripts and subscripts to not change the vertical math centre of an expression.

We will use line breaking characters (including regarding <CR,LF>, <CR,VT> and <CR,FF> as *single* line separators each) to separate the rows in the matrix layout; this is regardless of "platform" (this goes for the C1 version and the "markdown" version (Appendix A); in the HTML/SVG/XML version NLFs are treated as specified in those standards). Compare TeX's "\\". If all the rows in a matrix layout have the same number of cells, the cells are aligned into columns. Otherwise, there are no columns, and the rows are stretched separately/individually to fill out the row so that all rows in the matrix layout have the same width. However, all rows in a matrix layout *should* have the same number of cells; if not, the horizontal cell separation is always the default separation (the column and row separation can be adjusted, see below).

Cell contents may be empty. A "show cell outline" editing mode may show the cell boundaries, rectangular outline, to help in editing a math expression (this may be automatically engaged when the "insert marker" is within a math matrix); normally the cell outlines are not shown.

The start and end matrix layout brackets nest, including that **SME**…**EME** also nest (in the same "nest", not two separate "nests"), though it would be unusual to have a matrix inside a matrix.

If there are **SME**:s that are not closed within the matrix layout, they are all closed by the end matrix layout **SCI#** control. Any "surplus" **EME** controls in a matrix layout cell are ignored (i.e., they cannot close an **SME** from outside the matrix layout, indeed not outside the cell). Note that this is *error recovery* and should be associated with some kind of error indication. It shall not happen in a well-formed math expression as defined here.

For the XML/SVG/HTML compatible variant, the XML/SVG/HTML parsing handles the nesting, and in XML any imbalance is an error, and in HTML would cause a similar error recovery.

Note that visible brackets, parentheses and similar visible "grouping" characters ("fences") must be explicit and never imply any **SME**…**EME** bracketing; the (invisible) bracketing control sequences *never* generate visible fences (a "show invisibles" editing mode not counted).

The "elements" of a matrix layout need not be regular math expressions, it may be fragments like "= *x* + *y*" or "**if** *x* < 10 **and** *y* < 5".

Note again that there is absolutely no semantics implied by this system for math layout and formatting. The author of a "math expression" may lay out a bunch of arbitrary graphical symbols in math expressions, as defined here, or a concoction of just left 'fences' (no need for visible bracketing to match in any way), if it suits the author.

4.10.2 Using HTML/XML/SVG compatible control tags

For the HTML/XML/SVG compatible version, we will use **<mx>**…**</mx>** for matrices. The content is a sequence of rows: **<mr>**…**</mr>**. The content of a math matrix row is a sequence of cells: **<mc>**…**</mc>**. **<mx>**, **<mr>** and **<mc>** have attributes related to horizontal alignment and row and column spacing (the latter will be detailed further below). The **<mx>** will have two attributes corresponding to the **SCI,**, **SCI.**, **SCI;**, **SCI:**.

A math matrix tag, **<mx>**, has an attribute saying on which decimal mark to align numbers (for cells that have numerical alignment). It also has an attribute that gives the default horizontal column separation, none or "about 1.5 em". See the DTD fragment below for more details about attributes to **<mx>**, **<mr>**, and **<mc>**.

A math matrix row has cells that have horizontal alignment attributes (corresponding to **HT**, **HTJ**, **SCI HT**, and **SCI VT**:

<mr><mc h="*(l|r|c|n)*">…</mc>…<mc h="*(l|r|c|n)*"></mr>.

For the *h* attribute, “l” means left align (default), “c”, horizontal centre align, “r” right align, and “n” numerical alignment in the column (comma or full stop as decimal mark as per the attribute to **<mx>**.

4.11 Vertically stretched layout

Sometimes, in math expressions, some symbols should be stretched compared to their non-math expression size (raw size from font). This may happen both above or below an expression (which we have already dealt with, section 4.6), as well as to the left of or the right of an expressions (this section). We here add an operator for left side vertical stretch and an operator for right side vertical stretch.

Vertical stretching often applies to fences (parentheses, brackets, …) and “big” operators, like sum, integral, product, on the left side and on the right side, as well as the root symbol. Compare *left* and *right* in (La)TeX.

We make a little bit of a special case for U+221A, SQUARE ROOT. Despite the name, it is the general “root” symbol, there must be no “2” in the glyph (as that is the default…). When the left operand to the left side stretch operator is based on U+221A (it may have a left superscript), the glyph is stretched not only vertically, but also the top bar is also horizontally stretched over the right argument, obviating visible outer parentheses for the right operand. Similarly for **MMS** < U+221A> (U+221A does not have a mirror character encoded, so we must use the **MMS** control) as the base of the right operand of right side stretch operator made to also horizontally stretch over the left operand. No other symbol is special treated in that way; the “presuperscripted” root symbols encoded in Unicode should never be used in a math expression (as defined here). Note that there is no “SQRT” control code or HTML tag; we use the SQUARE ROOT symbol in Unicode.

4.11.1 Using C1 control characters

We will use a number of **SCI**-based controls with (ASCII) parentheses/brackets as single character part of the control. The left side stretch controls (stretch operator) use (, [, [, and < after the **SCI**, and the right side stretch controls (stretch operator) use),], }, and > after the **SCI**.

- **SCI (** LSS (cmp. TeX’s *left*) Vertically stretch the left operand to cover the size of the right operand. Useful for left-side fences (parentheses, brackets, braces, …) and similar; note that a left side fence need not be a Ps, it can be a Pe or something else, like a vertical line or a vertical arrow, or just contain something vertically stretchable (like an integral sign with super- and subscripts). The method for the stretching is not specified here, but more than about 3 *em* in height stretched parentheses are rounded only at the ends. If the left side is a root sign (possibly with left superscript), the root sign is stretched vertically *and* horizontally stretched to “cover” over the right-side operand. (Note that presuperscripted root signs should not be used.)
- **SCI [** LSST Like LSS but with 5% extra stretch.
- **SCI {** LSSU Like LSS but with 10% extra stretch.
- **SCI <** LSSV Like LSS but with 15% extra stretch.

- **SCI)** RSS (cmp. TeX’s *right*) Vertically stretch the right operand to cover the size of the left operand. Useful for right-side fences (parentheses, brackets, braces, …) and similar; note that a left side fence need not be a Pe, it can be a Ps or something else, like a vertical line or a vertical arrow, or just contain something vertically stretchable (like a mirrored integral sign with leftsuper- and leftsubscripts). The method for the stretching is not specified here, but more than about 3 *em* in height stretched parentheses are rounded only at the ends. If the right side is a mirrored root sign (possibly with a right superscript), the root sign is stretched vertically *and* horizontally to “cover” the left side. (Note that presuperscripted “mirrored” root signs should not be used.)
- **SCI]** RSST Like RSS but with 5% extra stretch.
- **SCI }** RSSU Like RSS but with 10% extra stretch.
- **SCI >** RSSV Like RSS but with 15% extra stretch.

Left/right side stretch:
  (x (**LSS**|**LSST**|**LSSU**|**LSSV**))*base((**RSS**|**RSST**|**RSSU**|**RSSV**)y)*
where *base* is as defined in section 4.10.1, and *x*, *y* are as defined in 4.2.1 or 4.3.1 or 4.4.1.

Note that there can be multiple “stretch items” on either side. For instance multiple fences on either side, even of varying (outwardly growing…) size. As a convenience, one can do that without having to use explicit **SME**…**EME** bracketing.

The method of stretching is not specified, but can range from composition to glyph generation

4.11.2 Using HTML/SVG/XML compatible control tags

We will use one operator (as a tag) for left side stretch (like TeXs *left*, but with an attribute for extra stretch) and one for right side stretch (like TeXs *right*). The attribute values correspond to the variants in the C1 version.

Left/right side stretch:
  (x<lss s="(n|t|u|v)"/>)*base(<rss s="(n|t|u|v)"/>y)*
where *base* is as defined in section 4.10.2, and *x*, *y* are as defined in 4.2.2 or 4.3.2 or 4.4.2.

4.12 LTR layout, sequences

We have now dealt with various ways of building up expressions from “subexpressions” (including symbols, variables, numerals). Well, not really subexpressions, but subcomponents. These subcomponents need not be expressions in any sense, but can be punctuation, like parentheses (commonly an argument to the vertical stretch layout operator). But what about something so simple as “1 + 1”? This is a sequence of subcomponents. Since the representation forms proposed here do not deal with semantics at all, we regard this simply as a sequence of subcomponents, not an expression. Further, in an example such as “1 + 2 × 3”, that is also just a sequence of subcomponents. There is no notation, in this proposal, of priorities of (math) operators (there is, of course, for the invisible layout operators, without giving them any semantics). So “1 + 2 × 3” is just a sequence of five subcomponents. Note that there are many “math operator symbols” in Unicode, and trying to assign a priority to each would a hopeless task, and may even be different in different use areas for the “math expression” representations specified here. Note that the situation is the same also in TeX and the math expression representation there. Any additional parsing, such as matching parentheses, (visible) operator priority resolution, etc. is left to a higher level parsing, and is likely to be dependent on “math area” (which need not be math… for instance, chemical reaction formulas are in scope…). Note also that “**SME** 1 + 2 **EME** × 3” and “1 + 2 × 3” display exactly the same, since the math bracketing controls are never displayed as parentheses (or the like); compare TeX where “{1 + 2} \times 3” and “1 + 2 \times 3” display exactly the same (as “1 + 2 × 3”). (One may want to have an auxiliary checker warning about such things, but that is out of scope for the proposal here.)

Math components in a sequence are always laid out left-to-right. That holds also for RTL math expressions. Bidi is *never* applied to a math expression as a hole, *only* to *individual* identifiers (and numerals, though very few scripts have RTL numerals), in an RTL script. In particular, *automatic* bidi mirroring is *absolutely never* applied in math expressions (that still allows for *explicit* mirroring using a math control, **MMS**, see below, or simply by an editing change to an already encoded mirror symbol). This is vital for the reliable layout of math expressions. Bidi processing is notorious for messing up the semantics of whatever plain text it is applied to. Bidi controls, and symbol mirroring compound the problem. It is paramount to avoid all and any problems that bidi processing has. Therefore, bidi processing is totally ruled out except for the very limited use on individual identifiers. (This very same restriction on bidi processing should be applied to various source codes, like C++, XML, if at all appling bidi processing for those.)

The LTR laid out sequence of “components” is what goes between the invisible math open control and the invisible math close control: **SME**…**EME** or **<me>**…**</me>**.

There is no parsing of parenthesis or other visible fences, no parsing using (visible) operator precedences. The reason is that there are multiple non-trivial conventions for fences, and that there are also many potential operators, and multiple conventions for visible operator precedencies and associativities. Staying out of non-layout parsing is nothing unique for the proposal here. TeX does the same, as well as any other reasonable (and generally useful) math expression coding for traditional math layout. Doing parsing that recognises fences, operators and their precedences etc. are left to an additional layer of parsing that is specific to the conventions of interest. These will of course differ considering which conventions are used and are out of scope for this specification.

The items in the list are aligned on their vertical math centres.

4.12.1 Using C1 control characters

A sequence of zero or more math expressions according to 4.2.1, 4.3.1, 4.4.1, 4.6.1, 4.7.1, 4.8.1, 4.10.1, 4.11.1. Whitespace, **SP**, **HT**, *NLF*, may occur around the math expressions in the sequence, and may be needed to break, e.g., letter sequences into multiple letter sequences. Note, however, that in a matrix representation, **HT** is not whitespace, but a cell content terminator; the cell content is otherwise represented as a sequence according to this section. Also, in a matrix representation an *NLF* is a row separator, not whitespace. (Inside an explicit **SME**…**EME** in a cell, **HT** and *NLF* are whitespace.) Whitespace (not counting no-break or specific width spaces) is collapsed, effectively though maybe not explicitly, to a single **WJ** (WORD JOINER; which is actually a word *separator*).

4.12.2 Using HTML/SVG/XML compatible control tags

A sequence of zero or more math expressions according to 4.2.2, 4.3.2, 4.4.2, 4.6.2, 4.7.2, 4.8.2, 4.10.2, 0. Whitespace, **SP**, **HT**, *NLF* (limited to those allowed in XML/HTML/SVG), may occur around the math expressions, and may be needed to break letter sequences into multiple letter sequences. Rows and cells in a matrix are here always begin/end bracketed, so no special meaning for **HT** or *NLF* there, in this variant representation. Whitespace (not counting no-break or specific width spaces) is collapsed, effectively though maybe not explicitly, to a single **WJ**.

5 Spacing

Ideally, the spacing (both vertical and horizontal) should be "just right". However, that may be difficult to achieve, and here we do not have the goal of enabling all possible desires in how to space parts of math expressions. But we will specify some basic spacings and some spacing tweaks.

Unicode has several space characters, there are currently 17 characters with general category Zs. Most of them are no-break fixed width. But in order not to have to explicitly insert lots of fixed width space characters, we will define some heuristics (like done in TeX and other systems).

First, we split the basic elements of math expressions into subclasses:

1. We already have identifiers and numbers.
2. Split up the symbols into disjoint sets (a. to e. are excluded from the heuristic):
   a. letter-like symbols (general category Sm and So that are based on letters),
   b. default opening fences (general category Po),
   c. default closing fences (general category Pe),
   d. list item separators (comma, semicolon, and like characters),
   e. slashes: /, \, … (including vertical bar, |): though often used as binary operators, they do not by default get binary operator extra spacing, some are sometimes used as fences, such as |; note that the "slashes" that we here classify as controls are (here) not symbols (like "big solidus"),
   f. all other symbols (potential unary or binary operator symbols).

Note that the "binary operator extra space" must never be included in the glyphs in the (math) fonts. Whether to have that extra spacing is either computed via the heuristic, or requested by controls (**SCI+**, **SCI-**). The glyphs in the fonts are always *without* the extra space for binary operators.

5.1 Bounding boxes

For each math layout component, a bounding box is calculated. A bounding box for a math expression component is a trapezoid with horizontal bottom and horizontal top. The bounding boxes are used to position above- and below-scripts, including their stretch, position superscripts and subscripts, and position and stretch stretched left/right components.

Sometimes one may want to use a rectangle instead of a trapezoid. There is a control attribute for requesting that (**SCI @**; t="RECT"). See section 5.6 below.

The bounding box should "closely circumscribe" the (would be, if phantom, see section 5.7 below; **SCI***, **SCI|**; d="PHANTOM", d="VPHANTOM") display of the enclosed math expression. It should not be too tight, needing a little bit if typographical spacing.

The bounding box for a component that has a "division stroke" (either with **MHD** or the slash alternatives) as main layout operator is always rectangular. The same goes for a component that has an actual vertical stretch as main layout operator, as well as for math matrices.

5.2 Binary operators

Conventionally, a binary operator; except division operators (most of which, except /, are made via control codes in this representation system), set minus and dot between unit symbols; in a math expression get a little bit of extra spacing before and after the operator. This extra spacing must **not** be built into the “glyphs” in (math) fonts, since the same symbol may be used without (extra) spacing (usually prefix or postfix operator, though in some cases also infix operator), and with (“extra”) spacing (usually an (infix) binary operator). The spacing is added by the math layout system, *not* the font.

In order not to have to deal with the extra spacing for binary operators explicitly all the time, we will use a heuristic. It is intended to often get it right, but when it does not, there are controls to turn on (**SCI+**) binary operator extra spacing for a symbol, as well as turn it off (**SCI-**). The heuristic will be applied to class (f) symbols, per above. For a class (f) symbol, we consider it binary if both:

- just before it (skipping “ignored whitespace” in between) is a component based on either a) a default closing fence (i.e., general category Pe), b) an **EME** (C1 variant) or an **</me>** (XML/SVG/HTML variant), c) a number, d) an identifier, or there is a non-leftmost matrix cell start;
- just after it (skipping “ignored whitespace” in between) is a component based on either a) a default opening fence (general category Ps), b) an **SME** (C1 variant) or an **<me>** (XML/SVG/HTML variant), c) a number, d) an identifier, or there is a non-rightmost matrix cell end.

Otherwise handle the symbol as a unary operator (or a slash operator, category (e)), not adding extra spacing.

(A previous formulation did not say “based on”, but was more textual. However, this check is likely to be done after parsing, looking at the parse tree, so the present formulation is then more direct to implement. Mostly the “guess” (binary/unary) will be the same, but not always.)

Note that this still may “guess” wrong. For instance, sometimes a default opening fence is used as a closing fence and v.v. Thus, this is not perfect. To correct a “wrong guess” (by that heuristic), one can add **SME**/**EME** or **<me>**/**</me>** at places, or alternatively explicitly say to add or not “binary operator spacing” before and after a symbol. In the C1 version: **SCI -** before a class (f) symbol says not to add any binary operator extra spacing regardless of the result of the heuristic, and **SCI +** before a class (e) or class (f) symbol says to add binary operator extra spacing regardless of the result of the heuristic (neither of these apply to fraction controls, which we regard as controls rather than symbols, nor to default fences). For the HTML/XML/SVG compatible version **<st b="y">**...**</st>** says to add extra spacing, while **<st b="n">**...**</st>** says not to add extra spacing (b="h", the default, says to use the heuristic).

5.3 Fences

Fences that have general category Ps or Pe do not get any extra spacing. Since, rarely, default fences can be used as binary operators, one can give a fence binary operator spacing via **SCI +** (or **<st b="y">**). Other characters that may be used as fences may be given an extra spacing by the heuristic, but that can (and should if they are used as fences) be supressed by **SCI -** (or **<st b="n">**).

5.4 List item separators

List item separators, type d above, automatically get a suitable extra space “after” it (this extra space must not be included in the glyphs from the font, it is added as a part of the math expression layout). For “,” and “;” “after” means “on the right side of”, for “⹁” and “⁏” (suitable for RTL math expressions) “after” means “on the left side of”. **SCI -** can be used to remove this extra space for a list item separator punctuation symbol. Note that commas that occur in numerals are considered part of the numeral, not a separate symbol, and thus do not get any spacing after them.

5.5 Automatic space at certain boundaries

In some cases, it is handy to get an automatic narrow/medium space. E.g., instead of “sin$x$”, automatically get “sin x” (automatically inserting a four-per-em space). Likewise, instead of “6mm”, automatically get “6 mm”. Here we look at the adjacent components of a sequence of math expression components. Components that are composed (division, superscripted, etc.) are not referenced here, i.e. we look at these after parsing, not the textual representation; so for instance “33 1 **MHD** 3” should be displayed (approximately) as “$33\,\frac{1}{3}$”, *without* 1 em space between them, even though “33” and “1” are textually adjacent in the source.

1. A numeral adjacent to (either order; skipping collapsable, to WJ, whitespace) either a) an upright identifier, b) one of the symbols %, ‰, ‱ or c) a currency symbol: automatically add a four-per-em space between them.
2. Italic/slanted identifier adjacent to (either order; skipping collapsable, to WJ, whitespace) either a) an upright identifier, b) one of the symbols %, ‰, ‱ or c) a currency symbol: automatically add a four-per-em space between them.
3. A numeral adjacent to (skipping collapsable, to WJ, whitespace) a numeral: automatically add a 1 em space between them. Placing numerals adjacent to each other without an explicit visible operator, or list separator, is not recommended.

If that automatic space is not desired, one can use, e.g., an empty math expression (**SME EME**, **<me/>**) in between them.

Note that an upright numeral followed by an italic identifier does *not* get extra space: e.g., $2x$.

5.6 Trapezoid bounding box to rectangle bounding box

The bounding boxes, which helps to position a math subcomponent with other subcomponents, is not always a rectangle. It is often a parallelogram (for italic/slanted: ids, numbers, letter-like symbols), or a trapezoid with horizontal top and base (e.g. variable with a superscript). This is used to place superscripts and subscripts close to a variable or an (italic) letter-like symbol.

But sometimes one may want to “straighten up” the trapezoid by converting it to a closest enclosing rectangle. Thus positioning superscripts and subscripts as if the variable or (italic, letter-like) symbol was rectangular. We here will do this by first enclosing the subexpression with **SME**…**EME**, and then have **SCI @** (RECT) directly after the **SME**. For the HTML/SVG/XML compatible variant, we use an attribute to **<me>**: **<me t="RECT">**.

5.7 Phantom components

Sometimes one wants to try to get a line-up of some kind within a math expression. Often, the best way to do that is to use a matrix layout. But sometimes that is not so handy. It is sometimes useful to just get the space of an expression, but not really include the expression in what is displayed (or, for that matter, formula manipulated). Like MathML and OMML we allow parts of a math expression to just be converted to fixed size space (just its trapezoid, not showing the content), the size of that space being equal to the space the part would take if it had been included. This is not just transparent, it is more than transparent; transparent glyphs can still get a shadow, and they are still a part of the text. Phantom components are not part of the math expression, just the space it would take if shown; changing the colour will not show them, though "show invisibles" or "show code" will display them (but those editor features are out of scope for this proposal).

In the C1 version, we use the prefix **SCI *** (PHA) just after **SME** to keep just the trapezoid (with horizontal top and bottom) space, not displaying the math expression. For the HTML/XML/SVG compatible version, we use an extra attribute to **<me>**: **<me d="PHANTOM">**.

Sometimes one only wants to keep the vertical component of the spacing (pure vertical, no inclination), letting the horizontal component have zero length (and implicitly doing rectangularisation). In the C1 version, we use the prefix **SCI |** (VPHA) just after **SME**. For the HTML/XML/SVG compatible version, we use an extra attribute to **<me>**: **<me d="VPHANTOM">**.

5.8 Math matrix row spacing

There are no cell margins other than the size adjustment for the row and the column for math matrices, depending on the sizes of other cells in the same row (max height adjusted for math vertical centre alignment) and column (max width adjusted for decimal alignment). However, there is a spacing between the rows and between the columns. The spacing here is *between* rows, not before or after a row, and *between* columns, not before or after a column.

By default, rows in a math matrix have a 0.5 em separation. Sometimes one may want to have a somewhat smaller spacing between two rows. And sometimes one may want a somewhat larger spacing between two rows. However, one would usually want to make the same adjustment for all row pairs in a matrix, even though that is not required by the mechanism we will use.

For the C1 version: Use a sequence of **PLU** directly after the *NLF* between rows to decrease the spacing between those two rows. Each **PLU** decreases the spacing by 1/6 em. The resulting spacing must still be positive, so rows will never overlap. Instead use a sequence of **PLD** directly after the *NLF* between rows to increase the spacing between those two rows. Each **PLD** increases the spacing by 1/6 em. An implementation may limit the number of **PLD**s in the sequence that are interpreted. All other occurrences of **PLD** or **PLU** (in a math expression) are ignored or may cause an error message.

For the HTML/SVG/XML compatible version, we add an attribute to the **<mr>** (math matrix row) tag: **<mr r="*n*">**, where *n* is a small integer number (including negative numbers), where the number corresponds to the number of **PLD**s (negative value refers to **PLU**) in the C1 version of this format, indicating the spacing adjustment *between* the previous row and this row. If there is no previous row, the attribute has no effect. If the number is out of bounds, the effect is limited.

5.9 Math matrix column spacing

The default column spacing is either 0 em or 1.5 em (depending on the spacing setting for the matrix). Like for rows, one may want to adjust the spacing *between* columns (there is no margins for cells). One could do that by using no-break spaces or phantom expressions. But that is cumbersome, and it may be hard to get the extra spacing desired. So we introduce another mechanism.

For the C1 version: Use a sequence of **SCI &,** just after the tabulation terminating cells in the top row, but not after the last cell the row, to increase the spacing *between* columns. 1/6 em (1/3 en) per **SCI &**. It is interpreted only when occurring on the top row of a matrix, and also only when the matrix has columns (i.e., equal number of cells on all rows of the matrix). For **SCI :** or **SCI ;** matrices, one may use a sequence of **BS** (where **BS** is U+0094) to decrease the spacing between the columns by 1/6 em per **BS**; but using **SCI .** or **SCI ,** with **SCI &** is preferred in this case, avoiding to use **BS**. However, the spacing will never be negative, so cells will never overlap. All other occurrences of **SCI &** or **BS** (in a math expression) are ignored and may cause an error message. These controls terminate, rather than separate, cells in math expressions as defined here, but for the rightmost column any **SCI &** or **BS** are ignored. As shortcuts, **SCI '** gives an increase of 1 em (6 **SCI &**), and **SCI "** gives an increase of 3 em (18 **SCI &**).

For the HTML/SVG/XML compatible version, we add an attribute to the **<mc>** (math matrix cell) tag: **<mc c="*n*">**, where *n* is a small integer number (including negative numbers), where the number corresponds to the number of **SCI &**s (negative value refers to **BS**) in the C1 version of this format, indicating the spacing adjustment between the previous column and this column. If there is no previous column, there are no columns or the cell is not in the top row, the attribute has no effect. If the number is out of bounds, the effect is limited.

6 Math expression vertical math centre

In many scripts (not all), the glyphs composing a text are aligned on the baseline. While “one-liner” math expressions can be said to have a baseline too, that does not work for horizontal line divisions, math matrices and other compositions. Instead we will, conventionally, use a math vertical centre, which for a “one-liner” is at the level of the glyph for MINUS SIGN. While the horizontal centre of a math expression coincides with the true horizontal centre, the vertical math centre of a math expression is often not coincident with the true vertical centre. In addition, we will also have a few controls that select another math vertical centre (picking up a vertical math centre from a subcomponent instead of using the default vertical math centre).

6.1 Vertical math centre for styled elements

The vertical math centre for styled elements (identifiers, decimal numerals, symbols) and for embedded texts (which are each single line, since any *NLF* inside a text block is interpreted as **SP**, and there is no automatic line breaking in a text block) is the vertical position of the MINUS SIGN glyph (at the current font size). This should also coincide with the vertical position of the horizontal line part of the PLUS SIGN and of the horizontal single arrows and the vertical mid-point of many other symbols like equals sign, horizontal double arrows, and more.

6.2 Vertical math centre for sequences

The sequences of math components that occur inside **SME**…**EME**/**<me>**...**</me>** are aligned on their vertical math centre, which becomes the vertical math centre of the sequence. The same goes for the sequences within matrix cells, and indeed each row of cells.

6.3 Vertical math centre for superscripted/subscripted expressions

When superscripting/subscripting (to the left or to the right) or above- or below-scripting, the vertical math centre is the vertical math centre of the “base” of the super/sub/above/below-scripting.

6.4 Vertical math centre for fractions

For horizontal line fractions, the vertical math centre is the position of the horizontal line. For fractions that use the DIVISION SLASH or BIG SOLIDUS controls (or the **<dv/>** tag control, with attribute), the math vertical centre is at the vertical centre of the slash line. (The exact way to draw and position this generated vertical line is not specified here but should be done according to “conventional math expression typographical practice”).

However, this can be altered to some extent by controls/attributes; see below.

6.5 Vertical math centre for matrices

The default vertical math centre of a math matrix layout is the true vertical centre of the matrix layout. However, this can be altered to some extent by controls/attributes; see below.

6.6 Stretched side elements

Adding (stretched) side elements, like square root or fences, the combined vertical math centre inherits the math vertical centre of the base component (what is “inside” the side elements).

6.7 Selecting alternate vertical math centre

Two types of math expressions, as defined here, allow for picking another vertical math centre than the one computed as above. It is fractions and matrices. For fractions one can select either the nominator’s vertical math centre or the denominator’s vertical math centre, as alternative to the division line position. For matrices one can select the math centre of one of the rows (0 (bottom row) to 9 only) vertical math centre as the vertical math centre of the matrix instead of the true vertical centre of the matrix layout.

6.7.1 Using C1 control characters

- For fraction control characters, **MHD**, U+2215, U+29F5, U+29F8, U+29F9, an **SCI 0** after the fraction control character selects the denominator’s vertical math centre as the vertical math centre for the fraction. An **SCI 1** instead selects the nominator’s vertical math centre. One can use a matrix layout with just one cell to use the true vertical centre as the vertical math centre of the fraction.
- For matrices, an **SCI 0** after the (start) **SCI** [**,.;:**] selects the bottom row’s vertical math centre as the vertical math centre for the matrix instead of the matrix. An **SCI 1** selects the vertical math centre of the second row from bottom. Similarly for **SCI 2** to **SCI 9**,

assuming that the matrix has enough rows. Selecting a row that does not exist has no effect, and the vertical math centre is computed as per default.

6.7.2 Using HTML/SVG/XML compatible control tags

We use an additional attribute to the **<dv>** and **<mx>** tags: the **v** attribute, with allowed values "c" (math vertical centre as default), and "c0", …, "c9" reference a row as per previous section. For **<dv>** only "c", "c0" and "c1" are allowed. Referring to a non-existent row results in default calculation of math vertical centre.

7 Bidi and RTL math expressions

In modern Unicode text support, it is common to also support the bidi algorithm, for supporting scripts that are (normally) written right-to-left, like Arabic, Hebrew, and a few other scripts. One cannot directly apply the bidi algorithm on text that contain math expressions. It has to be applied with great caution, to preserve the integrity of the math expression as written and to be presented. The bidi algorithm is notorious for messing things up, when the text is not simple prose. There cannot be an unreliable change of order of arguments, nor an unreliable change in which operator or fence is used in a math expression.

The bidi algorithm is specified for purportedly "plain text". So the specification misses out on many aspects of text, leaving implementors of various text based formats much to their own devices regarding bidi. Texts have not only "plain" paragraphs, but they have structure, there are quotations, headings, images, vector graphics (which may contain text) and tables (which very likely contain text in turn). So this structure must first be found and handled properly. Each part of the structure may have different requirements on the bidi algorithm and its applicability (assuming that bidi is at all supported or enabled; that is *not* required). For instance, a part of the text may be *source code* in a computer programming language or data language. For these it is best to disallow bidi processing, or heavily restrict it to only identifiers (each in isolation).

Math expressions (embedded in a text) are one type of text structure that needs to be handled specially w.r.t. bidi: each math expression being an atomic entity in their paragraph, with special bidi handling inside of the math expression. The special handling is a bit similar to the special handling needed for computer program/data source code (if bidi is at all applied to those).

This means that the bidi algorithm has to be strongly restricted in a math expression, essentially to only identifiers (each in isolation), and so-called bidi mirroring (via the bidi algorithm) must be *completely ruled out* in math expressions (also in embedded text blocks). That is, change of order of arguments and change of symbol (for instance to a corresponding mirrored symbol) are *editing* operations, *not* display operations.

7.1 Insulation from text bidi algorithm

Math expressions shall be *insulated from (overall) bidi processing*. A math expression shall be seen as an atomic object for the bidi algorithm, just like a table, image, vector graphic, quotation, etc. must be. Inside of a math expression, bidi applies *only* to individual identifiers (and, for two scripts, numerals) as well as embedded text items (**STX … ETX**; **<txt>**…**</txt>**), *each in isolation* (assuming that bidi processing is at all enabled). Indeed, since identifiers (in a math expression, as defined here) must be made of letters (only!) of the same bidi category, bidi processing need only be applied to identifiers in a bidi script (just reversing the letter string).

And, bidi mirroring shall *never* be applied inside a math expression, not even inside **STX … ETX** (**<txt>**…**</txt>**) embedded text blocks. That goes for all kinds of symbols (indeed sequences of symbols), whether “operators”, arrows (or arrow-like), or “fences” (parentheses, brackets, etc.) or other punctuation. There must be *absolute* reliability about which mirror version is displayed, since there is a radical change of semantics when mirroring a symbol.

Indeed this of course applies to the math expression as a whole; swapping arguments around radically changes the semantics of a math expression. While the here proposed math layout and styling system does not imply any semantics, the styling and layout must of course respect the integrity of all and any semantics that may be applied. Hence no overall bidi processing or swapping of arguments, and no automatic mirroring, for display, of symbols of any kind inside a math expression. (Editing support like “replace symbol by its mirror” is of course permitted.)

Thus, bidi does not apply to math expressions, with two exceptions and *only* if bidi is enabled outside of the math expression. The *paragraph direction* (a bidi algorithm input parameter) for bidi is always LTR for an identifier/numeral as well as for embedded texts (each individually).

1. Individual math styled parts (in isolation), whether explicit or default. The entire name (or numeral) will either be left as is or totally reversed (a variable/function name in Arabic for instance); note that identifiers and numerals must have letters of the same bidi class or (respectively) have digits in the same script (and hence the same bidi class).
2. **STX … ETX** blocks, where bidi controls are *not* interpreted (or even allowed), and bidi mirroring is *not* applied. HTML bidi attributes, in tags in the text block (**<txt>**…**</txt>**), are *not* interpreted. Note that (longest) sequences of P*, S*, Z*, except for Z*+, between letters/digits (*NLF*s are counted as SP), are implicitly embraced by **SME**…**EME** or **<me>**…**</me>** inside the text block (even if those characters are coded as HTML/XML character escapes).

7.2 RTL math expressions

That math expressions in the systems presented here are always laid out in LTR direction does not prevent so-called right to left math expressions. RTL math expressions is sometimes used in Arabic or Hebrew script contexts, or when using another script that is generally written right to left. It is just that the math expression is given in “visual order”, and that is in order to guarantee semantic stability of the math expression, likewise for never applying automatic symbol mirroring. Left side of an operation is always the left side, etc. Bidi can be applied in math expressions, but only to identifiers and numerals in isolation, and, with restrictions, to embedded texts (**STX**…**ETX**, **<txt>**…**</txt>**). Symbols (including punctuation), in math expressions, always are in math style, and are never mirrored, not even in embedded texts.

Note that while many math symbols do have the bidi-mirrored property set to “true”, and likewise for most “fences”, but arrows and arrowlike symbols have bidi-mirrored set to “false”. For math expression use, that is, well, incongruent. But as mentioned, in a math expression there must never be any bidi-mirroring of any symbol (S*, P*) at all during the display stage. We cannot allow any “display time” mirroring of any symbol, because that is way too unreliable and will change the semantics of a math expression. Any mirroring must be done as an *edit* replacement operation (which may have software support, whether bidi-mirrored=true or not, to make it easier for the math expression author).

Another reason for not doing argument “reversal” (here and there) and symbol mirroring automatically is another quirk for right to left math expressions: Some operations never should be argument swapped and operator mirrored, in particular minus, where the operator is symmetric, so an argument swap would be utterly confusing. Often the same applies to division, definitely when using a horizontal line, where swapping the augments would be utterly confusing. But it often also applies when using a slash for division. For the latter, however, we allow (if the math expression author so wishes) to use mirror *control* characters (in the C1 version): U+29F5 (REVERSE SOLIDUS OPERATOR), U+29F8 (BIG REVERSE SOLIDUS), or use the symbol \ (REVERSE SOLIDUS) (and then likely use / (SOLIDUS) for set minus with swapped (by the author) arguments). In addition, the bidi algorithm has no idea about argument structure anyway, so the moving around would be completely erratic.

“Automatic” bidi symbol mirroring *must never* be used when displaying a math expression. Note that arrows in Unicode are not formally mirroring anyway. For consistency, and, importantly, reliability in display, automatic mirroring is *not ever* applied to any symbols within a math expression. Furthermore, bidi control characters are *ignored* within a math expression.

For the bidi algorithm, a math expression (from outer start bracket to outer end bracket) must be regarded as a single(!) “character” with bidi class ON (Other_Neutral). This is like how embedded images or embedded line drawings (vector graphics) should be handled. Inside the math expression, it is only inside embedded texts and for individual identifiers and numerals (each in isolation) that the bidi algorithm may be invoked (and then without bidi mirroring), if it is enabled outside of the math expression.

7.3 Glyph mirroring (inserted via an edit operation only)

As explained above all mirroring must be done as character substitutions, as edit operations. To aid in this mirroring edit, one can use the data in the *BidiMirroring.txt* file and the *ExtraMirroring.txt* file. For instance, use the entry (from *ExtraMirroring.txt*)

```
            21F6;2B31 # THREE RIGHTWARDS ARROWS
```

to map U+21F6 `THREE RIGHTWARDS ARROWS` (⇶) to its mirror, U+2B31 `THREE LEFTWARDS ARROWS` (⬱). This may well have programmatic support, but it is *not* a display operation.

This does leave a small problem mostly for right to left math expressions: Some symbols (here we include punctuation and arrows/arrow-like) do not have an allocated “mirror character”, even though most do. This holds for some bidi-mirrored characters as well as some that do not have bidi-mirrored set to “true”. These are called out in *BidiMirroring.txt* and in *ExtraMirroring.txt* as comments at the end of each file.

One would expect to use only ∫ in an LTR (left-to-right) math expression, not the mirror of that. And there is no mirror character for ∫ encoded. However, in a right-to-left (RTL) math expression one may use the mirror of ∫. We will use a control code for doing mirroring of symbols that do not have a mirror character allocated. We will use **MMS** (MATH MIRROR SYMBOL) which we here define as the control code U+0099, a hitherto undefined C1 control code. We will use **MMS** ∫ for denoting a mirrored integral sign, and likewise for mirroring other symbols that do not have a mirror character encoded (see Annex D). Another example: To get the symbol “lazy s”, ∽, (which does *not* have an encoding in Unicode), use **MMS** with U+223E (INVERTED LAZY S, ∾), <**MMS**, U+223E>: **MMS** ∾.

A disadvantage with this “visual” encoding of math expressions is that it is a slight difficulty in typing the expression in “reading order”. However, an editor may move the current insertion point (when editing an RTL math expression) to before (left of) the inserted character, instead of the usual after (right of) the inserted character. However, math expressions are *not* "running text" after all, so editing math expressions is a bit special anyway.

Annex D gives a list of **MMS**-mirrorable characters. Technically, MMS could be applied to a symbol that has a mirror character allocated, but it is preferable to just use the mirror character.

8 Character mappings and variants for symbols

Unicode has a number of mappings, and above we basically proposed a new mapping: mirror symbol (which is applied only one symbol at the time) as an edit operation.

But there are several other mappings, plus (for editors that handle styled text) also style mappings (e.g., change to italic, change font, change colour, etc.).

8.1 Case and style mappings

Case mapping can, in the case of document editing and display, be done as an edit operation, or be done for display (though the latter, to be strict, is non-conforming to Unicode, it violates "character identity").

However, there *shall* be ***no** case mapping of any kind* inside a math expression. This insulation from case mapping covers not only case mapping as an edit operation (on a selected part), but also style setting based or font-based case mapping.

Of course, a letter may be replaced by another letter, including its case map. But that is an substitution operation that is not directly related to case mapping, except by happenstance.

The reason is that the "same" letter but of different case are different "mathematical identifiers" (be that variables, constants, or unit designations, etc.). Also: the same letter (of the same case) but different style, often signify different "mathematical identifiers". That is why the math styling operators (as defined here) are *not* text styling in the ordinary sense.

This should apply also to **STX/<txt>** embedded texts, though it might not be so clear-cut then.

As mentioned above, identifiers in math expression have a math style (not related to the style of the text surrounding the math expression). A math editing program can map an identifier (one at the time) to another math style (replacing **SCI** or **<st>** for that identifier). But no style changes like that done for non-math text, though that can be applied to embedded **STX/<txt>** content.

8.2 Unicode normalisation mappings

Mapping to and between Unicode normal forms NFC and NFD may be performed also for math expressions (as defined in this proposal), except that (for the XML/HTML/SVG compatible version) '>' need be shielded from a combining solidus stroke if the '>' is part of a tag. The latter "should not happen" (it is inappropriate according to this specification, as combining characters are for math expressions only allowed in properly formed combining sequences).

Note, however, that mapping to NFKD or NFKC may turn an unpermitted math expression into a permitted one or do inappropriate mappings for "GREEK * SYMBOL" and "HANGUL LETTER *", and may split some "composed" symbols. (It also maps isolated form Arabic letter after an **SCI** to its nominal letter; but one may allow the nominal letter and not just the isolated form.) Recall that letters that have a compatibility decomposition are in general (with some exceptions) not allowed in identifiers in the math expressions as defined here. Applying NFKD or NFKC is thus not recommended, since it may have undesirable effects (even to the point of changing the semantics of the math expression).

The “context” of the math expression may have a mechanism for character references, like XML and HTML do. Also ECMA-48 has a mechanism for character references, original is just for C1 control characters (**ESC** …), but a proposed extension has extended that to character references also for characters with scalar values larger than 009F (**CSI** … **_** or **ESC [** … **_**; where … is the scalar value in decimal).

Whether the normalisation (NFD, NFC) affect also the characters given via character references or not depends on the point when the normalisation is applied, or if it is extended. For the HTML/SVG/XML compatible version, **&gt;** (or **&#x3E;**) followed by combining solidus stroke, may, *by extension*, NFC normalise to ≯ (**&ngt;** or **&#8815;**).

8.3 Mapping HT to SPs (and sometimes v.v.)

In several contexts it is commonplace to replace **HT** by a sequence of **SP** characters. That is not always a good idea, since **HT** often has a semantics beyond spaces, even if not moving to a tab stop. It is sometimes ok, or even a good idea, to replace **HT** by a sequence of **SP**s, but one needs to be cautions. In many situations, **HT**s need to be kept, and not substituted away.

For the XML/SVG/HTML compatible version, mapping **HT** to **SP**s is ok. **HT** and **SP** are always just “whitespace” and sequences of **HT** are collapsed in display to either “nothing” (or effectively WS, WORD SEPARATOR), or to a single **SP** (in **<txt>** blocks) in display.

For the C1 version of math expression representations, as defined here, **HT** is used to terminate cell content in math matrices. So in the context of a math matrix, **HT** *cannot* be replaced by **SP**s, as that would mess up the math matrix (as represented in the C1 version proposed here).

8.4 NLF mappings

It is common to map between certain *NLF*s, in particular between <**CR**,**LF**> and **LF**, but sometimes also **CR**. This is unproblematic for both variants of math expression representation proposed here. While mapping an NLF to SP (quite a common mapping), is unproblematic for the HTML/SVG/XML compatible variant, it cannot be done in matrices in the C1 version, since there each NLF is a row separator. Likewise for the reverse mapping, “breaking lines”.

VT is nowadays sometimes used as a C0 alternative to **LS**, though that is not **VT**:s original semantics. By letting **SCI LS** and **SCI VT** be equivalent (and that (**CR**)**VT** and **LS** are both *NLF*s), also a conversion, that is likely “blind” to **SCI**, between **LS** and **VT** (though maybe rare) can be done without any problems w.r.t. the math expression representation in the C1 variant proposed here.

8.5 Math symbol U+FE00 variants

Unicode’s *StandardizedVariants.html* specifies a number of standard variants for some math operators and other math symbols. Some are, however, ill-conceived and *shall not* be used. One of them is particularly ill-conceived and really needs to be removed from *StandardizedVariants.html*.

- U+2205 U+FE00; *EMPTY SET as zero with long diagonal stroke overlay*: this variant is ***EXTREMELY*** strongly deprecated by this proposal for math expressions, and this U+FE00 **SHALL** be ignored; the empty set symbol is **ALWAYS** based on a circle. For a slashed zero, use a 0 with U+0338 (COMBINING LONG SOLIDUS OVERLAY); but that should not be used for denoting the empty set, though it was a typewriter fallback. This variant selection has fortunately been removed from Unicode, but we still mention it here, in case you look at an earlier version of Unicode.
- U+2229 U+FE00; *INTERSECTION with serifs*: strongly deprecated for use in math expression and this FE00 should be ignored.
- U+222A U+FE00; *UNION with serifs*: strongly deprecated for use in math expression and this FE00 should be ignored.
- U+2293 U+FE00; *SQUARE CAP with serifs*: strongly deprecated for use in math expression and this FE00 should be ignored.
- U+2294 U+FE00; *SQUARE CUP with serifs*: strongly deprecated for use in math expression and this FE00 should be ignored.

- U+2278 U+FE00; *NEITHER LESS-THAN NOR GREATER-THAN with vertical stroke*: though no longer in Unicode, this variation sequence may well be interpreted for use in math expressions.
- U+2279 U+FE00; *NEITHER GREATER-THAN NOR LESS-THAN with vertical stroke*: though no longer in Unicode, this variation sequence may well be interpreted for use in math expressions.

- U+2296 ⊖ *CIRCLED MINUS* is here deprecated in favour of U+229D ⊝ CIRCLED DASH, which is actually a circled minus with a white rim, and in math expression both should display as U+229D ⊝ CIRCLED DASH.
- U+2295 U+FE00; ⊕ *CIRCLED PLUS with white rim*: this is the form that should be displayed in math expression also without U+FE00.
- U+2297 U+FE00; ⊗ *CIRCLED TIMES with white rim*: this is the form that should be displayed in math expression also without U+FE00.
- U+2298; ⊘ CIRCLED DIVISION SLASH; this too should have a white rim. This character is actually CIRCLED SOLIDUS, it is not related to DIVISION SLASH which we here regard as a control character.
- U+229C U+FE00; *CIRCLED EQUALS with equal sign touching the circle*: this is strongly deprecated in math expressions and this U+FE00 should be ignored.

- U+29C0; *CIRCLED LESS-THAN*: just noting that the glyph has a white rim.
- U+29C1; *CIRCLED GREATER-THAN*: just noting that the glyph has a white rim.
- The following variants are preferred, in math expressions, also without U+FE00:

  - 22DA FE00; with slanted equal; # LESS-THAN EQUAL TO OR GREATER-THAN
  - 22DB FE00; with slanted equal; # GREATER-THAN EQUAL TO OR LESS-THAN
  - 2AAC FE00; with slanted equal; # SMALLER THAN OR EQUAL TO
  - 2AAD FE00; with slanted equal; # LARGER THAN OR EQUAL TO
  - 2A9D FE00; with similar following the slant of the upper leg; # SIMILAR OR LESS-THAN
  - 2A9E FE00; with similar following the slant of the upper leg; # SIMILAR OR GREATER-THAN

8.6 Emoji in math expressions

Some character that can reasonably be used in math expressions (some arrows in particular) have emoji forms. However, there are no emoji in math expressions, so all “potential emoji” characters are *always* in text form (regardless of modifier (U+FE0E, U+FE0F), if any) in a math expression. Emoji that do not have a text form must not occur in a math expression.

8.7 Input mappings (or shortcuts)

There are some extremely common math symbols that are not in ASCII, and hence uncommon on most keyboards. Symbols such as MINUS SIGN, ALMOST EQUAL TO, symbols for less/greater than or equal to, plus-minus sign. Programming languages tend to use HYPHEN-MINUS as fallback for MINUS SIGN, <= as fallback for less-than-or-equal sign. We do not want such fallbacks in traditional math expressions, but still allow for easy entry of such very common math symbols. There are also some other “fallbacks” that are in ASCII but they are just fallbacks. The proper characters can be input, either from a palette, or by character references. But that is not all that easy and should not be needed to do so for really common math symbols.

To make things a bit easier, a small number of ASCII characters and a few other characters are mapped within math expressions as defined in this proposal. These are intended for making manual input easier, while also avoiding some mistakes/fallbacks. Nominally, these mappings are processed before the parsing of the math expression, as a preprocessing in a math expression. Conventionally, parsing is usually split into a lexing phase and a parsing phase taking the output of the lexing phase as input. The mappings below are to be done (nominally) during the lexing phase. (However, there is no requirement that there is a lexing phase per se.)

- HYPHEN-MINUS, -, is mapped to MINUS SIGN, −, in a math expression, except if it is in a **STX/<txt>** group (explicitly bracketed), where it is interpreted as NO-BREAK HYPHEN (HYPHEN is also interpreted as NO-BREAK HYPHEN); unless followed by + or --, see below.
- ASCII TILDE, ~, if directly followed by an **SP**, is mapped to TILDE OPERATOR; unless followed by ~.
- Two ASCII TILDE in a row, ~~, is mapped to ALMOST EQUAL TO (≈), which means ‘approximately equal to’ (the character named “APPROXIMATELY EQUAL TO” (≅) is an unusual alternative to the usual ‘approximately equal to’ (≈)).
- Three or more HYPHEN-MINUS in a row, ---, is mapped to **MHD** (MATH HORIZONTAL DIVISION) in the C1 version. Note that this maps to a control code, not to a symbol.
- Two SOLIDUS in a row, //, is mapped to DIVISION SLASH (which we regard as a control code here) in the C1 version.
- <= and >= are mapped to U+2A7D (⩽) and U+2A7E (⩾), respectively.

- ≤ and ≥ are mapped to U+2A7D (⩽) and U+2A7E (⩾), respectively.
- +- is mapped to U+00B1 (±).
- -+ is mapped to U+2213 (∓).
- =/ and != are mapped to U+2260 (≠).
- == is mapped to U+2261 (≡).
- ==> is mapped to ⇒. --> is mapped to U+1F852 (RIGHTWARDS SANS-SERIF ARROW)
- <== is mapped to ⇐. <-- is mapped to U+1F850 (LEFTWARDS SANS-SERIF ARROW)
- <=> is mapped to ⇔. <-> is mapped to U+1F858 (LEFT RIGHT SANS-SERIF ARROW)
- In many (non-math) fonts, "simple single line body" arrows have unfortunately assumed a "dingbat" look (e.g., →). While "clear" it is not appropriate for a math context. Fortunately, a proper looking (for math) set of "simple single line body" arrows are already encoded. To avoid the dingbat look (esp. if copied from a math expression to outside of a math expression) we map some arrow characters to their corresponding non-dingbat looking (and sans-serif) counterparts:
    - From 2190;LEFTWARDS ARROW to 1F850;LEFTWARDS SANS-SERIF ARROW
    - From 2191;UPWARDS ARROW to 1F851;UPWARDS SANS-SERIF ARROW
    - From 2192;RIGHTWARDS ARROW to 1F852;RIGHTWARDS SANS-SERIF ARROW
    - From 2193;DOWNWARDS ARROW to 1F853;DOWNWARDS SANS-SERIF ARROW
    - From 2194;LEFT RIGHT ARROW to 1F858;LEFT RIGHT SANS-SERIF ARROW
    - From 2195;UP DOWN ARROW to 1F859;UP DOWN SANS-SERIF ARROW
    - From 2196;NORTH WEST ARROW to 1F854;NORTH WEST SANS-SERIF ARROW
    - From 2197;NORTH EAST ARROW to 1F855;NORTH EAST SANS-SERIF ARROW
    - From 2198;SOUTH EAST ARROW to 1F856;SOUTH EAST SANS-SERIF ARROW
    - From 2199;SOUTH WEST ARROW to 1F857;SOUTH WEST SANS-SERIF ARROW

  Other non-dingbat arrows hopefully will not be commonly afflicted by a dingbat look.
- NUMBER SIGN (#) is mapped to octothorpe, which is VIEWDATA SQUARE, ⌗. See mathworld.wolfram.com/Primorial.html and mathworld.wolfram.com/Octothorpe.html: ⌗{$n : n > 1$}

Using characters, or character sequences, that are mapped might not allow for combining characters to be applied to the result of the mapping.

While not part of the proposal here, "the environment" may provide for decimal or hexadecimal character references, like **&#x***mmm***;** in XML (but not reference what we here regard as control characters, like DIVISION SLASH), and **CSI***nnn***_** in extended ECMA-48 . (For the markdown version, Appendix A, we do propose "internal" character reference mechanisms: **¤***nnnn***¤** for a decimal character reference (*nnnn* all ASCII digits) as well as name character reference (*nnnn* all ASCII letters), and **§***mmm***§** for a hexadecimal character reference; the numeric ones must refer to Unicode scalar values at least A0 (hexadecimal).)

Notes on some other characters:

- Invisible operators, there are a few defined in Unicode, must *not* be used; indeed 'default-ignorable' code points must not be used.
- Space characters (except for SPACE itself), like MEDIUM MATHEMATICAL SPACE (MMSP), may be used to adjust the position of subexpressions, but using "phantom" expressions and matrix column spacing adjustments should be preferred. All spaces, except SPACE itself, are interpreted as "no-break" inside a math expression.
- Line/paragraph and page break characters (incl. **VT**) as well as **HT** are interpreted as SPACE in math expressions, *except* that line/paragraph and page break characters are row separators for matrix layout (inside the grouping brackets for matrix layout), and

various tabs are interpreted as cell terminators in a math matrix (in the C1 version of math expressions as defined here).

- CHARACTER TABULATION (JUSTIFIED) (**HT** and **HTJ**) characters are interpreted as cell terminators within a row in a matrix layout, but otherwise **HT** is interpreted as SPACE (which is basically ignored) (and **HTJ** is not expected as "whitespace").
- A sequence of SPACE characters (including characters that are reinterpreted as SPACE) is interpreted as a (NO-BREAK) ZERO WIDTH SPACE or WORD JOINER. Other spaces, however, are *not* interpreted as a SPACE, like NO-BREAK SPACE, MEDIUM MATHEMATICAL SPACE, FIGURE SPACE, …
- ZERO-WIDTH SPACE and SOFT HYPHEN are ignored in math expressions.
- Vulgar fractions: use DIVISION SLASH or **MHD** (with numeral arguments), or the corresponding XML/HTML compatible tags, instead.
- Superscript digits (and superscript minus sign) are not permitted. Instead use **SUPL** (**<lsp/>**) and **SUPR** (**<rsp/>**) with ordinary digits to make numerals superscript within a math expression. Similarly for subscript digits.
- MIDDLE DOT must not be used as decimal marker, as that may cause unnecessary confusion. DOT OPERATOR (should look like, and work like, MIDDLE DOT in a math expression) can be used for multiplication, esp. between numbers as space is commonly used for digit grouping. For middle dot inside a Catalan word that is used as math identifier, please use U+013F and U+0140 which are allowed inside math expressions despite having compatibility mappings and despite Unicode's DoNotEmit.txt (also: U+0140 usually looks better typograpically than using the decomposed form, as the latter requires kerning that is not always there…). Similarly, use the IJ ligatures, if the identifier is in Dutch (and contains ij).
- Sometimes, × is used for multiplication between numerals (but next to never when there is a variable). Sometimes ÷ is used for division between numerals (no special layout), but otherwiese / (plain ASCII /) is used for division without special layout (for division with special layout, there a five characters that are here regarded as control characters).

8.8 Security issues

Any mappings done automatically in the background may result in security issues. That includes such simple things as mapping between line ending conventions, or case conversions. However, focussing on math expressions, firstly we recommend against many mappings, and secondly it would be ill advised to use, or even allow math expressions (as defined here) in a command language or in parameter settings in a security sensitive context. However, mapping away (i.e., deleting) for instance C1 control codes may "expose" security sensitive commands or parameter settings "hidden" by the C1 controls (or for that matter HTML tags, which some processes do delete); further, removing just the control codes, but leaving the content, of control sequences and control strings may "expose" characters that were inside the controls. This is a general issue when deleting or replacing characters, and the proposals here should not be introducing any in principle new security issues (not even for the symbol mappings above).

In addition, math expressions should be limited in coding size. Otherwise, an (outermost, first) **SME** followed by whatever but no (matching) **EME** could essentially hide all the rest of the text. A limit of, say, 1000 characters should allow for all reasonable math expressions, while avoiding an unmatched **SME** to "gobble up" all the rest of the text (likely with lots of parsing errors to boot). The parsing of a math expression can also be terminated by any syntax error, even without a size limit, not enabling math expression reading (starting with interpreting an **SME**) for, say, another 100 characters, just as a "safety margin". These are intended as error catchers, not hard-and-fast limits.

Furhtermore, an embedded text (STX…ETX, <txt>…</txt>), or embedded comment, may contain hyperlinks, which in turn may pose a security risk. Appropriate mitigation measures should be taken in case of hyperlinks, also when/if embedded in a math expression.

9 Grammar for math layout expressions

As mentioned before, the proposals here only cover styling and layout, not (mathematical) semantics. The proposals do cover "layout semantics", but only that. We have described the syntax for the math expression representations a bit informally so far. Here we will be a bit more formal and give a grammar (actually two) for the math expression representations. For the C1 (and C0) control codes variant, we will use an ABNF, and for the XML/HTML/SVG compatible variant we will give a (partial) DTD. A DTD cannot express many of the details, so we will then refer to the ABNF for these syntax details. That is also needed for the structure "reconstruction" (i.e., parsing) at an extra level of parsing after the XML/HTML/SVG parsing. Requiring such an extra level of parsing is vital for enabling a succinct expression in the XML/HTML/SVG compatible variant. Explicitly having XML/HTML/SVG bracketing (i.e. start and end tags) for everything would create the same (or worse) verbosity that MathML and OMML suffers from. Using infix "layout operator tags" instead is essential for avoiding this verbosity. This is normal in just about any programming language as well as non-XML math formats, but unorthodox for an XML compatible format.

The grammars do *not* cover the (pre-processing) symbol mappings discussed in section 8.

9.1 Character classifications

The characters defined in Unicode have properties to classify them, in particular "general category", "script", and "bidi category". We will here use these properties for indicating which characters are allowed where, without enumerating a large number of characters in the ABNF.

9.1.1 NLFs

We begin with NLF, "New line function", characters and character pairs. This applies to all implementations, regardless of platform.

NLF ::= **FF** | **CR FF** | **VT** | **CR VT** | **LF** | **CR LF** | **CR** | **NEL** | **PS** | **LS**

Note that the two-character compositions here stands for one NLF each, and must not be interpreted as two NLFs. While not, in principle, difficult, it is still quite common with NLFs disappearing or extra appearing, especially in copy-paste operations. This kind of bugs may cause problems for the C1 version and math matrices, while "just" cosmetic for the XML/HTML/SVG compatible version.

9.1.2 Numerals

Numbers, and thereby numerals, of course play an important part in most branches of mathematics. Here we will allow for decimal position-based numerals with certain (commonly used) punctuation. Using other bases than 10 (but still position-based) does happen, especially in computer science. If the base is larger than 10, then one commonly uses certain letters for larger-than-9 digits. However, we will not cover this here, except to say that numerals in bases greater than 10 will be “relegated” to be expressed using **STX**/**<txt>**.

As *decimal-digit*s we allow for all characters that have general category Nd *and* do not have a compatibility decomposition. That excludes the so-called “MATHEMATICAL” digits. We have other ways of “styling” decimal numerals in a math context. It also excludes superscript and subscript digits; and we have other means of making numerals superscript or subscript in a math expression. Within the numeral, the digits *shall* be of the same script, not a mixture; though this is not given in the grammar here, it needs to be checked by other means, though a script change *does* break up the numeral into several numerals. As usual, longest match rule applies.

```
num ::= (decimal-digit)+((“,”|“.”|PSP)(decimal-digit)+)*
            ((“,”|“.”)(decimal-digit)*(“_”|“:”)(decimal-digit)+|“…”)?
   // The digits must all be of the same script. Bidi only applies for numerals
   // in the N’Ko script (if supported), and then displays the numeral RTL in its entirety.
   // A (single!) SPACE between digits is mapped to PSP (PUNCTUATION SPACE, U+2008).
```

The second line here covers decimal repeats: “_” for overlined (**CSI 53m** in ECMA-48, though that control sequence is not permitted directly here); “:” for overdotted, i.e. a dot above each of the digits in the repeat section (**CSI 64:1=m**). Using parenthesis for decimal repeat is not covered here, since it is ambiguous in reading, and it can still be expressed using *num-sym-num-sym* or *num-sym-sym-num-sym*; for the rare case of RTL numerals, the reverse of that; note: no display time mirroring of the parentheses. The semantic level will need to put the pieces together when using parentheses for this.

Importantly, note that the *author* selects the displayed representation here; “.” is *always* displayed as “.” and “,” is *always* displayed as “,” in a numeral (and elsewhere); this is also important in math matrices for decimal lineup in columns, where the matrix start control tells whether “,” or “.” is the decimal marker. The decimals repeat marking is also selected by the author. These are *never* dynamic display preferences (per reader’s preference settings or other context); changes between the representations are edit changes to the source representation.

This syntax cannot handle numerals in bases greater than ten. A numeral in hexadecimal, base 16, can be given as **STX** <*hexadecimal numeral*>**ETX** (base is indicated by whatever method the author prefers, including no explicit mark as well as right side subscript in decimal). Exceptionally, math styling *could* be made available for numerals given as embedded text (using an **SCI**<*letter*> prefix), but that is *not* covered in this proposal (yet), and should only be allowed for such numerals, not other embedded texts.

9.1.3 Identifiers

Mathematical identifiers are often single-letter. But there are many exceptions where they are multi-letter. Like “sin”, “lim”, “sup”, … This is even more common in some uses, like in computing, where multi-letter variable names are quite common, e.g., “*coefficient*”. In contrast to identifiers in most programming languages, they do not contain digits however, though it is common to have digit indexed variables, like “x_2”.

For *id* and *sym*, we will first do a special character category reclassification, just for math expressions:

1. The following characters we here regard as *control characters*, just like the C1 controls **SME**, **EME**, **MHD**, **MMS**:

2044;FRACTION SLASH	(will not be used in math expressions as defined here)
215F;FRACTION NUMERATOR ONE	(will not be used in math expressions as defined here)
2215;DIVISION SLASH	
27CC;LONG DIVISION	(will not be used in math expressions as defined here)
29F5;REVERSE SOLIDUS OPERATOR	
29F8;BIG SOLIDUS	
29F9;BIG REVERSE SOLIDUS	

   In the C1 variant of math expressions as specified here, we allow only control characters that we give a well-defined math layout semantics. In the XML/HTML compatible variant we do not allow any control characters of any kind (except for some *NLF* and **HT**, which are treated as “white space”). I.e., no bidi controls or other “format controls”, nor C0 (except those treated as “white space”) nor C1 controls, nor the above “newly regarded as controls” characters in math expressions.

2. We will also consider SCRIPT CAPITAL P (Weierstraß p, it is actually lowercase, not capital) not as a symbol (it has general category Sm), but a letter with a (should-have-had) compatibility decomposition. This means that it will not be permitted to be used (use instead **SCI W**p or **<st s='W'>**p**</st>**). For the same reason, “MATHEMATICAL” letters and digits are not permitted.

As *letter*s we regard characters with general category Lu, Ll, Lo (not Lt, Lm, Sk, Nl, No; for Hangul Jamo, they need to be properly composed: (*choseong*+ *jungseong*+ *jongseong**), while actually orthographic syllables, we call them letters here). Like for numerals, we do not allow letters that have compatibility decompositions *except* for “GREEK <letter> SYMBOL” and “HANGUL LETTER *” which are allowed regardless of their compatibility decomposition (and must not be mapped):

0132;LATIN CAPITAL LIGATURE IJ
0133;LATIN SMALL LIGATURE IJ
013F;LATIN CAPITAL LETTER L WITH MIDDLE DOT
0140;LATIN SMALL LETTER L WITH MIDDLE DOT
037B; GREEK SMALL REVERSED LUNATE SIGMA SYMBOL
037C; GREEK SMALL DOTTED LUNATE SIGMA SYMBOL
037D; GREEK SMALL REVERSED DOTTED LUNATE SIGMA SYMBOL
03F0; GREEK KAPPA SYMBOL
03F1; GREEK RHO SYMBOL
03F2; GREEK LUNATE SIGMA SYMBOL
03F4; GREEK CAPITAL THETA SYMBOL
03F5; GREEK LUNATE EPSILON SYMBOL
03F6; GREEK REVERSED LUNATE EPSILON SYMBOL
03F9; GREEK CAPITAL LUNATE SIGMA SYMBOL
03FD; GREEK CAPITAL REVERSED LUNATE SIGMA SYMBOL
03FE; GREEK CAPITAL DOTTED LUNATE SIGMA SYMBOL
03FF; GREEK CAPITAL REVERSED DOTTED LUNATE SIGMA SYMBOL
3131..3163,3165..318E HANGUL LETTER *

In particular, that excludes the so-called “MATHEMATICAL” letters. We have another way of “styling” identifiers in a math context. We also exclude U+2135 to U+2138; use the ordinary Hebrew letters instead (note that bidi is applied to each identifier *in isolation*). Allowing some characters with compatibility decomposition also means that NFKD/NFKC *must not* be applied to math expressions; the exclution of other characters with compatibity decomposition is strict, it cannot be curcumvented by applying NFKD or NFKC. As noted elsewhere too, case mapping *must not* be applied to math expressions either.

There *should* be no mixing of scripts in a single identifier. Though it is common to mix Latin letters with Greek letters in math expressions, and even Hebrew letters, it is not done in a single identifier, except that some SI units do mix scripts (µg, for instance). The letters in an identifier may have combining marks attached to them. As usual, when matching regular expressions with repetitions, longest match (for the total regular expression) rule applies.

```
id ::= (letter(combining-mark)*)+((hyphen|middledot)(letter(combining-mark)*)+)*
    // The letters and combining marks (if script specific) should all be of the same script.
    // Note that hyphen cannot be given as HYPHEN-MINUS, since that is mapped to MINUS SIGN.
    // The combining marks are either not specific to any script, or specific to the same script
    // as its letter base.
```

9.1.4 Symbols

As symbols we regard all characters that have the general category P* (any punctuation) *except script-specific punctuation*, S* (any symbol, but excepting SCRIPT CAPITAL P) *except script-specific symbols*. As letter-like symbols, we consider all integral signs, n-ary summation, n-ary product and coproduct, as well as partial differential symbols.

Non-script-specific Po, Ps, Pe, Sc, Sm, non-script-specific So, and the letter-like symbols listed below. Note that here, Sc:s are not regarded as letter-like in that they have fixed upright style, normal weight.

The following we regard as permitted *letter-like symbols*, regardless of their compatibility decomposition:

2200;FOR ALL
2201;COMPLEMENT
2202;PARTIAL DIFFERENTIAL
2203;THERE EXISTS
2204;THERE DOES NOT EXIST
2206;INCREMENT
2207;NABLA
220F;N-ARY PRODUCT
2210;N-ARY COPRODUCT
2211;N-ARY SUMMATION
222B;INTEGRAL
222C;DOUBLE INTEGRAL
222D;TRIPLE INTEGRAL
222E;CONTOUR INTEGRAL
222F;SURFACE INTEGRAL
2230;VOLUME INTEGRAL
2231;CLOCKWISE INTEGRAL (the mirrored sign, via **MMS**, must still indicate clockwise, i.e. the ring arrow part is *not* mirrored)
2232;CLOCKWISE CONTOUR INTEGRAL (the mirrored sign, via **MMS**, must still indicate clockwise)
2233;ANTICLOCKWISE CONTOUR INTEGRAL (the mirrored sign, via **MMS**, must still indicate anticlockwise)
2320; TOP HALF INTEGRAL (these five chars could presumably be used by an (vertical) stretching mechanism, but not be used by "themselves")
2321; BOTTOM HALF INTEGRAL
23AE; INTEGRAL EXTENSION
23B2; SUMMATION TOP
23B3; SUMMATION BOTTOM
2A0B; SUMMATION WITH INTEGRAL
2A0C; QUADRUPLE INTEGRAL OPERATOR
2A0D; FINITE PART INTEGRAL
2A0E; INTEGRAL WITH DOUBLE STROKE
2A0F; INTEGRAL AVERAGE WITH SLASH
2A10; CIRCULATION FUNCTION (not clear: should the circle arrow part be mirrored by **MMS**?)
2A11; ANTICLOCKWISE INTEGRATION (the mirrored sign, via **MMS**, must still indicate anticlockwise)
2A12; LINE INTEGRATION WITH RECTANGULAR PATH AROUND POLE
2A13; LINE INTEGRATION WITH SEMICIRCULAR PATH AROUND POLE
2A14; LINE INTEGRATION NOT INCLUDING THE POLE

2A15; INTEGRAL AROUND A POINT OPERATOR
2A16; QUATERNION INTEGRAL OPERATOR
2A17;INTEGRAL WITH LEFTWARDS ARROW WITH HOOK (not clear: should the arrow with hook part be mirrored by **MMS**?)
2A18;INTEGRAL WITH TIMES SIGN
2A19;INTEGRAL WITH INTERSECTION
2A1A;INTEGRAL WITH UNION
2A1B;INTEGRAL WITH OVERBAR
2A1C;INTEGRAL WITH UNDERBAR

The following we regard as permitted *symbols*, regardless of their general category and compatibility decomposition:
00A8;DIAERESIS (prefer two FULL STOP)
00AF;MACRON (prefer LOW LINE)
00B4;ACUTE ACCENT
02D8;BREVE (can reasonably stretch a bit over a math expression)
02DA;RING ABOVE
02DD;DOUBLE ACUTE ACCENT
2017;DOUBLE LOW LINE (like LOW LINE, this can reasonably stretch a lot over/under a math expression)
2024;ONE DOT LEADER (prefer FULL STOP)
2025;TWO DOT LEADER (prefer two FULL STOP)
2026;HORIZONTAL ELLIPSIS (prefer three FULL STOP)
203E;OVERLINE (prefer LOW LINE)

The following symbols are *not* permitted, use instead the compositions given (C1 variant, spaces here are for readability only):

0606;ARABIC-INDIC CUBE ROOT	**MMS** √ **SUPR** ٣
0607;ARABIC-INDIC FOURTH ROOT	**MMS** √ **SUPR** ٤
221B;CUBE ROOT	3 **SUPL** √
221C; FOURTH ROOT	4 **SUPL** √
225D;EQUAL TO BY DEFINITION	= **ABV SCI A** def
225F;QUESTIONED EQUAL TO	= **ABV** ?
225E;MEASURED BY	= **ABV SCI A** m

```
 sym ::= symbol(combining-mark)*
    // Includes letter-like symbols (incl. U+2135-U+2138, integrals, sum, …), punctuation, arrows.
    // The symbols need not be of general category Sm, but any non-script-specific S*, P*.
    // We also allow some mappings of a symbol sequence to a symbol; see section 8.7;
    // those (pre-processing) mappings are not included in the grammar here.
    // (Note: some of those mappings map to control characters, not symbols.)
    // The combining marks must not be script specific.
```

Symbols, and in **STX**/**<txt>** blocks *symbol sequences*, in math expression are always untouched by the bidi algorithm. No reordering, no mirroring. And indeed, no style change either, they always have style **SCI E** (except for letter-like symbols and currency symbols) wherever symbols occur in a math expression (as defined here). Since not all left-right-asymmetric non-script-specific symbols have a mirror character allocated, there is a mechanism for explicitly mirroring a single (non-script-specific) symbol. That is interpreted regardless of bidi, but should not be used for symbols that do have a mirror character allocated.

9.2 Grammar for the C1 based version

Here we give an ABNF for the C1 variant of the proposed format. In Annex B an LALR grammar, as a *bison* source, is given (assuming some "lexical" parts are handled by a laxer which is not given). *me* is a math expression (actually a math expression component, it *need not* form any logical *expression*), *stack* and *stretch* is defined a bit further down here.

```
me ::= SME(PHA|VPHA)?RECT?(SP|HT|NLF)*((stretch|pragma)(SP|HT|NLF)*)*EME
            // Outside of the outer SME…EME, all math controls except SME are ignored.
            // If a SME has a PHA or VPHA, the PHA is inherited to all inner parts of the SME…EME;
            // there is no “unphantom” control; selection of the “vertical” part is done at the
            // outermost VPHA-marked SME…EME, and that implies RECT. Sequences
            // of no-break spaces are automatically embraced by STX…ETX. The syntax
            // requirements from the parsing exclude any other controls, like bidi controls.

st ::= (SCI [A-Za-z$<certain (isolated) Arabic letters>])?(id|num|(SCI +|SCI -)?MMS?sym)
            // The styling is the math default style as per above if there is no explicit style indicator.
            // The style indicator may be an Arabic letter after the SCI, but that applies only to
            // id/num in the Arabic script. MMS should be used only for RTL math expressions and
            // only when there is no corresponding mirror character encoded. Symbols that
            // are not letter-like cannot be style changed (even if one tries); on the other hand,
            // (SCI +|SCI -)? apply only to symbols that are not letter-like nor are default fences.

txt ::= STX (plaintext|me)* ETX
            // plaintext inherits surrounding style (outside of ME) but may be explicitly styled if the
            // context allows, except that substrings of symbols are still upright, normal weight,
            // sans-serif. plaintext cannot contain STX, ETX, SME, other math controls are ignored.
            // Non-empty longest sequences of non-script-specific P*, S*, Z*, (HT read as SP) but
            // except if such a sequence contain only SP:s, trimming SP:s at both ends, are
            // automatically embraced by SME…EME. No control characters, except NLFs and HT:s
            // (both interpreted as SP, and SP sequences are collapsed to a single NBSP) and CSI
            // styling control sequences, CSI …m, are allowed in the plaintext, iff math expressions
            // are embedded in an ECMA-48 context. Any style change is reset (‘popped’) when
            // reaching the ETX.

pragma ::= SCI STX (plaintext|me)* SCI ETX
            // a pragma is not directly rendered, but may (implementation defined)
            // contain directives that may affect some portion of the rendering.

closed ::= me | st | txt | mx    // mx is defined below here

stack ::= closed((BLW|BLWS|BLWT|BLWU)closed)*((ABV|ABVS|ABVT|ABVU)closed)*
    // above- and below-scripts

around ::= (closed SUPL)?(closed SUBL)?stack(SUBR closed)?(SUPR closed)?
    // left and right subscripts and superscripts (using * instead of ? here does not make sense);
    // note that the superscripts and subscripts are placed relative the “inner” stack

frac ::= around(MHD|U+2215|U+29F5|U+29F8|U+229F9)(SCI [01])?around
    // “division-like” (except for plain / or \ which are just handled as symbols)

stretch ::= (stack(LSS|LSST|LSSU|LSSV))*(around|frac)((RSS|RSST|RSSU|RSSV)stack)*
        // the “division operator” controls (MHD, …) bind harder than the stretch controls;
        // the LSS operators associate to the right, the RSS operators associate to the left

mc ::= ((SCI &|SCI '|SCI ")*|(BS)*)?SP*(stretch SP*)*(HT|HTJ|SCI HT|SCI VT|SCI LS)
```

// ~~an **HT** cell terminator may be omitted for the rightmost cell in a matrix row~~
mr ::= *mc** // the ((**SCI &|SCI '|SCI "**)*|(**BS**)*)? parts are interpreted only for the top row
mx ::= **SCI(,|.|;|:)(SCI [0-9])?***mr* (**NLF(PLD*|PLU*)***mr*)***SCI #** // more "powerful" than **SME…EME**
// in case of nesting imbalance (a syntax error), for syntax error recovery;
// iff there are equally many cells in each row, there are columns in the matrix;
// **SCI 0** refers to the bottom row, and then upwards for **SCI** [**1**-**9**]

9.3 DTD (grammar) for the XML/HTML/SVG compatible version for math expressions and their integration

Using XML, or something compatible with HTML, easily ends up with something that has quite extreme verbosity, with lots of tags (begin and end) and attributes. Look for instance at MathML and OMML (Office Math Markup Language). They are both quite verbose.

Still, most of the tags and even attributes are not really needed. With the right design much of that can be avoided. We will try cut down on the verbosity by:

a) Use short tag names and short attribute names.

b) For the "st" (style) tag, use default styles as described above for the C1 control characters variant of this specification, thus avoiding a large fraction (almost all) of the explicit **<st>** tags, leaving just the (variable, function, set, …) names. Spaces are needed for "breaking up" letter sequences into individual names for variables that are multiplied, function application, etc. Using these defaults heavily cuts down on the verbosity, even though the tag and attribute names are short. At the XML parsing level, it "knows" nothing about this kind of defaulting, however. A second level of (math representation specific) parsing is needed.

c) Several tags (without attributes) are not needed (given the design of the math expressions as given below) to enable proper parsing of math expressions (in the format described here) but are there only to express the "grammar" for the math expressions in this format. For these we will use ENTITY (which does *not* define a tag) rather than ELEMENT (which does define a tag) in the DTD. Omitting these needless tags heavily cuts down on the verbosity of this version of the math expression format as specified here. Again, we need the second level parsing to be able to "recreate" the intended math layout representation structure. But this is normal (domain specific) parsing, commonplace in computing, even though it has not been popular to combine with XML.

Due to b) and c) the DOM for a math expression needs to be "parsed" just as for the C1 version of the math expression but adapted to have the DOM of the math expression as input. This second level of parsing, the first level being the parsing into a DOM representation, is the price to pay for having cut down (heavily) on the verbosity of the tagging. But without cutting down on the verbosity, the HTML/XML/SVG compatible version would be so verbose as to be impractical, it would be page upon page of tags even for quite small math expressions. Using ELEMENT instead of ENTITY in the DTD below would have the (small) advantage of not needed a second level of parsing, but that has the great disadvantage of drowning the actual math expression in lots of needless tags. Note that the DTD is made so that it can still be parsed unambiguously, not losing any information by using ENTITY instead of ELEMENT at *select places* in the DTD below. The resulting markup is quite similar to C1 version, except for math expression matrices, which are similar to HTML tables, whereas the C1 (and C0) version uses **HT** (and similar) and ***NLF*** to create the matrix.

Note that the parsing specified here does not handle parentheses (fences) matching, nor visible operator precedences and associativities, nor distfix parsing (like { | } or { : } for sets). If such parsing is needed (like for expression manipulation), then that has to be done at a further additional level of parsing, which is out of scope for this proposal. This additional level can vary between different areas of mathematics and between different traditions even in the same area.

9.3.1 The <me> element

The <me> element is the math expression element:

```
<!ELEMENT me (cmt|%stretch;)*>      <!ATTLIST me
                                              d (Y|PHANTOM|VPHANTOM) "Y"
                                              t (T|RECT) "T"
          -- Size, colours and shadows  are inherited; other styling is not. CSS cannot be applied to me.
          d="PHANTOM" is inherited to all inner elements, regardless of d attribute value for the
          inner elements. Sequences of no-break spaces are automatically embraced by <txt>…</txt>.
          The syntax requirements from the extra parsing level exclude any controls, like bidi controls. -->
```

9.3.2 Integrating the <me> (math expression top level) element

The math expressions should be fully integrated with the surrounding document content mechanisms, not just a “namespace import” as done for MathML. For the XML/HTML/SVG compatible version of math expression representation defined here we here illustrate the desired integration with example DTD snippets.

For XHTML (outdated, but it does have a formal DTD), one would integrate the **<me>** element as part of **%Inline**, like this:

```
<!ENTITY % Inline "(#PCDATA | %inline; | %misc.inline; | me)*">
```

For HTML 5 there is no formal DTD. However, Jukka Korpela has written an informal DTD for HTML 5, https://jkorpela.fi/html5.dtd. So for HTML 5 (using the informal DTD to illustrate), one would integrate the <me> element as part of **%phrase**, like this:

```
<!ENTITY % phrase "a | area | link | meta | style | abbr | address | audio |
  b | bdi | bdo | br | button | canvas | cite | code | command |
  data | datalist | del | em | embed | i | iframe | img | input |
  ins | kbd | keygen | label | map | mark | menu | meter |
  noscript | object | output | progress | q | ruby | s | samp |
  script | section | select | small | span | strong | sub | sup |
  svg | textarea | time | u | var | video | wbr | (#PCDATA) | me">
```

For SVG, an integration would allow **<me>** as a “text content child element”, where a math expression would be treated as a single “character” when laying out the text (so a math expression would “rotate” (as a single text element) along a path, but *not* bend along a path).

9.3.3 The <txt> element and its integration

For our math embedded text tag, **<txt>**, one would have the XHTML ELEMENT declaration:

```
  <!ELEMENT txt (%Inline;)*>   <!ATTLIST txt %attrs;      --  implicit <span>  -->
```

This allows for math expressions to contain such things as style changes, colour changes, boxing and more, including form text fields and submit buttons, and of course **<me>** (since we integrated that into **%Inline**). And one may use CSS inside **<txt>** (including **<txt>** itself), except for **<me>**. However, there is no line breaking for the content of **<txt>.**

For HTML 5, **<txt>** would have the HTML 5 ELEMENT declaration (informal, since HTML 5 does not have a DTD):

```
  <!ELEMENT txt (%phrase;)*>   <!ATTLIST txt %global;   --  implicit <span>  -->
```

Both of these, with the integration as given above, allow for **<me>** within **<txt>**.

For SVG, an integration would make math **<txt>** elements similar to SVG **<text>** elements, except that there are no position attributes and no **<textPath>** elements allowed in **<txt>**, and no line breaking of **<txt>** content. It would probably be advisable to not allow use of **<tspan>**, instead using a more HTML-like **<span>** element within **<txt>** elements.

However, bidi controls, in HTML as attributes and CSS styles, shall not be effective in **<txt>** elements. Only "plain bidi", i.e., *not* modified by controls, and in isolation, is allowed, and the "paragraph direction" is always LTR in **<txt>**.

Similarly for pragmas (XHTML, HTML5 (informal), similar for SVG but not given here):

```
  <!ELEMENT cmt (%Inline;)*>
  <!ELEMENT cmt (%phrase;)*>
```

9.3.4 Style for <txt>

The (initial) style for a **<txt>** element is inherited from outside of the nearest **<me>** that either a) is outermost or b) is directly in a **<txt>**, that brackets the **<txt>** element. CSS can be applied to **<txt>** or any (HTML) element that is in the **<txt>** (but as mentioned, no bidi control CSS).

Non-empty longest sequences of P*, S*, Z* but except if such a sequence contain *only* Z*, are automatically embraced by **<me>**…**</me>**, trimming Z* at both ends for the embrace. Symbols, including punctuation, are thus still in (default) *math style* within a **<txt>**.

Even though the %Inline/%phrase are not "math extra-parsed", no control characters of any kind are permitted; thus, no bidi controls nor characters we here regard as controls (e.g. DIVISION SLASH).

While <cmt> is similar to <txt>, a <cmt> is not rendered, and thus have no point in styling.

9.3.5 Math styling and math layout "elements" and "entities"

The following "elements" and "entities" are used inside of <me>, and need an extra level of parsing (except for the math matrix structure) to be reconstructed properly:

```
<!ELEMENT st (#PCDATA)>      <!ATTLIST st
                                  s NMTOKEN #IMPLIED
                                  b (n|y|h) "h"
                                  m (n|MIRROR) "n"
            -- The #PCDATA must be a num (&puncsp; instead of SCI SP), id or sym as per above, nothing
             beyond that. CSS cannot be  applied to <st>. The s attribute value must be a letter as described
             for the C1 variant for styles. m="MIRROR" only applies to P* & S* symbols and used in RTL math
             expressions and only for symbols (including letter-like symbols) that do not have a mirror char
             encoded. Symbols that are not letter-like have only one effective style: s="E", ignoring given s. -->

<!ENTITY % st "(#PCDATA | st)"
              -- Note that this declares an entity %st; which is different from the element <st>. The #PCDATA
              must be a num (&puncsp; instead of SCI SP), id or sym as per above, nothing beyond that.
              It is default math styled as per above, CSS cannot affect the style. However, to properly
              reconstruct syntactic structure, an extra layer of parsing of the PCDATA needs to be applied -->
```

```
<!ENTITY % closed "(me | %st; | txt | mx)"
        -- One should be cautious when using txt and applying, e.g., superscripts, for readability; but still
        allowed. The same caution actually applies also to me, since one may need parentheses to make
        it readable: recall that <me> and </me> never generate visible fences, visible fences always need
        to be explicit. -->


<!ELEMENT (blw | abv) EMPTY>      <!ATTLIST (blw | abv)      r (n|s|t|u) "n"
        -- n: no stretch, s: horizontally stretch to fit (100%), t: stretch to 105%, u: stretch to 110%.
        Note that these are just “layout operators”, there are no math subexpressions here. -->

<!ENTITY % stack "(%closed;, (blw, %closed;)*, (abv, %closed;)*)" -- note that the stacks may be empty -->

<!ELEMENT (lsb | lsp | rsb | rsp) EMPTY  -- Note that these are just the layout "operators", no content.  -->


<!ENTITY % around "((%closed;, lsp)?, (%closed;, lsb)?, %stack;, (rsb, %closed;)?, (rsp, %closed;)?)"
        -- Note that the superscripts and subscripts may be omitted. Like for all other “ENTITY”s here,
        to be able to reconstruct the syntactic structure, an extra level of parsing is needed.  -->


<!ELEMENT dv EMPTY>      <!ATTLIST dv   t (t-2|t-1|t0|t1|t2) "t0"   v (c|c0|c1) "c"
       -- t0: horizontal line division, t1: slanted line division, t2: big level division (with slanted line) -->

<!ENTITY % frac "(%around;, dv, %around;)">

<!ELEMENT (lss | rss) EMPTY>      <!ATTLIST (lss | rss)     s (s|t|u|v) "s"
       -- s: vertically stretch to fit (100%), t: stretch to 105%, u: stretch to 110%, v: stretch to 115% -->

<!ENTITY % stretch "(%stack;, lss)*, (%around; | %frac;), (rss, %stack;)*"
       --  the <dv> tag “operator” binds harder than the <lss> and <rss> tag “operators”;
         <lss> associates to the right, <rss> associates to the left -->


<!ELEMENT mc (%stretch;)*>     <!ATTLIST mc      h (l|r|c|n) "l"       c CDATA "0"
        -- l: left align, r: right align, c: centre, n: numeric align; the a attribute only holds for the top row;
        the CDATA must be an integer numeral indicating the column spacing adjustment in 1/6 em;
        the a attribute is effective only when occurring on the top row and there are columns in the
        matrix. CSS cannot be applied to <mc>.    -->

<!ELEMENT mr (mc)+>      <!ATTLIST mr      r CDATA "0"
        -- a: the CDATA must be an integer numeral (ASCII digits, prefix HYPHEN-MINUS for negative),
        indicating the row spacing adjustment in 1/6 em. CSS cannot be applied to <mr>.  -->

<!ELEMENT mx (mr)+>      <!ATTLIST mx     a (a|p|aa|pp) "a"      v (c|c0|c1|c2|c3|c4|c5|c6|c7|c8|c9) "c"
         -- a: align on comma for numeric alignment, p: align on full stop for numeric alignment;
         likewise for aa (comma) and pp (full stop), but have a 1.5 em default column separation;
         c: use default math centre, c0: use bottom row centre as vertical math centre for the matrix, …;
         CSS cannot be applied to <mx>.   -->
```

The top-level element here is <me>. All other elements and (DTD) entities defined above must be, in use, within an outermost <me>…</me>. Other elements defined above must not be used outside of an (outer) <me>…</me>.

The <mr> and <mc> could have been done “operator-like”, as we have done for several other tags (like for superscript, division, etc.). For now, at least, we let <mr> and <mc> be more in the tradition of SGML/XML/HTML/… (compare HTML tables). This makes the conversion to/from the

XML/HTML/SVG compatible representation and the other equivalent representations (C1, markdown) a bit harder, but the XML/HTML/SVG parsing a bit simpler (normal parsing, not special case operator parsing), but such "conventional" tagging is more verbose.

10 Copy/paste

While the behaviour of programs that help in the input and change of texts, including math expressions, is not in scope for this proposal, some remarks are still in order.

Many modern text editor programs allow for selecting a portion of text and do some operations on that portion. These operations often include copying or even "cutting" that portion of text, and then paste in that portion of text elsewhere (some programs allow for drag-and-drop). However, care must be taken that the results are not unexpected. For instance, pasting in a portion of a math expression somewhere outside of a math expression, should still appear as a math expression. I.e., one may need to add information that this is a math expression. If the editor works directly on the representation encoding, one may need to add **SME**…**EME** or **<me>**…**</me>** around the pasted in portion, or even balance up such control brackets (depending on how selection of a test substring is allowed to be).

A text editor may be using an internal structure that is different from the representation used in a file saved or read. This can be used to easier select a "reasonable" subexpression of a math expression, or, if starting the selection outside of a math expression, not to have the "end" in the middle of a math expression.

This is a subject of interaction design for the editor program, and there will be lots of other considerations and adaptations to how the editor program works in relation to math expressions and other things, like text styling. But enabling math expressions will imply various considerations related to math expression editing need to be made in order to make the editing of math expressions "natural" and without surprises.

11 Expression structure, higher level parsing

The grammars above only give the expression structure sufficient for layout of the math expression. It does not cover grouping by fences (parentheses, etc.), nor visible operator precedences or associativities.

To cover that, requires an additional layer of parsing. But that is out of scope for this proposal. However, it is needed for enabling more semantic oriented operations, including certain edit operation or equation manipulation, or even computerized theorem provers.

But even the simplest of this is out of scope for this proposal. The proposals here only cover the structure needed for layout/display of mathematical/logical expressions (including those that have physical unit expressions), as well as simple chemical (or nuclear) reaction formulas (more graph-like ones are not covered). Indeed, this proposal does not even cover matching open and close fences. The reason for the latter is that it is not uncommon to either have just the left side or the right side omitting the other (especially common for braces), and the existence of notations like "[*a*, *b*)" and "[*a*, *b*[" (both for intervals).

In addition, there are quite a lot of potential binary operators, used in different branches of mathematics and formal logic. Defining parsing precedences and associativities for all of them would be not only too much, but also inappropriate. That belongs outside of the systems proposed here. Indeed, what constitutes fences is not firmly set, and may vary or develop in ways we cannot foresee. However, if there are explicit **SME**…**EME** (or **<me>**…**</me>**) that conflict with *common* fences, a system should give a warning.

12 Comparison to other math markups

12.1 Comparison to LaTeX, some simple examples

First a few simple examples using the C1 version of the math controls. Note that the spaces are (mostly) here for readability.

LaTeX	Proposed control code representation; C0/C1 version
\(x^2 + y^2 = z^2\)	**SME** x **SUPR** 2 + y **SUPR** 2 = z **SUPR** 2 **EME**
\(E=mc^2\)	**SME** E=m **SP** c **SUPR** 2 **EME** *(note the space between m and c)*
\(\sqrt{x^2+1}\)	**SME** √ **LSS SME** x **SUPR** 2+1 **EME EME**

And then the same simple expressions using the XML/HTML/SVG compatible version.

LaTeX	Proposed control tag representation; XML/HTML/SVG compatible version
\(x^2 + y^2 = z^2\)	<me>x<rsp/>2+y<rsp/>2=z<rsp/>2</me>
\(E=mc^2\)	<me>E=m c<rsp/>2</me> *(note the space between m and c)*
\(\sqrt{x^2+1}\)	<me>√<lss/><me>x<rsp/>2+1</me></me>

And the markdown version given in appendix A.

LaTeX	Hinted markdown code representation (appendix A)
\(x^2 + y^2 = z^2\)	${x^2+y^2=z^2}
\(E=mc^2\)	${E=m c^2} *(note the space between m and c)*
\(\sqrt{x^2+1}\)	${√$({x^2+1}}

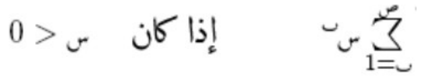
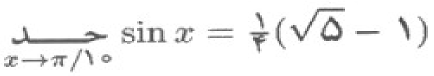
12.2 Some more examples

The LTR math examples here are taken from https://www.w3.org/TR/arabic-math/. We also copy (part of) some images from that reference. We have not copied the MathML; it is *way* too large; see the reference to see the MathML notation. Note that the **MIT** (math italic) for the integral sign, is there just to get an italic integral sign as (usual) in TeX. The default we use is to use an upright integral sign, which is normal notation.

	Proposed control code representation
$f(x)=\begin{cases}\sum_{i=1}^{s} x^i & \text{if } x<0\\ \int_1^s x^i\,dx & \text{if } x\in \mathsf{S}\\ \tan\pi & \text{otherwise (with } \pi\simeq 3.141)\end{cases}$	**SME** f(x)={**LSS SCI ;** ∑ **BLW SME** i=1 **EME ABV** s **LSSV** x **SUPR** i **HT SCI " STX** if **SP ETX** x<0 **HT LF** **SME MIT** ∫ **SUBR** 1 **SUPR** s **EME LSSV** x **SUPR** i **NNBSP** d x **HT STX** if **SP ETX** x∈ **MSU** S **HT LF** tan **SP** π **HT STX** otherwise **SP** (with **SP SME** π≃3.141 **EME**) **ETX HT SCI # EME** *(spaces and line breaks are for readability)*

	<me>f(x)={<lss/><mx c='cc'> <mr><mc>∑<blw/><me>i=1</me> <abv/>s<lss s='v'/>x<rsp/>i</mc> <mc c='18'><txt>if </txt>x<0 </mc></mr> <mr><mc><me><st s='C'>∫</st><rsb/> 1 <rsp/> s</me><lss s='v'/>x <rsp/> i NNBSP d x </mc><mc> <txt>if </txt>x∈ <st s='A'>S</st> </mc></mr> <mr><mc>tan π</mc><mc> <txt>otherwise (with <me>π≃3.141</me>)</txt> </mc></mr></mx></me>
	${f(x)=\{$(\[; LF ∑$/{i=1}$\s$<x^i \!$" 'if SP"x<0 \!LF {$C∫_1^s}$<x^i NNBSP d x \! 'if SP"x∈ $AS \!LF tan π \! 'otherwise (with ${π≃3.141})" \!LF \]} *(To make the example more explicit, some characters have been given as short names, like for the C1 version. That is for readability here only.)*

The RTL math examples here are taken from https://www.w3.org/TR/arabic-math/. We also copy (part of) some images from that reference. We have not copied the MathML; it is *way* too large; see the reference to see the MathML notation.

Math expression display	Proposed control code representation *(due to bidi display processing, the display of the code here may be messed up; as it may be (in various ways!!) in other editors or terminal emulators where bidi is implemented)*
	SME 0 > س NBSP NBSP STX إذاكان ETX NBSP NBSP NBSP SME ب SUPL س EME RSS SME MMS ∑ BLW SME 1=ب EME ABV ص EME EME *(most spaces: readability)*
	<me> 0 > س &nbsp;&nbsp;<txt> إذاكان </txt> &nbsp;&nbsp;&nbsp;<me> ب <lsp/> س </me> <rss/> <me> <st s='C' m='MIRROR'>∑</st> <blw/> <me> 1=ب </me> <abv/> ص </me></me>
	${0>س NBSP NBSP 'إذاكان" NBSP NBSP NBSP {ب$~س}$) {\%∑$/{1=ب}$\ص}}
	SME حد BLW SME x → π/١٠ EME sin x=١ MHD ۴ (√ LSS ۵ - ١) EME *(most spaces: readability)*
	<me> حد <blw/><me>x ⟶ π/١٠</me>sin x=١ <dv/> ۴ (√ <lss/> ۵ - ١) </me>
	${حد$/{x⟶π/١٠} sin x=١\/۴ (√$(۵ - ١) }

13 Conclusion

In this paper two proposals for markup for mathematical expressions has been given. One based on C1 (and in matrices also C0) control codes. Apart from NLFs and tabs, the used control codes have not been given any semantics by ECMA-48 (ISO/IEC 6429) and were left for future standardisation (outside of ECMA-48, just like some other control codes are defined outside of

ECMA-48). The other equivalent format is compatible with HTML/XML, using “tags” (with “attributes”) instead of C1 control codes. In addition, Appendix A below will give a markdown version of the format, for easier direct keyboard input.

The two alternative representations are equivalent, and easily mappable from one to the other. Both are succinct and lend themselves to “hand editing” in the source format. This is in contrast to MathML and OMML which are both very verbose and essentially unmanageable for hand editing, requiring an editor which handles the representation in the background, as it is so unwieldy. The here proposed representations, however, have a succinctness comparable to TeX, where authors regularly edit the code for the math expressions “by hand”.

Both of the representations require a parsing step to determine the arguments to the layout operations. Both require an extra parsing step to determine the arguments to mathematical operators and other structures. This extra parsing step is *not* required for displaying the math expression and will be different for different branches of mathematics/logic/physics/chemistry, and even for different traditions in those branches. Therefore, this extra parsing step is not specified here, and thus there are no rules for fence matching, operator precedence, etc.

Both representations are geared to take advantage of the rich set of potential operators (or other notation, like roots and *n*-ary operators) provided by Unicode, avoiding the need to define special controls for those. Both representations use Unicode characters appropriately, for instance not applying combining characters to math expressions (they apply only to base characters), nor are so-called “non-characters” used for control purposes.

For the goals we set out at the start of this paper, we have achieved them. Repeating them here for convenience:

a) Enable proper math expression layout and styling for at least basic math expressions, preferably more.

b) Use control characters from the C1 area to control the math expression layout and formatting succinctly. Ordinary printable characters stand for themselves, not for any control function.

c) Give an alternative format in HTML/XML style with the same expressive power as in b), but still be succinct (in contrast to e.g. MathML and OMML which are very verbose).

d) Handle “fences” (parentheses, brackets, etc.) correctly. Every such character should be displayed if occurring in a properly formed math expression, but there must be no nesting requirement of any kind for such characters (that would violate common notations).

e) Handle letter and digit styling properly. No “MATHEMATICAL” letters and digits (disallowed for use with the styling and layout controls proposed here), but instead use styling controls. Plus, for convenience, math style defaults for some common names.

f) Properly allow for multilettered names for functions, variables, chemical elements, units, and more.

g) Handle RTL (right to left) math expressions properly. RTL math expressions are sometimes used in conjunction with RTL scripts, like Arabic or Hebrew. That includes mirroring symbols (when the author of the expression wants it) also for characters that do not have a mirrored “twin” character in *BidiMirroring.txt* or *ExtraMirroring.txt*, but also *not* to mirror when that would be wrong to do, i.e., no automatic mirroring anywhere in a math expression. Plus, no automatic rearrangement of math expression parts.

h) No use of “non-character” characters in the formats. (“Non-characters” are intended for purely internal use, and must not be used for data interchange, like in a file format that, here, may include math expressions.)

i) The systems proposed here are well integrable with the “surrounding” text formatting system. The C1 based system is well integrable with ECMA-48 styling, and the HTML/XML compatible variant can be well integrated with XHTML, HTML(5) and SVG/<text>, and can likely be well integrated with several other XML or SGML based document formats.

Regarding point a), the system(s) presented here is about as powerful as MathML and OMML, but without their rather extreme verbosity. The system presented here should therefore be usable both for web pages (based on HTML and maybe also SVG) and other documents (using the XML compatible representation or one or the other two representations proposed or suggested here) where math expressions are given.

14 REFERENCES

[AsciiMath] *ASCII Math*, http://asciimath.org/. (Is integrated into PlantUML.)

[AsciiMathJax] https://math.chapman.edu/~jipsen/asciimathjax/asciimathjax.pdf.

[Bidi math comments] Marcel Schneider, *Bidi-mirroring mathematical symbols issues feedback*, http://www.unicode.org/L2/L2017/17438-bidi-math-fdbk.html, 2018-01-18.

[BidiMirroring] http://www.unicode.org/Public/UNIDATA/BidiMirroring.txt.

[C0/C1 stability] *C0/C1 stability*, Kent Karlsson, https://www.unicode.org/L2/L2022/22013r-c0-c1-stability.pdf.

[ECMA-48] *Control Functions for Coded Character Sets*, 5th ed., June 1991, https://www.ecma-international.org/publications/files/ECMA-ST/Ecma-048.pdf.

[Styling upd. ECMA-48] *Text styling control sequences – modernised ECMA-48 (ISO/IEC 6429) text styling*, 2025, https://github.com/kent-karlsson/control/blob/main/ecma-48-style-modernisation-2025-11.pdf, Kent Karlsson.

[eqn/troff] *Typesetting Equations with eqn*. https://www.oreilly.com/library/view/unix-text-processing/9780810462915/Chapter09.html, Unix text processing, chapter 9. Dale Dougherty and Tim O'Reilly, 1987.

[ExtraMirroring] *Glyph mirroring: ExtraMirroring.txt*, Kent Karlsson, https://www.unicode.org/L2/L2022/22026r-non-bidi-mirroring.pdf. See also appendices C and D here.

[HTML] *HTML* – World Wide Web Consortium (W3C), https://www.w3.org/html.

[ISO/IEC 6429:1992] Information technology – *Control functions for coded character sets*, 3rd ed. Same as [ECMA-48] 5th ed.

[ISO/IEC 10646] Information technology – *Universal Coded Character Set* (UCS), https://www.iso.org/standard/76835.html. Same coded characters as in [Unicode].

[LaTeX] *The LaTeX Companion*, Goossens, Mittelbach, Samarin, 1994.

[LaTeXW] *The LaTeX Web Companion*, Michel Goossens, Sebastian Rahtz, 1999.

[Markdown] *Markdown*; https://www.markdownguide.org/, https://commonmark.org/, https://daringfireball.net/projects/markdown/; and several other variants.

[MathJax] https://www.mathjax.org/.

[OMML] ECMA-376-1:2016, *Office Open XML File Formats — Fundamentals and Markup Language Reference*, https://www.ecma-international.org/wp-content/uploads/ECMA-376-Fifth-Edition-Part-1-Fundamentals-And-Markup-Language-Reference.zip, Office Math Markup Language, section 22.1 *Math* (pages 3603 to 3724).

[PlantUML] https://plantuml.com/, a graphic drawing system geared towards certain diagram types.

[MathML] *MathML* – World Wide Web Consortium (W3C), https://www.w3.org/Math/.

[SVG] Scalable Vector Graphics – World Wide Web Consortium (W3C), https://www.w3.org/Graphics/SVG.

[TeX] Knuth, Donald Ervin, *The TeXbook*, Computers and Typesetting, Addison-Wesley, ISBN 0-201-13448-9, 1984. (Plus later developments.)

[Unicode] *The Unicode Standard*, http://www.unicode.org/versions/latest/, web site https://home.unicode.org/, http://unicode.org/main.html, …, 2017, …

[UnicodeMath] Murray Sargent III, *UnicodeMath – A Nearly Plain-Text Encoding of Mathematics*, https://www.unicode.org/notes/tn28/UTN28-PlainTextMath-v3.1.pdf, 2016-11-16.

[Unicode math support] Barbara Beeton, Asmus Freytag, Murray Sargent III, *Unicode support for mathematics*, Unicode Technical Report 25, http://www.unicode.org/L2/L2017/17165-tr25-15-unicode-math.pdf, 2017-05-10.

Appendix A – Markdown format

We have given a C1 control codes-based format for math expressions, and an equivalent XML/HTML compatible format. They are equivalent with each other, but not with any other existing math expression representation format.

However, both the formats we have given have some "quirks": for the C1 version it had to be adapted to how C1 (and C0) is defined in ECMA-48 . For the HTML/XML compatible format, we of course had to adapt to how XML is defined (but with the added requirement for special-purpose parsing). The C1 version has some input problems since most text editors cannot deal with control codes. The XML/HTML/SVG compatible version, while purely in characters allowed in XML, still has the quirks of XML. In many instances where pseudo-formatted input is supported, often referred to as "mark-down", a special format is used that is neither control codes nor XML control tags. There is a variety of such formats. Wikipedia has one. Trac has another, Jira yet another. However, they are supposedly easy to input from a keyboard. Wikipedia has as a kind of "mark-down" for math expression, TeX math expressions. They are relatively easy to input from the keyboard, if you know TeX math expression format, and its quirks…

Not that we can ever make a quirk-free format, but at least we can make one that is equivalent to the formats already presented in this paper but is also relatively easy to input from the keyboard. One simple approach to getting inputability from the keyboard is to use this markdown format. Unlike most other (non-math expression) markdowns, this markdown format is fully equivalent to the nominal formats.

- **${** for an outermost **SME**, including for an outermost **SME** in a text block,
- **{** for a non-outermost **SME**, **\{** for {, **}** for **EME**, **\}** for }, though not in a text block,
- **$** for **SCI** (except if the **SCI** is followed by a C0 control) and **\$** for $, so **SCI A** is given as **$A**, etc., in this markdown notation,
- **'** for **STX**, **\'** for ' and **''** (two ') for **SCI STX**, in a math markdown math expression but not in an embedded text or pragma block,
- **"** for **ETX**, **\"** for " and **""** for **SCI ETX**, but only in an embedded text or pragma block (in this math markdown),
- only in a text block: **\@** for @ (symbol, so it will be math bracketed, and **@**<…>**@** for **CSI**<…>**m**, allowing for ECMA-48 style setting (SGR, including proposed updates and extensions, like **@76@** for supercript size, suitable for "inline" **MHD** (**\/**) arguments (e.g. **${'@76@${x}"\/'@76@${y}"}**)) in embedded texts (the setting may be ignored in contexts that do not allow for any "in-text" styling, but otherwise mapped to the styling mechanism available); style changes are auto-terminated at the end of the text block, but may be terminated explicitly by an appropriate ECMA-48 style resetting (*except* **@0@** (**CSI0m**), which is only for terminal emulators),
- **^** for **SCI ^**, **_** for **SCI _** (as shortcuts), **\^** for ^, **_** for _, though not in a text block,
- (left-subscript, left-superscript, above-script, below-script are not suitable for shortcuts),
- \\ for \ (note: \ is commonly used for set minus), though not in a text block,
- **\/** for **MHD**, **\%** for **MMS**, when not in a text block,
- **\[**[**.,:;**] (optionally followed by *NLF*) for **SCI**[**.,:;**] (rather than **$**[**.,:;**]), though not in a text block, **\]** (optionally preceded by *NLF*) for **SCI #,** if for a math matrix started with **\[**[**.,:;**].
- math matrix cell terminators (i.e., if in a math matrix, and not explicitly math bracketed): **\!** for **HT** (left adj.), **\?** for **HTJ** (right adj.), **\&** for **SCI HT** (center align), **\#** for **SCI VT** (numeric align), and **\,** for **BS** (where **BS** is U+0094); note that **$"**, **$4**, etc. are already covered above (third point),
- math matrix row separators (i.e., if in a math matrix, and not explicitly math bracketed): *NLF*s, just like for the C1 version, (note that **{** *NLF* **}** can be used to get a non-row separating line break inside a math matrix), **\;** could be used where *NLF* cannot be used.

It will have almost all the quirks of the C1 version (apart from the use of C0/C1 control characters, where only *NLF*s are still used) but input from a common keyboard is possible. Special symbols may be inputable via a palette, or by using character name references, e.g., **¤***alpha***¤** (while no list is given here, fow now, compare character reference names for LaTeX as well as for “UnicodeMath”) or a numeric character reference (**¤***nnnn***¤** for a decimal reference, **§***mmm***§** for a hexadecimal reference (if a ¤ or a § is needed as a symbol in a (markdown) math expression, use **\§** for § and **\¤** for ¤, or use a character reference). Note that ¤ and § when given as a character reference are just the symbols. A character reference shall not reference a non-permitted character (mostly emoji and characters with a compatibility decomposition) and referencing a combining character might not work. A numeric character reference cannot be used for a character less than A0 (hexadecimal), named or special character references (like \\, \@) do not have that limitation, and must stand for a Unicode scalar value. Using character references might not allow for combining characters to be applied to the resulting character of the character reference, and a character reference for a letter will result in an single-letter identifier (the letter will not be a part of a multiletter identifier), that goes also for character references for (non-ASCII) decimal digits. Numeric or named character references cannot refer to character that we here regard as control characters (like DIVISION SLASH). These references are intended to make typing in of symbols (or non-Latin letters) easier (or at all possible in the case of special character references), but the preferred representation is the character direcly when possible.

This markdown representation can be used in several contexts, including HTML, SVG and other XML-like contexts, via CDATA blocks. This representation is a possible alternative to using the XML compatible markup (like HTML and SVG, which are formally neither XML nor SGML applications anymore) proposed in the main text. This representation can then be used in a CDATA section, e.g.: **<mth><![CDATA[${**E=m c**^**2**}]]></mth>** (one does need to watch out for ‘]]>’ within a markdown math expression that is in a CDATA section(list), then use ‘]]**]]><![CDATA[**>**]]><![CDATA[**’ or ‘]**]]><![CDATA[**]>**]]><![CDATA[**’ instead, or, simpler “]] >” or “]]>”, since that extra space will not be rendered in a math expression).

This format can also be used with other “markdown” markups, like those of Wikipedia, Confluence and others. (Disclaimer: These products are only mentioned as examples; there is no agreement to incorporate the proposed math expression representation into these products.)

While we for the main proposed formats, propose to use the styling mechanism of the “environment”, that does not really work for the markdown format. So we suggest a styling mechanism anchored in (extended/modernised) ECMA-48 styling. Even so, if a markdown format math expression is given inside an HTML document (example above) the inherited styling is inherited from the HTML DOM, *not* a default styling independent of the surrounding styling. For instance, $x + y + z$, can be represented as **${**x+**'@**31**@${**y**}"**+z**}**, (**\@** for @). since **CSI**31**m** is the ECMA-48 control sequence for (dull) red. Note that the scope of the colour change automatically ends where the text block ends (the matching "), no need for an explicit **@**39**@** to reset the colour (to the default colour, which is here the inherited colour, not a global default), unless one wants to end the colour change “early” in the text block. Similarly, for a size change of the y only with a 20% increase in size: **${**x+**'@**76:0:1?2**@${**y**}"**+z**}**, using proposed modernised ECMA-48 (the old way of doing font size changes in ECMA-48 is strongly deprecated in proposed modernised ECMA-48), where 0 stands for relative size and **1?2** stands for 1.2, i.e. a 20 percent increase (reset with **@**76:9**@** is not needed here since the change is autoterminated at the end of the text block).

Appendix B – LALR grammar

One way of making a parser is to use a parser generator. And one of the types of grammars some of these are based on is so-called LALR(1) grammars. And one of the most well-known parser generators based on LALR(1) grammars is *bison*. Below we give an LALR(1) grammar in bison syntax for generating a parser for the C1 variant of the math expressions as proposed here. Some things are “relegated” to a lexer (as usual when using this tool). But we do not give the details of the lexer here.

```
/* This LALR grammar (bison syntax) is given as an aid to implementers.          */
/* It illustrates math expression parsing. It is not intended to be used in      */
/* actual implementations, since it is for a single isolated math expression,    */
/* and several things are left open. It is for illustration and comparison only. */

/* The LALR grammar here is for a math expression, C1 variant, in isolation.     */
/* Having (probably several) math expressions embedded in a text is not handled. */
/* Nor do we here give any parsing of STX...ETX apart from embedded SME...EME.   */

/* There is no requirement to use an LALR parser generator, and there is no      */
/* requirement that the split between lexer and parser be made as done below.    */

/* The parse tree building, in bison coded in C or C++, is not given here.       */

/* The parsing for the XML/HTML compatible variant (working on the DOM) is       */
/* similar, except that <me>, <txt>, <mx>, <mr>, <mc> are already parsed by      */
/* the XML/HTML parser, 'attributes' are already parsed (as actual attributes),  */
/* as is handling of white-space, plus that embedded texts, for the HTML         */
/* subvariant, can contain HTML tags (DOM nodes) and CSS style specification.    */


/* Collecting letters to ID, digits to NUM and symbols to SYM is done in a       */
/* lexer, which is not detailed here. CHAR_NON_SME is for STX...ETX content.     */
%token ID NUM SYM txtchar

/* Controls for math expression layout.                                          */
/* The lexer needs to map certain character sequences to these tokens.           */
%token SME EME PHA VPHA RECT SP HT HTJ NLF STYLE BINOP NONBINOP MMS STX ETX
%token BLW BLWS BLWT BLWU ABV ABVS ABVT ABVU SUPL SUBL SUBR SUPR SCISTX SCIETX
%token LSS LSST LSSU LSSV RSS RSST RSSU RSSV MHD U2215 U29F5 U29F8 U29F9
%token SCIHT SCIVT SCIET SCIQ SCIQQ SCIBS PLD PLU /* math matrix adjustments */
%token SCI0 SCI1 SCI2 SCI3 SCI4 SCI5 SCI6 SCI7 SCI8 SCI9 /* vertical centre */
%token SCIC SCIP SCICC SCIPP SCIH  /* math matrix start and end */

%expect 0  /* There are no ambiguities in the LALR(1) grammar below       */
%start me  /* Here we only consider a single math expression in isolation */
%%

/* The lexer must insert STX...ETX around strings of no-break spaces. */
me : SME /* Push 'math expression' lexing state for lexer */
       phantom rectangular ws me-seq
     EME /* Pop 'math expression' lexing state for lexer */ ;
me-seq : me-seq stretch ws | me-seq cmt ws | %empty ;
phantom : PHA | VPHA | %empty ;
rectangular : RECT | %empty ;
ws : ws SP | ws HT | ws NLF | %empty ;

st : STYLE id ;
/* The STYLE may be explicit, but may be inserted by the lexer (via default styles). */
/* The lexer may defer the math style determination by using a "deferred" '$' style. */


/* The SYM may come from a mapping, e.g. mapping HYPHEN-MINUS to MINUS SIGN. */
id : ID | NUM | SYM | MMS SYM | BINOP SYM |
     BINOP MMS SYM | NONBINOP SYM | NONBINOP MMS SYM ;
```

```
/* In a TXT the lexer must insert SME...EME around strings of symbols,  */
/* including punctuation and spaces, except pure spaces.                */
/* Like elsewhere in math expressions, bidi controls are not permitted. */
txt : STX /* Push 'embedded text' lexing state for lexer */
        texts
      ETX /* Pop 'embedded text' lexing state for lexer */ ;
cmt : SCISTX /* Push 'embedded text' lexing state for lexer */
        texts
      SCIETX /* Pop 'embedded text' lexing state for lexer */ ;
texts : texts txtchar | texts me | ;

closed : me | st | txt | mx ;
stack : closed below above ;
below : below blw closed | ;
blw : BLW | BLWS | BLWT | BLWU ;
above : above abv closed | ;
abv : ABV | ABVS | ABVT | ABVU ;

around : supl subl stack subr supr
       |      subl stack subr supr
       | supl      stack subr supr
       |           stack subr supr ;
supl : closed SUPL ;
subl : closed SUBL ;
subr : SUBR closed | ;
supr : SUPR closed | ;

frac : around dv vc-adjf around ;
dv : MHD | U2215 | U29F5 | U29F8 | U29F9 ;
vc-adjf : SCI0 | SCI1 | ;

stretch : left sbase right ;
sbase : around | frac ;
left : left stack lss | ;
lss : LSS | LSST | LSSU | LSSV ;
right : rss stack right | ;
rss : RSS | RSST | RSSU | RSSV ;

/* TODO?: If NLF or SCI# is encountered without a preceding 'term', the lexer inserts HT.
*/
mc : sp mc-seq term ;
mc-seq : mc-seq stretch sp | ;
sp : sp SP | ;
term : HT | HTJ | SCIHT | SCIVT ; /* SCI LS and SCI VT are both mapped to SCIVT */

mr : mr mc
   | mr SCIET longer mc
   | mr SCIQ longer mc
   | mr SCIQQ longer mc
   | mr SCIBS shorter mc
   | ;
longer : longer SCIET | longer SCIQ | longer SCIQQ | ;
shorter : shorter SCIBS | ;

mrs : mrs NLF mr
    | mrs NLF PLD plds mr
    | mrs NLF PLU plus mr
    | ;
plds : plds PLD | ;
plus : plus PLU | ;

mx : mx-start vc-adj mr mrs SCIH ;
mx-start : SCIC | SCICC | SCIP | SCIPP ;
vc-adj : SCI0 | SCI1 | SCI2 | SCI3 | SCI4 | SCI5 | SCI6 | SCI7 | SCI8 | SCI9 | ;

%%
```

Appendix C – Adding *BidiMirroring.txt* data for arrows in math expressions

This is a copy of the data file, `ExtraMirroring-txt`, proposed in https://www.unicode.org/L2/L2022/22026r-non-bidi-mirroring.pdf. This copy is given here for completeness since this data file has not yet become available as a part of the Unicode database.

```
# ExtraMirroring-proposed-v2.txt
# Date: 2022-07-09
# © 2022 Unicode®, Inc.
# For terms of use, see https://www.unicode.org/terms_of_use.html
#
# Unicode Character Database
# For documentation, see https://www.unicode.org/reports/tr44/
#
# Extended_Mirrored_property...
#
# This file is an informative contributory data file in the
# Unicode Character Database.
#
# This data file lists symbol characters that have the Bidi_Mirrored=No property
# value, for which there is another Unicode character that has a glyph that is
# the mirror image of the original character's glyph. Letter characters are mostly
# excluded. Included are mathematical symbols, arrows and arrow-like symbols and
# other symbols. Arrows and arrow-like symbols were never given the Bidi_Mirrored=Yes
# property, though many are non-left-right symmetric and many of those already
# have a (left/right) "mirror" character encoded. Many of the more common arrows
# are used in mathematical and formal logic expressions. See also
# https://unicode.org/reports/tr9 HL 6.
#
# The repertoire covered by the file is Unicode 14.0.0.
#
# The file contains a list of lines with mappings from code point(s) to (an)other
# code point(s) for character-based mirroring, extending BidiMirroring.txt. There
# are a number of would-be-mirrorable characters that do not have a mirror character.
#
# Each mapping line contains two fields, separated by a semicolon ('; ').
# Each of the two fields contains code point(s) represented as
# variable-length hexadecimal value(s) with 4 to 6 digits.
# A comment indicates where the characters are "BEST FIT" mirroring.
#
# "Mirrorable" code points for which Bidi_Mirrored=No, but for which
# no appropriate character currently exist with mirrored glyph, are
# listed as comments at the end of the file.
#
# This file was originally created by Kent Karlsson.

002C; 2E41 # COMMA
002F; 005C # SOLIDUS
003B; 204F # SEMICOLON
003F; 2E2E # QUESTION MARK
005C; 002F # REVERSE SOLIDUS
005C 0304; 29F6 # reverse solidus with overbar
00AC; 2310 # NOT SIGN


037B; 03F2 # GREEK SMALL REVERSED LUNATE SIGMA SYMBOL
037C; 037D # GREEK SMALL DOTTED LUNATE SIGMA SYMBOL
037D; 037C # GREEK SMALL REVERSED DOTTED LUNATE SIGMA SYMBOL
03F2; 037B # GREEK LUNATE SIGMA SYMBOL
03F5; 03F6 # GREEK LUNATE EPSILON SYMBOL
03F6; 03F5 # GREEK REVERSED LUNATE EPSILON SYMBOL
03F9; 03FD # GREEK CAPITAL LUNATE SIGMA SYMBOL
03FD; 03F9 # GREEK CAPITAL REVERSED LUNATE SIGMA SYMBOL
03FE; 03FF # GREEK CAPITAL DOTTED LUNATE SIGMA SYMBOL
03FF; 03FE # GREEK CAPITAL REVERSED DOTTED LUNATE SIGMA SYMBOL
```

```
# These are not recommended quotation mark pairs; they are just mirror character pairs.
# For recommended quotation mark pairs, see CLDR.
2019; 201B # RIGHT SINGLE QUOTATION MARK
201B; 2019 # SINGLE HIGH-REVERSED-9 QUOTATION MARK
201D; 201F # RIGHT DOUBLE QUOTATION MARK
201E; 2E42 # DOUBLE LOW-9 QUOTATION MARK
201F; 201D # DOUBLE HIGH-REVERSED-9 QUOTATION MARK
2E42; 201E # DOUBLE LOW-REVERSED-9 QUOTATION MARK

2032; 2035 # PRIME
2033; 2036 # DOUBLE PRIME
2034; 2037 # TRIPLE PRIME
2035; 2032 # REVERSED PRIME
2036; 2033 # REVERSED DOUBLE PRIME
2037; 2034 # REVERSED TRIPLE PRIME
204C; 204D # BLACK LEFTWARDS BULLET
204D; 204C # BLACK RIGHTWARDS BULLET
204F; 003B # REVERSED SEMICOLON
2190; 2192 # LEFTWARDS ARROW
2192; 2190 # RIGHTWARDS ARROW
2196; 2197 # NORTH WEST ARROW
2197; 2196 # NORTH EAST ARROW
2198; 2199 # SOUTH EAST ARROW
2199; 2198 # SOUTH WEST ARROW
219A; 219B # LEFTWARDS ARROW WITH STROKE
219B; 219A # RIGHTWARDS ARROW WITH STROKE
219C; 219D # LEFTWARDS WAVE ARROW   ; pointing upwards
219D; 219C # RIGHTWARDS WAVE ARROW  ; pointing upwards
219E; 21A0 # LEFTWARDS TWO HEADED ARROW
21A0; 219E # RIGHTWARDS TWO HEADED ARROW
21A2; 21A3 # LEFTWARDS ARROW WITH TAIL
21A3; 21A2 # RIGHTWARDS ARROW WITH TAIL
21A4; 21A6 # LEFTWARDS ARROW FROM BAR
21A6; 21A4 # RIGHTWARDS ARROW FROM BAR
21A9; 21AA # LEFTWARDS ARROW WITH HOOK   ; hook on the upper side
21AA; 21A9 # RIGHTWARDS ARROW WITH HOOK  ; hook on the upper side
21AB; 21AC # LEFTWARDS ARROW WITH LOOP   ; loop on the upper side
21AC; 21AB # RIGHTWARDS ARROW WITH LOOP  ; loop on the upper side
21B0; 21B1 # UPWARDS ARROW WITH TIP LEFTWARDS
21B1; 21B0 # UPWARDS ARROW WITH TIP RIGHTWARDS
21B2; 21B3 # DOWNWARDS ARROW WITH TIP LEFTWARDS
21B3; 21B2 # DOWNWARDS ARROW WITH TIP RIGHTWARDS
21B6; 21B7 # ANTICLOCKWISE TOP SEMICIRCLE ARROW
21B7; 21B6 # CLOCKWISE TOP SEMICIRCLE ARROW
21BA; 21BB # ANTICLOCKWISE OPEN CIRCLE ARROW     ; head on top
21BB; 21BA # CLOCKWISE OPEN CIRCLE ARROW         ; head on top
21BC; 21C0 # LEFTWARDS HARPOON WITH BARB UPWARDS
21BD; 21C1 # LEFTWARDS HARPOON WITH BARB DOWNWARDS
21BE; 21BF # UPWARDS HARPOON WITH BARB RIGHTWARDS
21BF; 21BE # UPWARDS HARPOON WITH BARB LEFTWARDS
21C0; 21BC # RIGHTWARDS HARPOON WITH BARB UPWARDS
21C1; 21BD # RIGHTWARDS HARPOON WITH BARB DOWNWARDS
21C2; 21C3 # DOWNWARDS HARPOON WITH BARB RIGHTWARDS
21C3; 21C2 # DOWNWARDS HARPOON WITH BARB LEFTWARDS
21C4; 21C6 # RIGHTWARDS ARROW OVER LEFTWARDS ARROW
21C5; 21F5 # UPWARDS ARROW LEFTWARDS OF DOWNWARDS ARROW
21C6; 21C4 # LEFTWARDS ARROW OVER RIGHTWARDS ARROW
21C7; 21C9 # LEFTWARDS PAIRED ARROWS
21C9; 21C7 # RIGHTWARDS PAIRED ARROWS
21CD; 21CF # LEFTWARDS DOUBLE ARROW WITH STROKE
21CF; 21CD # RIGHTWARDS DOUBLE ARROW WITH STROKE
21D0; 21D2 # LEFTWARDS DOUBLE ARROW
21D2; 21D0 # RIGHTWARDS DOUBLE ARROW
21D6; 21D7 # NORTH WEST DOUBLE ARROW
21D7; 21D6 # NORTH EAST DOUBLE ARROW
21D8; 21D9 # SOUTH EAST DOUBLE ARROW
21D9; 21D8 # SOUTH WEST DOUBLE ARROW
21DA; 21DB # LEFTWARDS TRIPLE ARROW
21DB; 21DA # RIGHTWARDS TRIPLE ARROW
21DC; 21DD # LEFTWARDS SQUIGGLE ARROW        ; pointing directly left
21DD; 21DC # RIGHTWARDS SQUIGGLE ARROW       ; pointing directly right
21E0; 21E2 # LEFTWARDS DASHED ARROW     ; quadruple dash
21E2; 21E0 # RIGHTWARDS DASHED ARROW    ; quadruple dash
21E4; 21E5 # LEFTWARDS ARROW TO BAR
21E5; 21E4 # RIGHTWARDS ARROW TO BAR
21E6; 21E8 # LEFTWARDS WHITE ARROW
21E8; 21E6 # RIGHTWARDS WHITE ARROW
```

```
21F4; 2B30 # RIGHT ARROW WITH SMALL CIRCLE
21F5; 21C5 # DOWNWARDS ARROW LEFTWARDS OF UPWARDS ARROW
21F6; 2B31 # THREE RIGHTWARDS ARROWS
21F7; 21F8 # LEFTWARDS ARROW WITH VERTICAL STROKE
21F8; 21F7 # RIGHTWARDS ARROW WITH VERTICAL STROKE
21FA; 21FB # LEFTWARDS ARROW WITH DOUBLE VERTICAL STROKE
21FB; 21FA # RIGHTWARDS ARROW WITH DOUBLE VERTICAL STROKE
21FD; 21FE # LEFTWARDS OPEN-HEADED ARROW
21FE; 21FD # RIGHTWARDS OPEN-HEADED ARROW
2205; 29B0 # EMPTY SET ; glyph is ALWAYS based on a circle, NEVER based on a 0 or similar-
        to-0 glyph; for a 0-like glyph (not recommended) use <U+0030, U+0338>.
220B 0307; 22F5 # contains with dot above
220B 0331; 22F8 # contains with underbar
#2226; 2226 # [BEST FIT] NOT PARALLEL TO
223D 0307; 2A6A # reversed tilde operator with dot above
223D 0338; 2241 # reversed not tilde
#223D # REVERSED TILDE ; note: lazy s is NOT a glyph variant
2241; 223D 0338 # NOT TILDE (has bidiMirroring=Yes)
2244; 22CD 0338 # NOT ASYMPTOTICALLY EQUAL TO
#2246; 2246 # [BEST FIT] APPROXIMATELY BUT NOT ACTUALLY EQUAL TO
#2247; 2247 # [BEST FIT] NEITHER APPROXIMATELY NOR ACTUALLY EQUAL TO
224C 0307; 2A6D # reversed congruent with dot above
226B 0331; 2AA3 # double nested greater-than with underbar
22A3 0338; 22AC # reversed does not prove
22AC; 22A3 0338 # DOES NOT PROVE
22AD; 2AE4 0338 # NOT TRUE
22AE; 2AE3 0338 # DOES NOT FORCE
22AF; 2AE5 0338 # NEGATED DOUBLE VERTICAL BAR DOUBLE RIGHT TURNSTILE
22CD 0338; 2244 # not reversed tilde equals
22F5; 220B 0307 # ELEMENT OF WITH DOT ABOVE
22F8; 220B 0331 # ELEMENT OF WITH UNDERBAR
230C; 230D # BOTTOM RIGHT CROP
230D; 230C # BOTTOM LEFT CROP
230E; 230F # TOP RIGHT CROP
230F; 230E # TOP LEFT CROP
2310; 00AC # REVERSED NOT SIGN
231C; 231D # TOP LEFT CORNER  ; left Quine quote symbol
231D; 231C # TOP RIGHT CORNER ; right Quine quote symbol
231E; 231F # BOTTOM LEFT CORNER
231F; 231E # BOTTOM RIGHT CORNER
2326; 232B # ERASE TO THE RIGHT
232B; 2326 # ERASE TO THE LEFT
2347; 2348 # APL FUNCTIONAL SYMBOL QUAD LEFTWARDS ARROW
2348; 2347 # APL FUNCTIONAL SYMBOL QUAD RIGHTWARDS ARROW
239B; 239E # LEFT PARENTHESIS UPPER HOOK
239C; 239F # LEFT PARENTHESIS EXTENSION
239D; 23A0 # LEFT PARENTHESIS LOWER HOOK
239E; 239B # RIGHT PARENTHESIS UPPER HOOK
239F; 239C # RIGHT PARENTHESIS EXTENSION
23A0; 239D # RIGHT PARENTHESIS LOWER HOOK
23A1; 23A4 # LEFT SQUARE BRACKET UPPER CORNER
23A2; 23A5 # LEFT SQUARE BRACKET EXTENSION
23A3; 23A6 # LEFT SQUARE BRACKET LOWER CORNER
23A4; 23A1 # RIGHT SQUARE BRACKET UPPER CORNER
23A5; 23A2 # RIGHT SQUARE BRACKET EXTENSION
23A6; 23A3 # RIGHT SQUARE BRACKET LOWER CORNER
23A7; 23AB # LEFT CURLY BRACKET UPPER HOOK
23A8; 23AC # LEFT CURLY BRACKET MIDDLE PIECE
23A9; 23AD # LEFT CURLY BRACKET LOWER HOOK
23AB; 23A7 # RIGHT CURLY BRACKET UPPER HOOK
23AC; 23A8 # RIGHT CURLY BRACKET MIDDLE PIECE
23AD; 23A9 # RIGHT CURLY BRACKET LOWER HOOK
23B0; 23B1 # UPPER LEFT OR LOWER RIGHT CURLY BRACKET SECTION
23B1; 23B0 # UPPER RIGHT OR LOWER LEFT CURLY BRACKET SECTION
23E9; 23EA # BLACK RIGHT-POINTING DOUBLE TRIANGLE
23EA; 23E9 # BLACK LEFT-POINTING DOUBLE TRIANGLE
23ED; 23EE # BLACK RIGHT-POINTING DOUBLE TRIANGLE WITH VERTICAL BAR
23EE; 23ED # BLACK LEFT-POINTING DOUBLE TRIANGLE WITH VERTICAL BAR
25B6; 25C0 # BLACK RIGHT-POINTING TRIANGLE
25B7; 25C1 # WHITE RIGHT-POINTING TRIANGLE
25B7; 25C1 # WHITE RIGHT-POINTING TRIANGLE
25B8; 25C2 # BLACK RIGHT-POINTING SMALL TRIANGLE
25B9; 25C3 # WHITE RIGHT-POINTING SMALL TRIANGLE
25BA; 25C4 # BLACK RIGHT-POINTING POINTER
25BB; 25C5 # WHITE RIGHT-POINTING POINTER
25C0; 25B6 # BLACK LEFT-POINTING TRIANGLE
25C1; 25B7 # WHITE LEFT-POINTING TRIANGLE
```

```
25C1; 25B7 # WHITE LEFT-POINTING TRIANGLE
25C2; 25B8 # BLACK LEFT-POINTING SMALL TRIANGLE
25C3; 25B9 # WHITE LEFT-POINTING SMALL TRIANGLE
25C4; 25BA # BLACK LEFT-POINTING POINTER
25C5; 25BB # WHITE LEFT-POINTING POINTER
25E2; 25E3 # BLACK LOWER RIGHT TRIANGLE
25E3; 25E2 # BLACK LOWER LEFT TRIANGLE
25E4; 25E5 # BLACK UPPER LEFT TRIANGLE
25E5; 25E4 # BLACK UPPER RIGHT TRIANGLE
25F8; 25F9 # UPPER LEFT TRIANGLE
25F9; 25F8 # UPPER RIGHT TRIANGLE
25FA; 25FF # LOWER LEFT TRIANGLE
25FF; 25FA # LOWER RIGHT TRIANGLE

261A; 261B # BLACK LEFT POINTING INDEX
261B; 261A # BLACK RIGHT POINTING INDEX
261C; 261E # WHITE LEFT POINTING INDEX
261E; 261C # WHITE RIGHT POINTING INDEX
269E; 269F # THREE LINES CONVERGING RIGHT
269F; 269E # THREE LINES CONVERGING LEFT

27F2; 27F3 # ANTICLOCKWISE GAPPED CIRCLE ARROW    ; head on the left
27F3; 27F2 # CLOCKWISE GAPPED CIRCLE ARROW        ; head on the right
27F4; 2B32 # RIGHT ARROW WITH CIRCLED PLUS
27F5; 27F6 # LONG LEFTWARDS ARROW
27F6; 27F5 # LONG RIGHTWARDS ARROW
27F8; 27F9 # LONG LEFTWARDS DOUBLE ARROW
27F9; 27F8 # LONG RIGHTWARDS DOUBLE ARROW
27FB; 27FC # LONG LEFTWARDS ARROW FROM BAR
27FC; 27FB # LONG RIGHTWARDS ARROW FROM BAR
27FD; 27FE # LONG LEFTWARDS DOUBLE ARROW FROM BAR
27FE; 27FD # LONG RIGHTWARDS DOUBLE ARROW FROM BAR
27FF; 2B33 # LONG RIGHTWARDS SQUIGGLE ARROW ; pointing directly right
2900; 2B34 # RIGHTWARDS TWO-HEADED ARROW WITH VERTICAL STROKE
2901; 2B35 # RIGHTWARDS TWO-HEADED ARROW WITH DOUBLE VERTICAL STROKE
2902; 2903 # LEFTWARDS DOUBLE ARROW WITH VERTICAL STROKE
2903; 2902 # RIGHTWARDS DOUBLE ARROW WITH VERTICAL STROKE
2905; 2B36 # RIGHTWARDS TWO-HEADED ARROW FROM BAR
2906; 2907 # LEFTWARDS DOUBLE ARROW FROM BAR
2907; 2906 # RIGHTWARDS DOUBLE ARROW FROM BAR
290C; 290D # LEFTWARDS DOUBLE DASH ARROW
290D; 290C # RIGHTWARDS DOUBLE DASH ARROW
290E; 290F # LEFTWARDS TRIPLE DASH ARROW
290F; 290E # RIGHTWARDS TRIPLE DASH ARROW
2910; 2B37 # RIGHTWARDS TWO-HEADED TRIPLE DASH ARROW   ; with tail
2911; 2B38 # RIGHTWARDS ARROW WITH DOTTED STEM
2914; 2B39 # RIGHTWARDS ARROW WITH TAIL WITH VERTICAL STROKE
2915; 2B3A # RIGHTWARDS ARROW WITH TAIL WITH DOUBLE VERTICAL STROKE
2916; 2B3B # RIGHTWARDS TWO-HEADED ARROW WITH TAIL
2917; 2B3C # RIGHTWARDS TWO-HEADED ARROW WITH TAIL WITH VERTICAL STROKE
2918; 2B3D # RIGHTWARDS TWO-HEADED ARROW WITH TAIL WITH DOUBLE VERTICAL STROKE
2919; 291A # LEFTWARDS ARROW-TAIL
291A; 2919 # RIGHTWARDS ARROW-TAIL
291B; 291C # LEFTWARDS DOUBLE ARROW-TAIL
291C; 291B # RIGHTWARDS DOUBLE ARROW-TAIL
291D; 291E # LEFTWARDS ARROW TO BLACK DIAMOND
291E; 291D # RIGHTWARDS ARROW TO BLACK DIAMOND
291F; 2920 # LEFTWARDS ARROW FROM BAR TO BLACK DIAMOND
2920; 291F # RIGHTWARDS ARROW FROM BAR TO BLACK DIAMOND
2921; 2922 # NORTH WEST AND SOUTH EAST ARROW
2922; 2921 # NORTH EAST AND SOUTH WEST ARROW
2923; 2924 # NORTH WEST ARROW WITH HOOK  ; hook on the lower side
2924; 2923 # NORTH EAST ARROW WITH HOOK  ; hook on the lower side
2925; 2926 # SOUTH EAST ARROW WITH HOOK  ; hook on the upper side
2926; 2925 # SOUTH WEST ARROW WITH HOOK  ; hook on the upper side
2928; 292A # NORTH EAST ARROW AND SOUTH EAST ARROW
292A; 2928 # SOUTH WEST ARROW AND NORTH WEST ARROW
292B; 292C # RISING DIAGONAL CROSSING FALLING DIAGONAL
292C; 292B # FALLING DIAGONAL CROSSING RISING DIAGONAL
2931; 2932 # NORTH EAST ARROW CROSSING NORTH WEST ARROW
2932; 2931 # NORTH WEST ARROW CROSSING NORTH EAST ARROW
2933; 2B3F # WAVE ARROW POINTING DIRECTLY RIGHT
2936; 2937 # ARROW POINTING DOWNWARDS THEN CURVING LEFTWARDS
2937; 2936 # ARROW POINTING DOWNWARDS THEN CURVING RIGHTWARDS
2938; 2939 # RIGHT-SIDE ARC CLOCKWISE ARROW ; this is with "plain" head
2939; 2938 # LEFT-SIDE ARC ANTICLOCKWISE ARROW
293E; 293F # LOWER RIGHT SEMICIRCULAR CLOCKWISE ARROW
```

```
293F; 293E # LOWER LEFT SEMICIRCULAR ANTICLOCKWISE ARROW
2940; 2941 # ANTICLOCKWISE CLOSED CIRCLE ARROW    ; head on top
2941; 2940 # CLOCKWISE CLOSED CIRCLE ARROW        ; head on top
2942; 2943 # RIGHTWARDS ARROW ABOVE SHORT LEFTWARDS ARROW
2943; 2942 # LEFTWARDS ARROW ABOVE SHORT RIGHTWARDS ARROW
2945; 2946 # RIGHTWARDS ARROW WITH PLUS BELOW
2946; 2945 # LEFTWARDS ARROW WITH PLUS BELOW
2947; 2B3E # RIGHTWARDS ARROW THROUGH X
294A; 294B # LEFT BARB UP RIGHT BARB DOWN HARPOON
294B; 294A # LEFT BARB DOWN RIGHT BARB UP HARPOON
294C; 294D # UP BARB RIGHT DOWN BARB LEFT HARPOON
294D; 294C # UP BARB LEFT DOWN BARB RIGHT HARPOON
294F; 2951 # UP BARB RIGHT DOWN BARB RIGHT HARPOON
2951; 294F # UP BARB LEFT DOWN BARB LEFT HARPOON
2952; 2953 # LEFTWARDS HARPOON WITH BARB UP TO BAR
2953; 2952 # RIGHTWARDS HARPOON WITH BARB UP TO BAR
2954; 2958 # UPWARDS HARPOON WITH BARB RIGHT TO BAR
2955; 2959 # DOWNWARDS HARPOON WITH BARB RIGHT TO BAR
2956; 2957 # LEFTWARDS HARPOON WITH BARB DOWN TO BAR
2957; 2956 # RIGHTWARDS HARPOON WITH BARB DOWN TO BAR
2958; 2954 # UPWARDS HARPOON WITH BARB LEFT TO BAR
2959; 2955 # DOWNWARDS HARPOON WITH BARB LEFT TO BAR
295A; 295B # LEFTWARDS HARPOON WITH BARB UP FROM BAR
295B; 295A # RIGHTWARDS HARPOON WITH BARB UP FROM BAR
295C; 2960 # UPWARDS HARPOON WITH BARB RIGHT FROM BAR
295D; 2961 # DOWNWARDS HARPOON WITH BARB RIGHT FROM BAR
295E; 295F # LEFTWARDS HARPOON WITH BARB DOWN FROM BAR
295F; 295E # RIGHTWARDS HARPOON WITH BARB DOWN FROM BAR
2960; 295C # UPWARDS HARPOON WITH BARB LEFT FROM BAR
2961; 295D # DOWNWARDS HARPOON WITH BARB LEFT FROM BAR
2962; 2964 # LEFTWARDS HARPOON WITH BARB UP ABOVE LEFTWARDS HARPOON WITH BARB DOWN
2964; 2962 # RIGHTWARDS HARPOON WITH BARB UP ABOVE RIGHTWARDS HARPOON WITH BARB DOWN
2966; 2968 # LEFTWARDS HARPOON WITH BARB UP ABOVE RIGHTWARDS HARPOON WITH BARB UP
2967; 2969 # LEFTWARDS HARPOON WITH BARB DOWN ABOVE RIGHTWARDS HARPOON WITH BARB DOWN
2968; 2966 # RIGHTWARDS HARPOON WITH BARB UP ABOVE LEFTWARDS HARPOON WITH BARB UP
2969; 2967 # RIGHTWARDS HARPOON WITH BARB DOWN ABOVE LEFTWARDS HARPOON WITH BARB DOWN
296A; 296C # LEFTWARDS HARPOON WITH BARB UP ABOVE LONG DASH
296B; 296D # LEFTWARDS HARPOON WITH BARB DOWN BELOW LONG DASH
296C; 296A # RIGHTWARDS HARPOON WITH BARB UP ABOVE LONG DASH
296D; 296B # RIGHTWARDS HARPOON WITH BARB DOWN BELOW LONG DASH
296E; 296F # UPWARDS HARPOON WITH BARB LEFT BESIDE DOWNWARDS HARPOON WITH BARB RIGHT
296F; 296E # DOWNWARDS HARPOON WITH BARB LEFT BESIDE UPWARDS HARPOON WITH BARB RIGHT
2971; 2B40 # EQUALS SIGN ABOVE RIGHTWARDS ARROW
2972; 2B41 # TILDE OPERATOR ABOVE RIGHTWARDS ARROW
2973; 2B4C # LEFTWARDS ARROW ABOVE TILDE OPERATOR
2974; 2B4B # RIGHTWARDS ARROW ABOVE TILDE OPERATOR
2975; 2B42 # RIGHTWARDS ARROW ABOVE ALMOST EQUAL TO
2976; 2978 # LESS-THAN ABOVE LEFTWARDS ARROW
2977; 2B43 # LEFTWARDS ARROW THROUGH LESS-THAN
2978; 2976 # GREATER-THAN ABOVE RIGHTWARDS ARROW
2979; 297B # SUBSET ABOVE RIGHTWARDS ARROW
297A; 2B44 # LEFTWARDS ARROW THROUGH SUBSET
297B; 2979 # SUPERSET ABOVE LEFTWARDS ARROW
297C; 297D # LEFT FISH TAIL
297D; 297C # RIGHT FISH TAIL
29B0 0304; 29B1 # reversed empty set with overbar
29B0 030A; 29B2 # reversed empty set with small circle above
29B0 20D6; 29B3 # reversed empty set with left arrow above ; not certain that the arrow
       should be mirrored
29B0 20D7; 29B4 # reversed empty set with right arrow above ; -''-
29B0; 2205 # REVERSED EMPTY SET
29B1; 29B0 0304 # EMPTY SET WITH OVERBAR
29B2; 29B0 030A # EMPTY SET WITH SMALL CIRCLE ABOVE
29B3; 29B0 20D6 # EMPTY SET WITH RIGHT ARROW ABOVE ; not certain that the arrow should be
       mirrored
29B4; 29B0 20D7 # EMPTY SET WITH LEFT ARROW ABOVE ; -''-
29F6; 005C 0304 # SOLIDUS WITH OVERBAR
#2A24; 2A24 # [BEST FIT] PLUS SIGN WITH TILDE ABOVE
#2A26; 2A26 # [BEST FIT] PLUS SIGN WITH TILDE BELOW
#2A29; 2A29 # [BEST FIT] MINUS SIGN WITH COMMA ABOVE
2A6A; 223D 0307 # TILDE OPERATOR WITH DOT ABOVE
2A6D; 224C 0307 # CONGRUENT WITH DOT ABOVE
2AA3; 226B 0331 # DOUBLE NESTED LESS-THAN WITH UNDERBAR
2AE3 0338; 22AE # reversed does not force
2AE4 0338; 22AD # reversed not true
2AE5 0338; 22AF # negated double vertical bar double left turnstile
#2AF3; 2AF3 # [BEST FIT] PARALLEL WITH TILDE OPERATOR
```

```
2B00; 2B01 # NORTH EAST WHITE ARROW
2B01; 2B00 # NORTH WEST WHITE ARROW
2B02; 2B03 # SOUTH EAST WHITE ARROW
2B03; 2B02 # SOUTH WEST WHITE ARROW
2B05; 2B95 # LEFTWARDS BLACK ARROW
2B08; 2B09 # NORTH EAST BLACK ARROW
2B09; 2B08 # NORTH WEST BLACK ARROW
2B0A; 2B0B # SOUTH EAST BLACK ARROW
2B0B; 2B0A # SOUTH WEST BLACK ARROW
2B0E; 2B10 # RIGHTWARDS ARROW WITH TIP DOWNWARDS
2B0F; 2B11 # RIGHTWARDS ARROW WITH TIP UPWARDS
2B10; 2B0E # LEFTWARDS ARROW WITH TIP DOWNWARDS
2B11; 2B0F # LEFTWARDS ARROW WITH TIP UPWARDS
2B30; 21F4 # LEFT ARROW WITH SMALL CIRCLE
2B31; 21F6 # THREE LEFTWARDS ARROWS
2B32; 27F4 # LEFT ARROW WITH CIRCLED PLUS
2B33; 27FF # LONG LEFTWARDS SQUIGGLE ARROW  ; pointing directly left
2B34; 2900 # LEFTWARDS TWO-HEADED ARROW WITH VERTICAL STROKE
2B35; 2901 # LEFTWARDS TWO-HEADED ARROW WITH DOUBLE VERTICAL STROKE
2B36; 2905 # LEFTWARDS TWO-HEADED ARROW FROM BAR
2B37; 2910 # LEFTWARDS TWO-HEADED TRIPLE DASH ARROW    ; with tail
2B38; 2911 # LEFTWARDS ARROW WITH DOTTED STEM
2B39; 2914 # LEFTWARDS ARROW WITH TAIL WITH VERTICAL STROKE
2B3A; 2915 # LEFTWARDS ARROW WITH TAIL WITH DOUBLE VERTICAL STROKE
2B3B; 2916 # LEFTWARDS TWO-HEADED ARROW WITH TAIL
2B3C; 2917 # LEFTWARDS TWO-HEADED ARROW WITH TAIL WITH VERTICAL STROKE
2B3D; 2918 # LEFTWARDS TWO-HEADED ARROW WITH TAIL WITH DOUBLE VERTICAL STROKE
2B3E; 2947 # LEFTWARDS ARROW THROUGH X
2B3F; 2933 # WAVE ARROW POINTING DIRECTLY LEFT
2B40; 2971 # EQUALS SIGN ABOVE LEFTWARDS ARROW
2B41; 2972 # REVERSE TILDE OPERATOR ABOVE LEFTWARDS ARROW
2B42; 2975 # LEFTWARDS ARROW ABOVE REVERSE ALMOST EQUAL TO
2B43; 2977 # RIGHTWARDS ARROW THROUGH GREATER-THAN
2B44; 297A # RIGHTWARDS ARROW THROUGH SUPERSET
2B45; 2B46 # LEFTWARDS QUADRUPLE ARROW
2B46; 2B45 # RIGHTWARDS QUADRUPLE ARROW
2B47; 2B49 # REVERSE TILDE OPERATOR ABOVE RIGHTWARDS ARROW
2B48; 2B4A # RIGHTWARDS ARROW ABOVE REVERSE ALMOST EQUAL TO
2B49; 2B47 # TILDE OPERATOR ABOVE LEFTWARDS ARROW
2B4A; 2B48 # LEFTWARDS ARROW ABOVE ALMOST EQUAL TO
2B4B; 2974 # LEFTWARDS ARROW ABOVE REVERSE TILDE OPERATOR
2B4C; 2973 # RIGHTWARDS ARROW ABOVE REVERSE TILDE OPERATOR
2B60; 2B62 # LEFTWARDS TRIANGLE-HEADED ARROW
2B62; 2B60 # RIGHTWARDS TRIANGLE-HEADED ARROW
2B66; 2B67 # NORTH WEST TRIANGLE-HEADED ARROW
2B67; 2B66 # NORTH EAST TRIANGLE-HEADED ARROW
2B68; 2B69 # SOUTH EAST TRIANGLE-HEADED ARROW
2B69; 2B68 # SOUTH WEST TRIANGLE-HEADED ARROW
2B6A; 2B6C # LEFTWARDS TRIANGLE-HEADED DASHED ARROW
2B6C; 2B6A # RIGHTWARDS TRIANGLE-HEADED DASHED ARROW
2B6E; 2B6F # CLOCKWISE TRIANGLE-HEADED OPEN CIRCLE ARROW     ; head on top
2B6F; 2B6E # ANTICLOCKWISE TRIANGLE-HEADED OPEN CIRCLE ARROW ; head on top
2B70; 2B72 # LEFTWARDS TRIANGLE-HEADED ARROW TO BAR
2B72; 2B70 # RIGHTWARDS TRIANGLE-HEADED ARROW TO BAR
2B76; 2B77 # NORTH WEST TRIANGLE-HEADED ARROW TO BAR
2B77; 2B76 # NORTH EAST TRIANGLE-HEADED ARROW TO BAR
2B78; 2B79 # SOUTH EAST TRIANGLE-HEADED ARROW TO BAR
2B79; 2B78 # SOUTH WEST TRIANGLE-HEADED ARROW TO BAR
2B7A; 2B7C # LEFTWARDS TRIANGLE-HEADED ARROW WITH DOUBLE HORIZONTAL STROKE
2B7C; 2B7A # RIGHTWARDS TRIANGLE-HEADED ARROW WITH DOUBLE HORIZONTAL STROKE
2B80; 2B82 # LEFTWARDS TRIANGLE-HEADED ARROW OVER RIGHTWARDS TRIANGLE-HEADED ARROW
2B81; 2B83 # UPWARDS TRIANGLE-HEADED ARROW LEFTWARDS OF DOWNWARDS TRIANGLE-HEADED ARROW
2B82; 2B80 # RIGHTWARDS TRIANGLE-HEADED ARROW OVER LEFTWARDS TRIANGLE-HEADED ARROW
2B83; 2B81 # DOWNWARDS TRIANGLE-HEADED ARROW LEFTWARDS OF UPWARDS TRIANGLE-HEADED ARROW
2B84; 2B86 # LEFTWARDS TRIANGLE-HEADED PAIRED ARROWS
2B86; 2B84 # RIGHTWARDS TRIANGLE-HEADED PAIRED ARROWS
2B88; 2B8A # LEFTWARDS BLACK CIRCLED WHITE ARROW
2B8A; 2B88 # RIGHTWARDS BLACK CIRCLED WHITE ARROW
2B90; 2B91 # RETURN LEFT
2B91; 2B90 # RETURN RIGHT
2B92; 2B93 # NEWLINE LEFT
2B93; 2B92 # NEWLINE RIGHT
2B95; 2B05 # RIGHTWARDS BLACK ARROW
2B98; 2B9A # THREE-D TOP-LIGHTED LEFTWARDS EQUILATERAL ARROWHEAD
2B9A; 2B98 # THREE-D TOP-LIGHTED RIGHTWARDS EQUILATERAL ARROWHEAD
2B9C; 2B9E # BLACK LEFTWARDS EQUILATERAL ARROWHEAD
2B9E; 2B9C # BLACK RIGHTWARDS EQUILATERAL ARROWHEAD
```

```
2BA0; 2BA1 # DOWNWARDS TRIANGLE-HEADED ARROW WITH LONG TIP LEFTWARDS
2BA1; 2BA0 # DOWNWARDS TRIANGLE-HEADED ARROW WITH LONG TIP RIGHTWARDS
2BA2; 2BA3 # UPWARDS TRIANGLE-HEADED ARROW WITH LONG TIP LEFTWARDS
2BA3; 2BA2 # UPWARDS TRIANGLE-HEADED ARROW WITH LONG TIP RIGHTWARDS
2BA4; 2BA5 # LEFTWARDS TRIANGLE-HEADED ARROW WITH LONG TIP UPWARDS
2BA5; 2BA4 # RIGHTWARDS TRIANGLE-HEADED ARROW WITH LONG TIP UPWARDS
2BA6; 2BA7 # LEFTWARDS TRIANGLE-HEADED ARROW WITH LONG TIP DOWNWARDS
2BA7; 2BA6 # RIGHTWARDS TRIANGLE-HEADED ARROW WITH LONG TIP DOWNWARDS
2BA8; 2BA9 # BLACK CURVED DOWNWARDS AND LEFTWARDS ARROW
2BA9; 2BA8 # BLACK CURVED DOWNWARDS AND RIGHTWARDS ARROW
2BAA; 2BAB # BLACK CURVED UPWARDS AND LEFTWARDS ARROW
2BAB; 2BAA # BLACK CURVED UPWARDS AND RIGHTWARDS ARROW
2BAC; 2BAD # BLACK CURVED LEFTWARDS AND UPWARDS ARROW
2BAD; 2BAC # BLACK CURVED RIGHTWARDS AND UPWARDS ARROW
2BAE; 2BAF # BLACK CURVED LEFTWARDS AND DOWNWARDS ARROW
2BAF; 2BAE # BLACK CURVED RIGHTWARDS AND DOWNWARDS ARROW
2BB0; 2BB1 # RIBBON ARROW DOWN LEFT
2BB1; 2BB0 # RIBBON ARROW DOWN RIGHT
2BB2; 2BB3 # RIBBON ARROW UP LEFT
2BB3; 2BB2 # RIBBON ARROW UP RIGHT
2BB4; 2BB5 # RIBBON ARROW LEFT UP
2BB5; 2BB4 # RIBBON ARROW RIGHT UP
2BB6; 2BB7 # RIBBON ARROW LEFT DOWN
2BB7; 2BB6 # RIBBON ARROW RIGHT DOWN
2BEC; 2BEE # LEFTWARDS TWO-HEADED ARROW WITH TRIANGLE ARROWHEADS
2BEE; 2BEC # RIGHTWARDS TWO-HEADED ARROW WITH TRIANGLE ARROWHEADS
2E10; 2E11 # FORKED PARAGRAPHOS
2E11; 2E10 # REVERSED FORKED PARAGRAPHOS
2E2E; 003F # REVERSED QUESTION MARK
2E36; 2E37 # DAGGER WITH LEFT GUARD
2E37; 2E36 # DAGGER WITH RIGHT GUARD
2E41; 002C # REVERSED COMMA
301D; 301E # REVERSED DOUBLE PRIME QUOTATION MARK
301E; 301D # DOUBLE PRIME QUOTATION MARK
A9C1; A9C2 # JAVANESE LEFT RERENGGAN
A9C2; A9C1 # JAVANESE RIGHT RERENGGAN
FD3E; FD3F # ORNATE LEFT PARENTHESIS
FD3F; FD3E # ORNATE RIGHT PARENTHESIS
FF0F; FF3C # FULLWIDTH SOLIDUS
FF3C; FF0F # FULLWIDTH REVERSE SOLIDUS
FFE9; FFEB # HALFWIDTH LEFTWARDS ARROW
FFEB; FFE9 # HALFWIDTH RIGHTWARDS ARROW

1D102; 1D103 # MUSICAL SYMBOL FINAL BARLINE
1D103; 1D102 # MUSICAL SYMBOL REVERSE FINAL BARLINE
1D106; 1D107 # MUSICAL SYMBOL LEFT REPEAT SIGN
1D107; 1D106 # MUSICAL SYMBOL RIGHT REPEAT SIGN
1EF06; 1EF07 # CHINESE LEFT CURLY BRACKET  # www.unicode.org/L2/L2017/17219-chinese-
       math.pdf
1EF07; 1EF06 # CHINESE RIGHT CURLY BRACKET
1F448; 1F449 # WHITE LEFT POINTING BACKHAND INDEX
1F449; 1F448 # WHITE RIGHT POINTING BACKHAND INDEX

1F500---1F504 more arrows; to do: check left-rightness

1F508; 1F568 # SPEAKER
1F509; 1F569 # SPEAKER WITH ONE SOUND WAVE
1F50A; 1F56A # SPEAKER WITH THREE SOUND WAVES
1F568; 1F508 # RIGHT SPEAKER
1F569; 1F509 # RIGHT SPEAKER WITH ONE SOUND WAVE
1F56A; 1F50A # RIGHT SPEAKER WITH THREE SOUND WAVES
1F57B; 1F57D # LEFT HAND TELEPHONE RECEIVER
1F57D; 1F57B # RIGHT HAND TELEPHONE RECEIVER
1F598; 1F599 # SIDEWAYS WHITE LEFT POINTING INDEX
1F599; 1F598 # SIDEWAYS WHITE RIGHT POINTING INDEX
1F59A; 1F59B # SIDEWAYS BLACK LEFT POINTING INDEX
1F59B; 1F59A # SIDEWAYS BLACK RIGHT POINTING INDEX
1F59C; 1F59D # BLACK LEFT POINTING BACKHAND INDEX
1F59D; 1F59C # BLACK RIGHT POINTING BACKHAND INDEX
1F5E6; 1F5E7 # THREE RAYS LEFT
1F5E7; 1F5E6 # THREE RAYS RIGHT

1F650---1F667 (leafs, emoji?)
1F66C,1F66E (rocket emoji...)
1F67C,1F76D (heavy (reverse) solidus)

1F780; 1F782 # BLACK LEFT-POINTING ISOSCELES RIGHT TRIANGLE
```

```
1F782; 1F780 # BLACK RIGHT-POINTING ISOSCELES RIGHT TRIANGLE
1F800; 1F802 # LEFTWARDS ARROW WITH SMALL TRIANGLE ARROWHEAD
1F802; 1F800 # RIGHTWARDS ARROW WITH SMALL TRIANGLE ARROWHEAD
1F804; 1F806 # LEFTWARDS ARROW WITH MEDIUM TRIANGLE ARROWHEAD
1F806; 1F804 # RIGHTWARDS ARROW WITH MEDIUM TRIANGLE ARROWHEAD
1F808; 1F80A # LEFTWARDS ARROW WITH LARGE TRIANGLE ARROWHEAD
1F80A; 1F808 # RIGHTWARDS ARROW WITH LARGE TRIANGLE ARROWHEAD
1F810; 1F812 # LEFTWARDS ARROW WITH SMALL EQUILATERAL ARROWHEAD
1F812; 1F810 # RIGHTWARDS ARROW WITH SMALL EQUILATERAL ARROWHEAD
1F814; 1F816 # LEFTWARDS ARROW WITH EQUILATERAL ARROWHEAD
1F816; 1F814 # RIGHTWARDS ARROW WITH EQUILATERAL ARROWHEAD
1F818; 1F81A # HEAVY LEFTWARDS ARROW WITH EQUILATERAL ARROWHEAD
1F81A; 1F818 # HEAVY RIGHTWARDS ARROW WITH EQUILATERAL ARROWHEAD
1F81C; 1F81E # HEAVY LEFTWARDS ARROW WITH LARGE EQUILATERAL ARROWHEAD
1F81E; 1F81C # HEAVY RIGHTWARDS ARROW WITH LARGE EQUILATERAL ARROWHEAD
1F820; 1F822 # LEFTWARDS TRIANGLE-HEADED ARROW WITH NARROW SHAFT
1F822; 1F820 # RIGHTWARDS TRIANGLE-HEADED ARROW WITH NARROW SHAFT
1F824; 1F826 # LEFTWARDS TRIANGLE-HEADED ARROW WITH MEDIUM SHAFT
1F826; 1F824 # RIGHTWARDS TRIANGLE-HEADED ARROW WITH MEDIUM SHAFT
1F828; 1F82A # LEFTWARDS TRIANGLE-HEADED ARROW WITH BOLD SHAFT
1F82A; 1F828 # RIGHTWARDS TRIANGLE-HEADED ARROW WITH BOLD SHAFT
1F82C; 1F82E # LEFTWARDS TRIANGLE-HEADED ARROW WITH HEAVY SHAFT
1F82E; 1F82C # RIGHTWARDS TRIANGLE-HEADED ARROW WITH HEAVY SHAFT
1F830; 1F832 # LEFTWARDS TRIANGLE-HEADED ARROW WITH VERY HEAVY SHAFT
1F832; 1F830 # RIGHTWARDS TRIANGLE-HEADED ARROW WITH VERY HEAVY SHAFT
1F834; 1F836 # LEFTWARDS FINGER-POST ARROW
1F836; 1F834 # RIGHTWARDS FINGER-POST ARROW
1F838; 1F83A # LEFTWARDS SQUARED ARROW
1F83A; 1F838 # RIGHTWARDS SQUARED ARROW
1F83C; 1F83E # LEFTWARDS COMPRESSED ARROW
1F83E; 1F83C # RIGHTWARDS COMPRESSED ARROW
1F840; 1F842 # LEFTWARDS HEAVY COMPRESSED ARROW
1F842; 1F840 # RIGHTWARDS HEAVY COMPRESSED ARROW
1F844; 1F846 # LEFTWARDS HEAVY ARROW
1F846; 1F844 # RIGHTWARDS HEAVY ARROW

1F850; 1F852 # LEFTWARDS SANS-SERIF ARROW
1F852; 1F850 # RIGHTWARDS SANS-SERIF ARROW
1F854; 1F855 # NORTH WEST SANS-SERIF ARROW
1F855; 1F854 # NORTH EAST SANS-SERIF ARROW
1F856; 1F857 # SOUTH EAST SANS-SERIF ARROW
1F857; 1F856 # SOUTH WEST SANS-SERIF ARROW

1F860; 1F862 # WIDE-HEADED LEFTWARDS LIGHT BARB ARROW
1F862; 1F860 # WIDE-HEADED RIGHTWARDS LIGHT BARB ARROW
1F864; 1F865 # WIDE-HEADED NORTH WEST LIGHT BARB ARROW
1F865; 1F864 # WIDE-HEADED NORTH EAST LIGHT BARB ARROW
1F866; 1F867 # WIDE-HEADED SOUTH EAST LIGHT BARB ARROW
1F867; 1F866 # WIDE-HEADED SOUTH WEST LIGHT BARB ARROW
1F868; 1F86A # WIDE-HEADED LEFTWARDS BARB ARROW
1F86A; 1F868 # WIDE-HEADED RIGHTWARDS BARB ARROW
1F86C; 1F86D # WIDE-HEADED NORTH WEST BARB ARROW
1F86D; 1F86C # WIDE-HEADED NORTH EAST BARB ARROW
1F86E; 1F86F # WIDE-HEADED SOUTH EAST BARB ARROW
1F86F; 1F86E # WIDE-HEADED SOUTH WEST BARB ARROW
1F870; 1F872 # WIDE-HEADED LEFTWARDS MEDIUM BARB ARROW
1F872; 1F870 # WIDE-HEADED RIGHTWARDS MEDIUM BARB ARROW
1F874; 1F875 # WIDE-HEADED NORTH WEST MEDIUM BARB ARROW
1F875; 1F874 # WIDE-HEADED NORTH EAST MEDIUM BARB ARROW
1F876; 1F877 # WIDE-HEADED SOUTH EAST MEDIUM BARB ARROW
1F877; 1F876 # WIDE-HEADED SOUTH WEST MEDIUM BARB ARROW
1F878; 1F87A # WIDE-HEADED LEFTWARDS HEAVY BARB ARROW
1F87A; 1F878 # WIDE-HEADED RIGHTWARDS HEAVY BARB ARROW
1F87C; 1F87D # WIDE-HEADED NORTH WEST HEAVY BARB ARROW
1F87D; 1F87C # WIDE-HEADED NORTH EAST HEAVY BARB ARROW
1F87E; 1F87F # WIDE-HEADED SOUTH EAST HEAVY BARB ARROW
1F87F; 1F87E # WIDE-HEADED SOUTH WEST HEAVY BARB ARROW
1F880; 1F882 # WIDE-HEADED LEFTWARDS VERY HEAVY BARB ARROW
1F882; 1F880 # WIDE-HEADED RIGHTWARDS VERY HEAVY BARB ARROW
1F884; 1F885 # WIDE-HEADED NORTH WEST VERY HEAVY BARB ARROW
1F885; 1F884 # WIDE-HEADED NORTH EAST VERY HEAVY BARB ARROW
1F886; 1F887 # WIDE-HEADED SOUTH EAST VERY HEAVY BARB ARROW
1F887; 1F886 # WIDE-HEADED SOUTH WEST VERY HEAVY BARB ARROW
1F890; 1F892 # LEFTWARDS TRIANGLE ARROWHEAD
1F892; 1F890 # RIGHTWARDS TRIANGLE ARROWHEAD
1F894; 1F896 # LEFTWARDS WHITE ARROW WITHIN TRIANGLE ARROWHEAD
1F896; 1F894 # RIGHTWARDS WHITE ARROW WITHIN TRIANGLE ARROWHEAD
```

```
1F898; 1F89A # LEFTWARDS ARROW WITH NOTCHED TAIL
1F89A; 1F898 # RIGHTWARDS ARROW WITH NOTCHED TAIL
1F8A0; 1F8A1 # LEFTWARDS BOTTOM-SHADED WHITE ARROW
1F8A1; 1F8A0 # RIGHTWARDS BOTTOM SHADED WHITE ARROW
1F8A2; 1F8A3 # LEFTWARDS TOP SHADED WHITE ARROW
1F8A3; 1F8A2 # RIGHTWARDS TOP SHADED WHITE ARROW
1F8A4; 1F8A5 # LEFTWARDS LEFT-SHADED WHITE ARROW
1F8A5; 1F8A4 # RIGHTWARDS RIGHT-SHADED WHITE ARROW
1F8A6; 1F8A7 # LEFTWARDS RIGHT-SHADED WHITE ARROW
1F8A7; 1F8A6 # RIGHTWARDS LEFT-SHADED WHITE ARROW
1F8A8; 1F8A9 # LEFTWARDS BACK-TILTED SHADOWED WHITE ARROW
1F8A9; 1F8A8 # RIGHTWARDS BACK-TILTED SHADOWED WHITE ARROW
1F8AA; 1F8AB # LEFTWARDS FRONT-TILTED SHADOWED WHITE ARROW
1F8AB; 1F8AA # RIGHTWARDS FRONT-TILTED SHADOWED WHITE ARROW

# The following characters have no appropriate mirroring character,
# but are otherwise conceivably mirrorable.

# 0026; AMPERSAND
# 0606; ARABIC-INDIC CUBE ROOT
# 0607; ARABIC-INDIC FOURTH ROOT
# 2052; COMMERCIAL MINUS SIGN
# 2057; QUADRUPLE PRIME
# 214B; TURNED AMPERSAND
# 21AF; DOWNWARDS ZIGZAG ARROW      ; arrow to the south-west
# 21B4; RIGHTWARDS ARROW WITH CORNER DOWNWARDS
# 21B5; DOWNWARDS ARROW WITH CORNER LEFTWARDS
# 21B9; LEFTWARDS ARROW TO BAR OVER RIGHTWARDS ARROW TO BAR
# 2319; TURNED NOT SIGN             ; 'line marker'
# 237C; RIGHT ANGLE WITH DOWNWARDS ZIGZAG ARROW ; arrow to the south-west
# 238B; BROKEN CIRCLE WITH NORTHWEST ARROW
# 23B2; SUMMATION TOP
# 23B3; SUMMATION BOTTOM
# 23EF; BLACK RIGHT-POINTING TRIANGLE WITH DOUBLE VERTICAL BAR
# 2794; HEAVY WIDE-HEADED RIGHTWARDS ARROW
# 2798; HEAVY SOUTH EAST ARROW
# 2799; HEAVY RIGHTWARDS ARROW
# 279A; HEAVY NORTH EAST ARROW
# 279B; DRAFTING POINT RIGHTWARDS ARROW
# 279C; HEAVY ROUND-TIPPED RIGHTWARDS ARROW
# 279D; TRIANGLE-HEADED RIGHTWARDS ARROW
# 279E; HEAVY TRIANGLE-HEADED RIGHTWARDS ARROW
# 279F; DASHED TRIANGLE-HEADED RIGHTWARDS ARROW
# 279F; HEAVY DASHED TRIANGLE-HEADED RIGHTWARDS ARROW
# 27A1; BLACK RIGHTWARDS ARROW
# 27A2; THREE-D TOP-LIGHTED RIGHTWARDS ARROWHEAD
# 27A3; THREE-D BOTTOM-LIGHTED RIGHTWARDS ARROWHEAD
# 27A4; BLACK RIGHTWARDS ARROWHEAD
# 27A5; HEAVY BLACK CURVED DOWNWARDS AND RIGHTWARDS ARROW
# 27A6; HEAVY BLACK CURVED UPWARDS AND RIGHTWARDS ARROW
# 27A7; SQUAT BLACK RIGHTWARDS ARROW
# 27A8; HEAVY CONCAVE-POINTED BLACK RIGHTWARDS ARROW
# 27A9; RIGHT-SHADED WHITE RIGHTWARDS ARROW
# 27AA; LEFT-SHADED WHITE RIGHTWARDS ARROW
# 27AB; BACK-TILTED SHADOWED WHITE RIGHTWARDS ARROW
# 27AC; FRONT-TILTED SHADOWED WHITE RIGHTWARDS ARROW
# 27AD; HEAVY LOWER RIGHT-SHADOWED WHITE RIGHTWARDS ARROW
# 27AE; HEAVY UPPER RIGHT-SHADOWED WHITE RIGHTWARDS ARROW
# 27AF; NOTCHED LOWER RIGHT-SHADOWED WHITE RIGHTWARDS ARROW
# 27B1; NOTCHED UPPER RIGHT-SHADOWED WHITE RIGHTWARDS ARROW
# 27B2; CIRCLED HEAVY WHITE RIGHTWARDS ARROW
# 27B3; WHITE-FEATHERED RIGHTWARDS ARROW
# 27B4; BLACK-FEATHERED SOUTH EAST ARROW
# 27B5; BLACK-FEATHERED RIGHTWARDS ARROW
# 27B6; BLACK-FEATHERED NORTH EAST ARROW
# 27B7; HEAVY BLACK-FEATHERED SOUTH EAST ARROW
# 27B8; HEAVY BLACK-FEATHERED RIGHTWARDS ARROW
# 27B9; HEAVY BLACK-FEATHERED NORTH EAST ARROW
# 27BA; TEARDROP-BARBED RIGHTWARDS ARROW
# 27BB; HEAVY TEARDROP-SHANKED RIGHTWARDS ARROW
# 27BC; WEDGE-TAILED RIGHTWARDS ARROW
# 27BD; HEAVY WEDGE-TAILED RIGHTWARDS ARROW
# 27BE; OPEN-OUTLINED RIGHTWARDS ARROW
# 292D; SOUTH EAST ARROW CROSSING NORTH EAST ARROW
# 292E; NORTH EAST ARROW CROSSING SOUTH EAST ARROW
# 292F; FALLING DIAGONAL CROSSING NORTH EAST ARROW
# 2930; RISING DIAGONAL CROSSING SOUTH EAST ARROW
# 2934; ARROW POINTING RIGHTWARDS THEN CURVING UPWARDS
# 2935; ARROW POINTING RIGHTWARDS THEN CURVING DOWNWARDS
# 293A; TOP ARC ANTICLOCKWISE ARROW
# 293B; BOTTOM ARC ANTICLOCKWISE ARROW
# 293C; TOP ARC CLOCKWISE ARROW WITH MINUS
# 293D; TOP ARC ANTICLOCKWISE ARROW WITH PLUS
# 2944; SHORT RIGHTWARDS ARROW ABOVE LEFTWARDS ARROW
# 2970; RIGHT DOUBLE ARROW WITH ROUNDED HEAD
# 299A; VERTICAL ZIGZAG LINE
# 29BC; CIRCLED ANTICLOCKWISE-ROTATED DIVISION SIGN
# 2B4D; DOWNWARDS TRIANGLE-HEADED ZIGZAG ARROW
# 2B7E; HORIZONTAL TAB KEY; leftwards triangle-headed arrow to bar over rightwards
       triangle-headed arrow to bar
# 2B7F; VERTICAL TAB KEY; downwards triangle-headed arrow to bar left of triangle-headed
       upwards arrow to bar
# 2B8C; ANTICLOCKWISE TRIANGLE-HEADED RIGHT U-SHAPED ARROW
# 2B8D; ANTICLOCKWISE TRIANGLE-HEADED BOTTOM U-SHAPED ARROW
# 2B8E; ANTICLOCKWISE TRIANGLE-HEADED LEFT U-SHAPED ARROW
# 2B8F; ANTICLOCKWISE TRIANGLE-HEADED TOP U-SHAPED ARROW
# 2B94; FOUR CORNER ARROWS CIRCLING ANTICLOCKWISE
# 2B99; THREE-D RIGHT-LIGHTED UPWARDS EQUILATERAL ARROWHEAD
# 2B9B; THREE-D LEFT-LIGHTED DOWNWARDS EQUILATERAL ARROWHEAD
#
# 1F10E; CIRCLED ANTICLOCKWISE ARROW
# 1F5D8; CLOCKWISE RIGHT AND LEFT SEMICIRCLE ARROWS
# 1F8B0; ARROW POINTING UPWARDS THEN NORTH WEST
# 1F8B1; ARROW POINTING RIGHTWARDS THEN CURVING SOUTH WEST

# EOF
```

Appendix D – MMS mirroring

The following symbols can be mirrored by **MMS** (or the corresponding attribute in the HTML/XML/SVG compatible version), unless otherwise excluded (like the “MATHEMATICAL” characters). Some are still recommended against, like ARABIC-INDIC CUBE ROOT, which should be composed, or SUMMATION TOP, which is a glyph component for a stretched N-ARY SUMMATION symbol. Note that negation stroke is not mirrored by MMS, nor is integration direction (clockwise, anticlockwise).

&	0026	AMPERSAND
؆	0606	~~ARABIC-INDIC CUBE ROOT~~ ; use instead **MMS √ SUPR ٣** ; mirror: **٣ SUPL √**
؇	0607	~~ARABIC-INDIC FOURTH ROOT~~ ; use instead **MMS √ SUPR ٤** ; mirror: **٤ SUPL √**
⁒	2052	COMMERCIAL MINUS SIGN
⁗	2057	QUADRUPLE PRIME
⅀	2140	~~DOUBLE-STRUCK N-ARY SUMMATION~~ ; use instead **SCI I ∑** ; mirror: **SCI I MMS ∑**
⅋	214B	TURNED AMPERSAND
↯	21AF	DOWNWARDS ZIGZAG ARROW ; arrow to the south-west
↴	21B4	RIGHTWARDS ARROW WITH CORNER DOWNWARDS
↵	21B5	DOWNWARDS ARROW WITH CORNER LEFTWARDS
↹	21B9	LEFTWARDS ARROW TO BAR OVER RIGHTWARDS ARROW TO BAR
∁	2201	COMPLEMENT
∂	2202	PARTIAL DIFFERENTIAL ; default style is **SCI A**, upright serifed normal weight, for italic use **SCI C**
∃	2203	THERE EXISTS
∄	2204	THERE DOES NOT EXIST ; the negation slash does not mirror
∑	2211	N-ARY SUMMATION
∖	2216	SET MINUS; actually SMALL SET MINUS from TeX \smallsetminus, ordinary set minus is \, U+005C
√	221A	SQUARE ROOT ; actually ROOT SYMBOL, “square” is just default
∛	221B	~~CUBE ROOT~~ ; use instead **3 SUPL √** ; mirror: **MMS √ SUPR 3**
∜	221C	~~FOURTH ROOT~~ ; use instead **4 SUPL √** ; mirror: **MMS √ SUPR 4**
∝	221D	PROPORTIONAL TO
∦	2226	~~NOT PARALLEL TO~~ ; negation slash does not mirror, so this one has itself as mirror
∫	222B	INTEGRAL
∬	222C	DOUBLE INTEGRAL
∭	222D	TRIPLE INTEGRAL
∮	222E	CONTOUR INTEGRAL
∯	222F	SURFACE INTEGRAL
∰	2230	VOLUME INTEGRAL
∱	2231	CLOCKWISE INTEGRAL ; the arrow part is not mirrored by **MMS** (cmp. U+293A, **MMS** ⤺)
∲	2232	CLOCKWISE CONTOUR INTEGRAL ; the arrow part is not mirrored by **MMS** (cmp. U+2941, ⥁, turned 90° clockwise)
∳	2233	ANTICLOCKWISE CONTOUR INTEGRAL ; the arrow part is not mirrored by **MMS** (cmp. U+2940, ⥀, turned 90° clockwise)
∹	2239	EXCESS
∻	223B	HOMOTHETIC
∾	223E	INVERTED LAZY S
∿	223F	SINE WAVE
≀	2240	WREATH PRODUCT

≁	2241	NOT TILDE ; prefer to mirror as <∽,U+0338>
≂	2242	MINUS TILDE
≄	2244	NOT ASYMPTOTICALLY EQUAL TO ; prefer to mirror as <⋍,U+0338>
≆	2246	APPROXIMATELY BUT NOT ACTUALLY EQUAL TO
≇	2247	NEITHER APPROXIMATELY NOR ACTUALLY EQUAL TO ; m: <≌,,U+0338> (for ≌, "lazy s" on top is not a (free) glyph variant)
≈	2248	ALMOST EQUAL TO ; unclear if this should mirror in RTL math
≉	2249	NOT ALMOST EQUAL TO ; unclear if this should mirror
≊	224A	ALMOST EQUAL OR EQUAL TO ; unclear if this should mirror
≋	224B	TRIPLE TILDE ; unclear if this should mirror in RTL math
≟	225F	QUESTIONED EQUAL TO ; prefer to use **= ABV ?**
≠	2260	~~NOT EQUAL TO~~ ; negation slash does not mirror, so this one has itself as mirror
≢	2262	~~NOT IDENTICAL TO~~ ; negation slash does not mirror, so this one has itself as mirror
⊌	228C	MULTISET ; unclear if the arrow part should mirror by MMS
⊧	22A7	MODELS
⊪	22AA	TRIPLE VERTICAL BAR RIGHT TURNSTILE
⊬	22AC	DOES NOT PROVE ; prefer to mirror as <⊣,U+0338>
⊭	22AD	NOT TRUE ; prefer to mirror as <⫤,U+0338>
⊮	22AE	DOES NOT FORCE ; prefer to mirror as <⫣,U+0338>
⊯	22AF	NEGATED DOUBLE VERTICAL BAR DOUBLE RIGHT TURNSTILE ; prefer to mirror as <⫥,U+0338>
⊾	22BE	RIGHT ANGLE WITH ARC
⊿	22BF	RIGHT TRIANGLE
⋵	22F5	ELEMENT OF WITH DOT ABOVE ; prefer to mirror as <∋,U+0307>
⋸	22F8	ELEMENT OF WITH UNDERBAR ; prefer to mirror as <∋,U+0331>
⋹	22F9	ELEMENT OF WITH TWO HORIZONTAL STROKES
⋿	22FF	Z NOTATION BAG MEMBERSHIP
⌙	2319	TURNED NOT SIGN ; 'line marker'
⌠	2320	~~TOP HALF INTEGRAL~~ ; intended for internal composition of stretched integral sign, not for direct use in a math expression (internally MMS-ed)
⌡	2321	~~BOTTOM HALF INTEGRAL~~ ; intended for internal composition of stretched integral sign, not for direct use in a math expression (internally MMS-ed)
⍼	237C	RIGHT ANGLE WITH DOWNWARDS ZIGZAG ARROW ; arrow to the south-west
⎋	238B	BROKEN CIRCLE WITH NORTHWEST ARROW
⎲	23B2	~~SUMMATION TOP~~ ; intended for internal composition of stretched sum sign, not for direct use in a math expression (internally MMS-ed)
⎳	23B3	~~SUMMATION BOTTOM~~ ; intended for internal composition of stretched sum sign, not for direct use in a math expression (internally MMS-ed)
⏯	23EF	BLACK RIGHT-POINTING TRIANGLE WITH DOUBLE VERTICAL BAR
➔	2794	HEAVY WIDE-HEADED RIGHTWARDS ARROW
➘	2798	HEAVY SOUTH EAST ARROW
➙	2799	HEAVY RIGHTWARDS ARROW
➚	279A	HEAVY NORTH EAST ARROW
➛	279B	DRAFTING POINT RIGHTWARDS ARROW
➜	279C	HEAVY ROUND-TIPPED RIGHTWARDS ARROW
➝	279D	TRIANGLE-HEADED RIGHTWARDS ARROW
➞	279E	HEAVY TRIANGLE-HEADED RIGHTWARDS ARROW
➟	279F	DASHED TRIANGLE-HEADED RIGHTWARDS ARROW
➠	279F	HEAVY DASHED TRIANGLE-HEADED RIGHTWARDS ARROW
➡	27A1	BLACK RIGHTWARDS ARROW
➢	27A2	THREE-D TOP-LIGHTED RIGHTWARDS ARROWHEAD
➣	27A3	THREE-D BOTTOM-LIGHTED RIGHTWARDS ARROWHEAD

➤	27A4	BLACK RIGHTWARDS ARROWHEAD
➥	27A5	HEAVY BLACK CURVED DOWNWARDS AND RIGHTWARDS ARROW
➦	27A6	HEAVY BLACK CURVED UPWARDS AND RIGHTWARDS ARROW
➧	27A7	SQUAT BLACK RIGHTWARDS ARROW
➨	27A8	HEAVY CONCAVE-POINTED BLACK RIGHTWARDS ARROW
➩	27A9	RIGHT-SHADED WHITE RIGHTWARDS ARROW
➪	27AA	LEFT-SHADED WHITE RIGHTWARDS ARROW
➫	27AB	BACK-TILTED SHADOWED WHITE RIGHTWARDS ARROW
➬	27AC	FRONT-TILTED SHADOWED WHITE RIGHTWARDS ARROW
➭	27AD	HEAVY LOWER RIGHT-SHADOWED WHITE RIGHTWARDS ARROW
➮	27AE	HEAVY UPPER RIGHT-SHADOWED WHITE RIGHTWARDS ARROW
➯	27AF	NOTCHED LOWER RIGHT-SHADOWED WHITE RIGHTWARDS ARROW
➱	27B1	NOTCHED UPPER RIGHT-SHADOWED WHITE RIGHTWARDS ARROW
➲	27B2	CIRCLED HEAVY WHITE RIGHTWARDS ARROW
➳	27B3	WHITE-FEATHERED RIGHTWARDS ARROW
➴	27B4	BLACK-FEATHERED SOUTH EAST ARROW
➵	27B5	BLACK-FEATHERED RIGHTWARDS ARROW
➶	27B6	BLACK-FEATHERED NORTH EAST ARROW
➷	27B7	HEAVY BLACK-FEATHERED SOUTH EAST ARROW
➸	27B8	HEAVY BLACK-FEATHERED RIGHTWARDS ARROW
➹	27B9	HEAVY BLACK-FEATHERED NORTH EAST ARROW
➺	27BA	TEARDROP-BARBED RIGHTWARDS ARROW
➻	27BB	HEAVY TEARDROP-SHANKED RIGHTWARDS ARROW
➼	27BC	WEDGE-TAILED RIGHTWARDS ARROW
➽	27BD	HEAVY WEDGE-TAILED RIGHTWARDS ARROW
➾	27BE	OPEN-OUTLINED RIGHTWARDS ARROW
⟀	27C0	THREE DIMENSIONAL ANGLE
⟌	27CC	~~LONG DIVISION~~ ; regarded as control and cannot be “MMS-ed” ; in addition, graphical appearance varies by tradition ; and it is part of a computation notation, not a math operator
⟓	27D3	LOWER RIGHT CORNER WITH DOT
⟔	27D4	UPPER LEFT CORNER WITH DOT
⤭	292D	SOUTH EAST ARROW CROSSING NORTH EAST ARROW
⤮	292E	NORTH EAST ARROW CROSSING SOUTH EAST ARROW
⤯	292F	FALLING DIAGONAL CROSSING NORTH EAST ARROW
⤰	2930	RISING DIAGONAL CROSSING SOUTH EAST ARROW
⤴	2934	ARROW POINTING RIGHTWARDS THEN CURVING UPWARDS
⤵	2935	ARROW POINTING RIGHTWARDS THEN CURVING DOWNWARDS
⤺	293A	TOP ARC ANTICLOCKWISE ARROW
⤻	293B	BOTTOM ARC ANTICLOCKWISE ARROW
⤼	293C	TOP ARC CLOCKWISE ARROW WITH MINUS
⤽	293D	TOP ARC ANTICLOCKWISE ARROW WITH PLUS
⥄	2944	SHORT RIGHTWARDS ARROW ABOVE LEFTWARDS ARROW
⥰	2970	RIGHT DOUBLE ARROW WITH ROUNDED HEAD
⦚	299A	VERTICAL ZIGZAG LINE
⦜	299C	RIGHT ANGLE VARIANT WITH SQUARE
⦝	299D	MEASURED RIGHT ANGLE WITH DOT
⦞	299E	ANGLE WITH S INSIDE
⦟	299F	ACUTE ANGLE
⦢	29A2	TURNED ANGLE
⦦	29A6	OBLIQUE ANGLE OPENING UP
⦧	29A7	OBLIQUE ANGLE OPENING DOWN
⦼	29BC	CIRCLED ANTICLOCKWISE-ROTATED DIVISION SIGN
⧂	29C2	CIRCLE WITH SMALL CIRCLE TO THE RIGHT
⧃	29C3	CIRCLE WITH TWO HORIZONTAL STROKES TO THE RIGHT
⧉	29C9	TWO JOINED SQUARES
⧎	29CE	RIGHT TRIANGLE ABOVE LEFT TRIANGLE
⧜	29DC	INCOMPLETE INFINITY

⧡	29E1	INCREASES AS
⧣	29E3	EQUALS SIGN AND SLANTED PARALLEL
⧤	29E4	EQUALS SIGN AND SLANTED PARALLEL WITH TILDE ABOVE
⧥	29E5	IDENTICAL TO AND SLANTED PARALLEL
⧴	29F4	RULE-DELAYED
⧶	29F6	SOLIDUS WITH OVERBAR
⧷	29F7	REVERSE SOLIDUS WITH HORIZONTAL STROKE
⨊	2A0A	MODULO TWO SUM
⨋	2A0B	SUMMATION WITH INTEGRAL ; both parts are mirrored by **MMS**
⨌	2A0C	QUADRUPLE INTEGRAL OPERATOR
⨍	2A0D	FINITE PART INTEGRAL
⨎	2A0E	INTEGRAL WITH DOUBLE STROKE
⨏	2A0F	INTEGRAL AVERAGE WITH SLASH ; unclear if the slash should be mirrored by **MMS**
⨐	2A10	CIRCULATION FUNCTION ; the semicircle part should probably be seen as the letter "c" and thus not be mirrored by **MMS**
⨑	2A11	ANTICLOCKWISE INTEGRATION ; the arrow part is not mirrored by **MMS** (cmp. U+293B, ⤻)
⨒	2A12	LINE INTEGRATION WITH RECTANGULAR PATH AROUND POLE ; unclear if the middle part should be mirrored by **MMS**
⨓	2A13	LINE INTEGRATION WITH SEMICIRCULAR PATH AROUND POLE ; unclear if the middle part should be mirrored by **MMS**
⨔	2A14	LINE INTEGRATION NOT INCLUDING THE POLE ; unclear if the middle part should be mirrored by **MMS**
⨕	2A15	INTEGRAL AROUND A POINT OPERATOR
⨖	2A16	QUATERNION INTEGRAL OPERATOR
⨗	2A17	INTEGRAL WITH LEFTWARDS ARROW WITH HOOK ; unclear if the arrow part should be mirrored by **MMS** ; (cmp. U+21A9, ↩, but with the hook on the downside)
⨘	2A18	INTEGRAL WITH TIMES SIGN
⨙	2A19	INTEGRAL WITH INTERSECTION
⨚	2A1A	INTEGRAL WITH UNION
⨛	2A1B	INTEGRAL WITH OVERBAR
⨜	2A1C	INTEGRAL WITH UNDERBAR
⨞	2A1E	LARGE LEFT TRIANGLE OPERATOR
⨟	2A1F	Z NOTATION SCHEMA COMPOSITION (clearly larger than U+2A3E)
⨠	2A20	Z NOTATION SCHEMA PIPING
⨡	2A21	Z NOTATION SCHEMA PROJECTION (cmp. U+21BE, ↾, mirror: ↿)
⨤	2A24	PLUS SIGN WITH TILDE ABOVE
⨦	2A26	PLUS SIGN WITH TILDE BELOW
⨩	2A29	MINUS SIGN WITH COMMA ABOVE
⨾	2A3E	Z NOTATION RELATIONAL COMPOSITION (smaller than U+2A3E)
⩗	2A57	SLOPING LARGE OR
⩘	2A58	SLOPING LARGE AND
⩪	2A6A	TILDE OPERATOR WITH DOT ABOVE ; prefer to mirror as <∽,U+0307>
⩫	2A6B	TILDE OPERATOR WITH RISING DOTS
⩬	2A6C	SIMILAR MINUS SIMILAR
⩭	2A6D	CONGRUENT WITH DOT ABOVE ; prefer to mirror as <≌,U+0307>
⩯	2A6F	ALMOST EQUAL TO WITH CIRCUMFLEX ACCENT
⩰	2A70	APPROXIMATELY EQUAL OR EQUAL TO
⩳	2A73	EQUALS SIGN ABOVE TILDE OPERATOR
⩴	2A74	DOUBLE COLON EQUAL

⪣	2AA3	DOUBLE NESTED LESS-THAN WITH UNDERBAR ; prefer to mirror as <⪢,U+0331>
⫝̸	2ADC	FORKING ; negation slash does not mirror, so this one has itself as mirror
⫢	2AE2	VERTICAL BAR TRIPLE RIGHT TURNSTILE
⫦	2AE6	LONG DASH FROM LEFT MEMBER OF DOUBLE VERTICAL
⫳	2AF3	PARALLEL WITH TILDE OPERATOR
⫻	2AFB	TRIPLE SOLIDUS BINARY RELATION
⫽	2AFD	DOUBLE SOLIDUS OPERATOR
⭍	2B4D	DOWNWARDS TRIANGLE-HEADED ZIGZAG ARROW
⭾	2B7E	HORIZONTAL TAB KEY ; leftwards triangle-headed arrow to bar over rightwards triangle-headed arrow to bar, cmp. U+21B9
⭿	2B7F	VERTICAL TAB KEY ; downwards triangle-headed arrow to bar left of triangle-headed upwards arrow to bar
⮌	2B8C	ANTICLOCKWISE TRIANGLE-HEADED RIGHT U-SHAPED ARROW
⮍	2B8D	ANTICLOCKWISE TRIANGLE-HEADED BOTTOM U-SHAPED ARROW
⮎	2B8E	ANTICLOCKWISE TRIANGLE-HEADED LEFT U-SHAPED ARROW
⮏	2B8F	ANTICLOCKWISE TRIANGLE-HEADED TOP U-SHAPED ARROW
⮔	2B94	FOUR CORNER ARROWS CIRCLING ANTICLOCKWISE
⮙	2B99	THREE-D RIGHT-LIGHTED UPWARDS EQUILATERAL ARROWHEAD
⮛	2B9B	THREE-D LEFT-LIGHTED DOWNWARDS EQUILATERAL ARROWHEAD
𝛛	1D6DB	~~MATHEMATICAL BOLD PARTIAL DIFFERENTIAL~~ use instead **SCI B** *∂*
𝜕	1D715	~~MATHEMATICAL ITALIC PARTIAL DIFFERENTIAL~~ use instead **SCI C** *∂*
𝝏	1D74F	~~MATHEMATICAL BOLD ITALIC PARTIAL DIFFERENTIAL~~ use instead **SCI D** *∂*
𝞉	1D789	~~MATHEMATICAL SANS-SERIF BOLD PARTIAL DIFFERENTIAL~~ use instead **SCI F** *∂*
𝟃	1D7C3	~~MATHEMATICAL SANS-SERIF BOLD ITALIC PARTIAL DIFFERENTIAL~~ use instead **SCI H** *∂*
🄎	1F10E	CIRCLED ANTICLOCKWISE ARROW
🗘	1F5D8	CLOCKWISE RIGHT AND LEFT SEMICIRCLE ARROWS
🢰	1F8B0	ARROW POINTING UPWARDS THEN NORTH WEST
🢱	1F8B1	ARROW POINTING RIGHTWARDS THEN CURVING SOUTH WEST

Appendix E – Giving some C0/C1 and more math chars ok Unicode properties

Here is a summary of some of the Unicode character properties for some characters used in math expressions. In addition, default-ignorable code points are not allowed in math expressions.

c.p.	Name and comment (if char in C0/C1/Cf/*Zx* is *not* interpreted, it *shall not* be invisible)	g.c.	Bidi	LB
0001	START OF HEADING *(can be reused as HPGL label terminator)*	**Zp**	**B**	**NL**
0002	**START OF TEXT** *(reused for embedded text in math. expr.)*	**Zp**	**B**	**NL**
0003	**END OF TEXT** *(reused for embedded text in math. expr.)*	**Zp**	**B**	**NL**
0009	CHARACTER TABULATION *(matrix cell terminator, non-mx: SP)*	**Cf**	**S**	**BA**
000A	LINE FEED *(matrix row separator, non-mx: SP)*	**Zp**	**B**	**LF**
000B	LINE TABULATION *(matrix row separator, non-mx: SP)*	**Zl**	**S**	**LF**
000C	FORM FEED *(matrix row separator, non-mx: SP)*	**Zl**	**S**	**LF**
000D	CARRIAGE RETURN *(matrix row separator, non-mx: SP)*	**Zp**	**B**	**CR**
0016	SYNCHRONOUS IDLE *(can be reused for ignored SP in SME…EME (as well as in ISn-based data formats for non-data SP))*	**Zs**	**L**	**AI**
001A	SUBSTITUTE *(origin for REPLACEMENT CHARACTER)*	**So**	**ON**	**AI**
001B	ESCAPE *(may be used for character ref. to CSI and SCI)*	**Cf**	**BN**	**CB**
002F	SOLIDUS *(math operator (Sm); division [no special layout])*	**Po**	**ON**	**SY**
005C	REVERSE SOLIDUS *(math operator (Sm); set difference)*	**Po**	**ON**	**SY**
0080	***START OF MATH EXPRESSION*** *(SME)*	**Zp**	**B**	**NL**
0081	***END OF MATH EXPRESSION*** *(EME)*	**Zp**	**B**	**NL**
0084	***MATH HORIZONTAL DIVISION*** *(MHD) (note that RTL will never change the order of operands for MHD in any way!)*	**Sm**	**S**	**NL**
0085	NEXT LINE (NEL) *(matrix row separator, non-mx: SP)*	**Zp**	**B**	**NL**
0089	CHARACTER TABULATION WITH JUSTIFICATION (HTJ) *(matrix cell terminator, non-mx: SP)*	**Cf**	**S**	**BA**
008B	PARTIAL LINE FORWARD (PLD) *(math matrix layout tweak)*	**Cf**	**S**	**CM**
008C	PARTIAL LINE BACKWARD (PLU) *(math matrix layout tweak)*	**Cf**	**S**	**CM**
0099	***MATH MIRROR SYMBOL*** *(MMS)*	**Cf**	**BN**	**CB**
009A	SINGLE CHARACTER INTRODUCER (SCI) *(new math controls)*	**Cf**	**BN**	**CB**
009B	CONTROL SEQUENCE INTRODUCER (CSI) *(some may be used in text (STX...ETX) parts of math expressions)*	**Cf**	**BN**	**CB**
00B1	PLUS-MINUS SIGN *(must **not** have operands swapped for RTL)*	**Sm**	**ES**	**PR**
2044	**~~FRACTION SLASH~~** *(**not permitted** in math expressions)*	**Sm**	**S**	**IS**
215F	**~~FRACTION NUMERATOR ONE~~** *(**not permitted** in math expression)*	**Sm**	**S**	**IS**
2212	MINUS SIGN *(must **not** have operands swapped for RTL; similarly for other operators that are glyphically left-right symmetrical (has itself as mirror symbol) but has non-commutative semantics)*	**Sm**	**ES**	**PR**
2215	**DIVISION SLASH** *(big division slash)* ***CONTROL***	**Sm**	**S**	**AI**
27CC	**~~LONG DIVISION~~** *(**not permitted** in math expressions (SME…EME) See "Text styling control sequences – modernised ECMA-48 (ISO/IEC 6429) text styling", section 5.15 OVERLINE, for styling varieties of long division starting layout; note, that a long division may have math expr. as components)*	**Sm**	**S**	**AI**
29F5	**REVERSE SOLIDUS OPERATOR** *(RTL big division slash)* ***CONTROL***	**Sm**	**S**	**AI**
29F8	**BIG SOLIDUS** *("level" big division operator)* ***CONTROL***	**Sm**	**S**	**AI**
29F9	**BIG REVERSE SOLIDUS** *(RTL "level" big division op.)* ***CONTROL***	**Sm**	**S**	**AI**
FFFC	OBJECT REPLACEMENT CHARACTER	**So**	**ON**	**CB**
*n*FFFE, *n*FFFF (with *n* in 0—F), FDD0—FDEF	**Non-characters**. *Primary use*: **object replacement characters** (*per paragraph*, in this case: full math expression). Handle them as additional *object replacement characters*. For math embedded texts during display processing; each *full* ('top level') math expression has a mapping table from such chars to math expression structures. Note that bidi processing may (i.e. if enabled) be applied to a math expression embedded text (each in full, as a "paragraph").	**So**	**ON**	**CB**

All currency signs Sc have line break property PO (i.e., normally **after** a number, cmp. units).
The line break property NL for SME, EME, etc. are used in case C0/C1 math expressions are *not* parsed; if they *are* parsed there is *no* line break at those points; 'display' layout would use NLFs.

Appendix F – Caveats

C0/C1 variant

Not all contexts allow for the control codes utilised by this variant format. For instance, XML, HTML, SVG, Office Open XML, and similar (XML compatible) document formats do not allow for any C1 characters and only a few C0 characters; that goes also for so-called CDATA sections. That is why we have defined alternative formats that avoid (except for the few generally allowed) C0/C1 characters, the XML compatible format is using "tags" and "attributes" as controls instead of C0/C1 characters and control sequences, and the markdown compatible format is more in parallel with the C0/C1 format, but avoiding most C0/C1 characters which are generally either not allowed or have less than readable display when not interpreted (and given that the math formats here not yet implemented anywhere (so far "just" a proposal), they are not interpreted.

Text styling (current style as well as controlling changes to the style)

We say that (in math expressions embedded) text blocks, one can use the text formatting method that is "in the environment". That can range from nothing (purportedly "plain text") to enhanced/modernised ECMA-48 text formatting (CSI-m; SGR). To the author's knowledge, other similar pre-existing methods (like those for ATARI) are defunct and should never be used. Using something HTML-like (with or without CSS), in text blocks, together with the C0/C1 variant of math expressions is probably not a good idea. Same goes for RFT-like (or TeX-like) commands for text formatting, though also technically possible.

When using ECMA-48 based styling (SGR) either in the C0/C1 variant (**CSI**…**m**) or the markup variant (@...@), code 0 must not be used. **CSI 0m** (or even **CSIm**) is intended for terminal emulators only. If used, the **CSI 0m** is recommended to "close" any math expression it is 'in' (as well as closing several other things). **CSI 0m** is intended to be at the start of prompt strings in command line interpreters, to reset "everything" (including tables (if supported), and bidi (if supported)) to "normal text" rendering, in case the previous output was erroneous in setting styling back or in case the output was forcefully terminated. A text editor should render **CSI 0m** (and @0@ in the markdown variant) nominally as SUB a visible error indication.

XML compatible variant

The XML-compatible variant requires a bit of special parsing. Using full bracketing (which would be needed for using "plain old XML parsing" only) would result in a horribly verbose markup. Worse even than MathML and OMML.

The "use of the surrounding text formatting method" has some caveats also here. Unfortunately, HTML and SVG has slight differences on this point; though they both use CSS, the "span" tags are different. And Office Open XML uses yet another way of controlling styling (without CSS).

Markdown variant

The markdown variant has the same special parsing as the previous two, with the addition of some extra character references, akin to some of the character references in HTML.

Be reminded of that if the markdown variant is combined with some XML compatible document format (HTML, SVG, Office Open XML, …), the character sequence ']]>' *within* a math expression must be replaced by a character sequence that is equivalent (for the math expression) but is not the CDATA end mark; e.g. use ']] >' (note the space).

The "use of the surrounding text formatting method" would have some caveats also here; but:

1) Most markdowns do not have controls for font size nor for colours (the only type of styles inherited to math expressions, apart from overstrike).
2) While surrounding markdown could be used for italics, bold, underline, it seems like a bad idea to have one type of controls for that and another one for colours and (relative…) font size.
3) Using HTML/CSS (or SVG/CSS) controls could be appropriate for when the math expression is given as a CDATA section (and then use @...@ for bracketing each tag), the markdown variant can be used in other ways than in CDATA in an XML compatible format. XML compatible document formats are not the main target for the markdown variant; for that, see instead the direct XML compatible variant for math expressions.

For these reasons we have opted for using (extended/modernised) ECMA-48 style controls (SGR; CSI-m) in this case. Note that ECMA-48 allows for implementations to implement a subset, but we do not specify which subset here; for now, that is up to implementations to decide.

Math alphanumeric characters are forbidden

Unicode (and ISO/IEC 10646) has several characters, letters and digits, that are purportedly "for mathematical expressions", and come in some predetermined styles (italic, bold, etc.). However, those characters are in the formats proposed here not allowed; their use *shall* cause a parse error.

The reason for this is that the math alphanumeric characters are basically wrongly designed. They do not allow for multiletter identifiers (they are supposed to be used only for single letter identifiers), they do not allow for other letters than A-Z, a-z (with a small number of exceptions), some Greek letters, and a few Hebrew letters. [Similarly for Arabic math letters.] It is not uncommon to use multiletter identifiers (and TeX caters for some of that via \mathit, as opposed to Tex's default math style or \mathnormal). This is common in engineering, computer science and other areas where math expressions are used. Also words in languages that use letters other than A-Z, a-z are used in some math expressions, including languages that use completely different scripts. In this proposal (all variant formats) we allow for multiletter identifiers (use space to separate identifiers in a sequence, to corresponding to default math style or \mathnormal for single-letter identifiers in italics), and we allow for all letters (and decimal digits) [scripts] encoded in Unicode, though exactly which styles to support for each script is still open. This does *not* mean that all implementations of the math formats proposed here must support all scripts, or even all letters in a script.

Having each "math alphanumeric" character be equivalent to **SCI**<*style letter*><*the char to which it compatibility maps*> would be possible, but not particularly helpful, especially since we have default styles (somewhat similar to what is in TeX, but more of it), and, when needed, the explicit SCI-styling (esp. in the XML and markdown variants) are easier to "type" (whether via keyboard or computer program) than the "math alphanumerics". It also simplifies the specification here to simply forbid them, along with most other characters with a compatibility mapping in Unicode.

Font size: use relative change

If changing the font size within a math expression, it is usually better to use a relative size (like ‘120%’) rather than an absolute size. That of course goes for all the proposed variant formats (as well as other ways of representing math expressions). Both (extended/modernised) ECMA-48 styling as well as CSS allow for relative font size setting. This way the font sizing of parts of a math expression will work nicely when the inherited size changes, like when copy-pasting a math expression from running text to a heading, caption or into a graphic or into a superscript/ subscript, or just changing the “surrounding” size for other reasons of editing or style sheet changes for the surrounding text.

“External” CSS (and similar mechanisms that allow for “external” style definitions)

When doing style changes within a math expression, one should normally do that explicitly in the math expression, not via reference to a style defined outside of the math expression itself. The reason for that advice is that when doing style changes via externally defined styles, the author of the math expression (whether human or a computer program), loses control over the style change, which is often (but not always) undesirable.

Appendix G – Using math expressions in programming languages

While neither of the format mechanism for math expressions proposed here are aimed at being directly useable in a program, each of the variants could be used in comments (for the C0/C1 variant, the math relevant control characters must be allowed in comments, which is most often unspecified; note also the LF and CRLF is often the “line comment” comment terminator, so another NLF should be used for matrix row separation for math expressions inside a line comment). It would be possible (recall that this paper is a proposal and does not describe any existing implementation, yet anyway), to display math expressions in program comments (via an IDE).

That was regarding comments in programming language source code.

It does not seem like any of the proposed variant formats should be used in actual source code (disregarding the comments). And trying to do so may well be misleading. Programming also numerical “stuff” is not math, and using math expressions for computer integers or computer floating point, and matrices there-of etc. is not only misleading, it would be (semantically) wrong. Numerical computing is not math, and there is no “special” notation in math for numerical approximations to otherwise mathematical computations. So not having math expressions a executable code, if you like, is not only a matter of minor or major technical obstacles, but also principled. They are different, and should not assume that same surface presentation.