A true plain text format for math expressions (and its XML compatible equivalent format)

Kent Karlsson
Stockholm, Sweden

2022-12-03

1 Introduction

This proposal is a two-for-one, or even three-for-one, if you count the hints in Appendix C. We will give a representation for math expressions, as traditionally displayed, in two (or even three) different formats. 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 or XML. Appendix C hints at a possible "mark-down" style variant, based on the C1 version, and is thus equivalent to two other formats proposed.

Both formats are succinct, avoiding the verbosity of MathML and OMML. Even more succinct than (La)TeX math expressions. And the format(s) are not tied to one particular implementation (unlike (La)TeX math expressions. The format(s) presented here are also fully capable of handling RTL (right-to-left) math expressions which is sometimes used in RTL script contexts.

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 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 actually a set of new formats, catering for different contexts. Here we will give the formats for plain text contexts and for HTML/XML contexts. Others could be devised for other contexts, if need be. 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. Also, there is a need for better integration with the contextual document formatting system. One should not need to have a separate system for giving size, colour, shadow, strike-through (assuming such settings are available in the contextual document formatting system). This way it is easy to, 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 hard to do, if at all possible, in existing systems for math expression layout.

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, MathML, OMML, AsciiMath, UnicodeMath) 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 text styling (the XML and HTML compatible variant). We thus will not need "math special" ways of doing styling, so no need for constructs like *<mstyle>*. Indeed, we will try to keep the controls and markup to a minimum, without losing too much expressive power.

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. 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 (both variants), bidi controls are *not* interpreted in the math expression, and bidi processing (and cursive joining) applies only if enabled outside of the math expression and then only for digit sequences and letter sequences *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*.

Further, the math layout system must not "undisplay" some printable characters in some circumstances (with one exception: explicitly "phantom" elements; plus that markup controls actually must be undisplayed, except when editing "source code"), nor require some kind of "fence" nesting (again not counting markup controls). Any such handling prevents certain common mathematical notations.

We *also* focus on the proper use of Unicode characters, e.g., not using combining characters to denote an over-an-expression mathematical operator, as well as never using parentheses or symbols to carry formatting significance. Formatting is done via formatting controls, not symbols, but formatting controls can affect certain symbols, e.g., their length or height.

All pre-existing formats, and the HTML/XML 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 both based on Unicode, which has many math symbols encoded. But both proposals here can handle RTL math expressions. Also, the formats proposed here do not "silence" printable characters (outside of the HTML/XML markup itself and ignoring "phantom" spacing) nor require that open and close "fences" match in any way, as opposed to some other formats. Nor are "MATHEMATICAL" characters (invented for MathML) used at all. Note that there is no requirement to use "MATHEMATICAL" characters at all in a math expression format, and for the formats here we will not use them, but instead a more general mechanism. And further, combining characters are used correctly in the two here 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 and XML. While, technically, the first format (as proposed here) is workable with HTML and XML, HTML and XML are blanket allergic to control characters (except certain whitespace characters). So, we propose a variant format that is compatible with HTML and 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).

The two proposed formats are parallel, as near as can be, and conversion between the two is straight-forward. Both 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”) can be used. The large number of markup tags that must be used when using MathML or OMML is avoided in the formats here.

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

Goals for this proposal:

a) Enable proper traditional 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 only.
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 (except if part of a “phantom” 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 (to be 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 multi-letter names for functions, variables, chemical elements, units, ...
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 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 use, 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.

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” for expressions. That is an “outside of the math expression” formatting issue. Line breaks, centring, and numbering are all outside of the math expression.
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.

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: **MEO**E=m c**SCI^**2**MEC**, where the bold parts represent C1 control codes. The space is needed to make two variable names, rather than one multi-letter one.
- Our format, XML: **<me>**E=m c**<rsp/>**2**</me>**, where the bold parts are XML tags.
- Our format, appendix C: **${**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 point of start of a math expression

Math expressions have a particular kind of styling, for a special kind of math font 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. 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 (with an exception for 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 and 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 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 (**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 https://www.unicode.org/L2/L2022/22026r-non-bidi-mirroring.pdf). Note: mirroring as 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 script.

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, and similar font styling, which implies hidden case mapping. This should be disallowed also for embedded texts (**TXT**/**<txt>**) both as editing operation and as display operation (e.g., via CSS styling).

4 Math expression formatting and math styling controls

As mentioned, we will propose two versions of the math layout system proposed here, one based on C1 control characters, and one that is compatible with HTML/XML using control tags.

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 **MEO**, **MEC**, **MHD** and **MMS**, respectively. They will be detailed below, but **MEO**…**MEC** 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 another control character. 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*.

Math styling and layout controls, **SCI** *x*, sci-sequences, should be ignored outside of outermost **MEO** … **MEC** brackets. Note that **MEO** … **MEC** nest. **MEO** … **MEC** are used as grouping brackets inside the outermost **MEO** … **MEC**. **MEO** … **MEC** *never* generate visible parentheses, brackets or braces. All parentheses, brackets or braces ("fences") must be given explicitly.

On the other hand, use of visible parentheses, brackets etc. *never* imply any **MEO** … **MEC** bracketing. 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 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) area and local conventions.

4.1 Math expression styling and layout controls

We will here use four hitherto unused (or in the case of U+0084: withdrawn) 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, MATH EXPRESSION OPEN, **MEO**. See section 4.2.1.
- U+0081, MATH EXPRESSION CLOSE, **MEC**. 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. 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.
- **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 from 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**. Note though that there are no tab stops of any kind inside a math expression.

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\u0008-\u000D\u001C-\u007E\u00A0-\U10FFFD]

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**; like for other C1 control codes: **ESC** [\u0040-\u004C\u0059-\u005B] (excluding code page switching (always), device controls (always for documents, in particular math expressions), control strings (always for math expressions), and private use control codes.

We will also consider a few Unicode math characters, general category Sm, to be control characters (no SCI), 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.

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 as is the case in TeX, so there is nothing unique 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. One to start the math group, and one to end the math group.

- U+0080, MATH EXPRESSION OPEN, **MEO**. 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. **MEO** is the only “math control” that is interpreted *outside* of a math expression. Thus, all **SCI** controls stay undefined outside of a math expression (as defined here).
- U+0081, MATH EXPRESSION CLOSE, **MEC**. An invisible close parenthesis used for grouping. Compare (La)TeX’s $ and }. It closes the most recent (textually) still open **MEO**. 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 **MEO**…**MEC** around the copied/pasted part of the math expression. Indeed, the editor may need to balance up **MEO**…**MEC**, in case the copied part was not well balanced. (The latter may result in strange parts being copied, though.)

A math bracketed expression is **MEO** *expr* **MEC** where *expr* is a math expression as defined in section 4.12.1. The **MEC** that closes the **MEO** that started outside of the math expression closes the math expression and returns to “normal” text. No styling changes, by whatever mechanism, are interpreted inside the outer **MEO**…**MEC** except partially within a **TXTO**…**TXTC** block (see below). 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.

The **MEO** and **MEC** never generate any visible bracketing (not counting a “show invisibles” mode, which a text editor may have).

4.2.2 Using HTML/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). 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).

4.3 Math styling

In math expressions the style of a "name" (of functions, variables, standard functions, sets, 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 fonts, and certain "ranges" of such fonts. 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.

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/XML tags with attributes. In both cases we will use default styles, like done in TeX and even more in AsciiMath, to cut down on the prevalence of the styling controls very much. 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/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, most importantly, symbols (including, of course, math symbols and 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 in a math expression.

The styled item can be one of:

1. A sequence of proper combining sequences where the base characters are letters. Included as letters are CJK ideographs and 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 (though sometimes Greek and Latin letters are mixed). However, an implementation need not support all scripts, 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, but not recommended; 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 (PUNCTUATION SPACE)) starting and ending with a decimal digit, indeed each instance of allowed punctuation must be preceded and succeeded by a digit. For convenience, a (single) SPACE *between* digits is interpreted as PSP. All the digits in the sequence *shall* be of 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 combining sequence with a base character that is a symbol (including punctuation, S*, P*), possibly preceded by an **MMS** (**MMS** is mostly intended for RTL math expressions). Technically all symbols (including punctuation and arrows) can be math styled, 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*. 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, 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 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.

In order to write a multiplication (of single letter variables), commonly written as a juxtaposition without any operator (INVISIBLE TIMES is not recommended), use a space to separate the operands of the multiplication. This should give the proper spacing (SPACEs are not rendered inside of a math expression, except inside an **TXT/<txt>** block, see below), as well as delimit any math styling to the part it should be at. When using multi-letter variables, which is uncommon except for certain standard functions, 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., **MEO** sin 5 **MEC** and **MEO** 50 kW **MEC**. The spacing otherwise, like between two italic letter sequences, is tighter (but not as tight as inside an italic letter sequence, compare \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 **TXT/<txt>** blocks can, in principle, have such zeroes with interior slash or dot).

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”, 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.
  Also multi-letter unit symbols (including prefixes) by default has this math style. (In SI, short unit "names" are called "symbols", 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). This includes units “allowed for use with” SI, not just the pure SI units. Specifically, identifiers matching this regular expression have **SCI A** math style as default:
  [YZEPTGMkhdcmµunpfazy][sgmtKLNJWAVCHFSΩT]|[dcmµunpfazy](B|Np|rad|sr|as)|Np|rad|sr|
  [YZEPTGMkhdcmµunpfazy]?(Hz|Wh|Ah|eV|Wb|bar|Pa|Bq|Sv|Gy|mol|kat|cd|lx|lm|ohm)|
  [YZEPTGMk]b|[YZEPTGMk]?(Bd|bit|px)|(Yi|Zi|Ei|Pi|Ti|Gi|Mi|Ki)[oB]|min|kcal|ha|°C|pH|pOH
  While “pH” and “pOH” are not counted as “units” per se, they do have some similarities with bell and neper, and (what we consider here) are typeset as unit style. “Wh” is really a multiplication of two units, in the representation proposed here: “W **SCI A**h”, but for convenience it is included in this regular expression as if it were a single unit symbol; “ha” is technically a historical SI unit (prefix “h”, *hecto*, to the unit “a”, *ar* or *are*), but is still in very common use; “u” is a common fallback for “µ”; “µ” and “Ω” stand for two characters each in the regular expression above. 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”, “T”, “B”, “b”, “o” as unit symbols (note: these are without prefix, just single-letter unit designations), as most of them have italic style, **SCI C**, by default (except N, C and H which have double-struck style, **SCI I**, by default), as well as “ppm” (note that SI recommends to use notation such as µL/L or µg/g, to be clear about what is measured, rather than saying “by volume” or “by weight”), “ppb” and others.

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 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 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 for all Sc, currency symbol, characters like “€”, “£”, etc. in math expressions (as defined here).

- **SCI B** MSUB Math serifed bold upright prefix operator. (Compare 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. <u>This is the default style for letter-based sequences, except for certain letter sequences (see above and below)</u>. Compare TeX’s default math style, *\mathit* and *\mathnormal* (actually corresponds more to **SCI c**). *\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 contexts.
- **SCI D** MBIT Math bold italics prefix operator. Cmp. \mathbfit.

- **SCI E** MUSS Math sans-serif upright style operator. <u>*Always* used for symbol-based combining sequences in math expressions (including punctuation, but not letter-like symbols or currency symbols). Such symbols in math expressions *cannot* have any other style.</u> This holds for these symbols also in **TXT** blocks (see below). (Compare 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 a C is directly after a "°" symbol); for "natural numbers", …, "quaternion numbers". (Compare 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.(Compare TeX's \mathcal.)
- **SCI N** MBSC Math bold script style prefix operator. Cmp. \mathbfcal.

- **SCI O** MSSC Math swash script prefix operator. (Compare 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. (Compare 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. (Compare 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. (Only lowercase p need be supported in this style.)

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 control character terminates the identifier (and is not permitted otherwise either, unless given a semantic here). 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, definitely no bidi mirroring) is done on individual letter and digit sequences, 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/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* is a non-empty sequence of combining sequences each with a letter as base character (which we refer to as identifier), a letter-like symbol, or a non-empty digit sequence possibly with full stops, commas and PSPs. Note that non-letter-like symbols (including punctuation, arrows) cannot be styled this way or any other way (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, 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 as the inherited (from outside of the math expression) text style, "normal" text styling may be used (like CSS), except that it mostly 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 (**MEO**…**MEC**, **<me>**…**</me>**). This implies that a symbol sequence in a text block get the math style for symbols. 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, but that should be avoided). 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>** $x + y$

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.

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.

4.4.1 Using C1 control characters

We will use **SCI '** and **SCI "** as the math embedded text brackets and refer to them as **TXTO** (TEXT OPEN) and **TXTC** (TEXT CLOSE) respectively. If there is a "normal" text styling mechanism available in the contextual system, then that may be used within the text block. Symbols are still in math style in a text block.

Each sequence of no-break spaces in a math expression is implicitly embraced by **TXTO**…**TXTC** and are thus allowed by the grammar.

Each sequence of symbols, P*, S*, Z*, except for pure Z*+ between letters/digits (*NLF*s are counted as **SP**), in an embedded text block is implicitly embraced by **MEO**…**MEC** and are thus insulated from bidi rearrangement (within the sequence) and bidi mirroring as well as get the math style for symbols also in the text block.

4.4.2 Using HTML/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. 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 no-break spaces in a math expression is implicitly embraced by **<txt>**…**</txt>** and are thus allowed by the grammar.

Each sequence of symbols in an embedded text block is implicitly embedded by **<me>**…**</me>** and are thus insulated from bidi rearrangement (within the sequence) and bidi mirroring as well as get the math style for symbols also in the text block.

4.5 Positioned layouts

Superscripting and subscripting are among the most common layouts for math expressions. They are used for exponentiation and indexing, and also for giving start and end of summations and integrals. Superscripts/subscripts can be to the right of, to the left of, or above/below what is superscripted or subscripted. Like for all other math layout in this proposal, there is no inherent semantics in superscripting or subscripting. That is for a higher level of interpretation.

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. That does not preclude the use of superscript digits in, e.g., unit denotation in "ordinary" text.

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 super/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), 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:

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 (with some exceptions, see below), nor is U+2118 (use **SCI W**p). (NO-

BREAK) HYPHEN is also usable, but not recommended; note that HYPHEN-MINUS is mapped to MINUS SIGN, not HYPEN.
2. A non-empty sequence of decimal digits and certain punctuation (COMMA, FULL STOP, PSP) starting and ending with a decimal digit, indeed each instance of allowed punctuation must be preceded and succeeded by a digit. A (single) 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 (these 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 (**MEO**…**MEC**) subexpression or an embedded text (**TXTO**…**TXTC**) 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). For instance, placing three dots above a variable (not a great idea for multi-letter variables), *x* say, use **TXTO…TXTC** (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 to cover the horizontal size of the above- or below-script. For horizontal arrows and operators that get an above- or below-script, the 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 spacing characters, *never* non-spacing characters. The exact mechanism for this size adaptation (often some kind of stretch, rather than 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 operator.

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 yet another math expression. The placement of the superscripts/subscripts need to adapt to that size adaptation. The exact mechanism for this size adaptation (often some kind of stretch, rather than font size change which would often look clumsy), is not specified here.

4.6 Above-script/below-script layout

Often math expressions have an above-script or a below-script, indeed there may be several. These are, layoutwise, not variants of superscripting/subscripting (even though we mixed them above). *Sometimes* above-scripting/below-scripting is, *semantically*, a variant of superscripting/ subscripting. But semantics is out of scope for the proposals here, only layout is 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\TXTO**def**TXTC**”, preferred over U+225D, ≝) 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>) 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 the width of the below-script.
- **SCI \** ABV (REVERSE SOLIDUS; <U+009A, U+005C>) 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 the width of the above-script.

4.6.1.1 Variants which stretch above/below brackets and arrow above/belowscripts

- **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 are rounded only at the ends.

- **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 are rounded only at the ends.
- **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 **MEO MEO** <*base expr*> **MEC ABVS ⏞ ABV TXTO** <*remark above the above-brace*> **TXTC MEC**.

The structure of an above-scripted/below-scripted math expression has the following structure:
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/XML compatible control tags

We will use the following control tags:

- **<blw r="*(n|s|t|u)*"/>**. These elements follow the base and an element of a stack of below-script math expressions. 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)*"/>**. These elements follow the **<blw>** elements (if any) and starts an element of a stack of above-script math expressions. The **r** attribute indicates 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 section 0, 4.3.2 (including default styled elements), or 4.4.2.

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) 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: **MEO MEO** … **MEC ABVS** <U+005E> **MEC** (for readability, the circumflex usually needs to be stretched, hence **ABVS** rather than **ABV**). (For dot(s) above/below, use 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 or below, which we dealt with above, they may also occur to the left of and to the right of an expression (keeping in mind readability, so it may need to be visibly parenthesised). 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 Superscripting the left operand to the upper left of the right operand (the base in the pseudo-grammar below).
- **SCI =** SUBL Subscripting the left operand to the lower left of the right operand.
- **SCI _** SUBR (cmp. TeX:s _) Subscripting the right operand to the lower right of the left operand.
- **SCI ^** SUPR (cmp. TeX:s ^) Superscripting the right operand to the upper right of the left operand (the base in the pseudo-grammar below).

Subscript/superscript corners:
(*a* **SUPL**)?(*b* **SUBL**)?*base*(**SUBR** *c*)?(**SUPR** *d*)?
where *base* is as defined in section 4.2.1 or 4.6.1, and *a*, *b*, *c*, *d* are as defined in 4.2.1, 4.3.1 or 4.4.1.

4.7.2 Using HTML/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):

- **<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 0 or 4.6.2, and *a*, *b*, *c*, *d* are as defined in 0, 4.3.2 or 4.4.2. Note that these tags are "operators", there is no content for these tags, in contrast to the **<sub>** and **<sup>** tags in HTML (used outside of math expressions).

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

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).

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 technically for a different reason) or the content of an **<st>** element. Indeed, these operators should be seen as control characters and not as symbols, and for the HTML/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 **TXT**/**<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…)

4.8.1 Using C1 control characters and control(!) graphic characters

We will use infix controls for fractions: the ones that Unicode already defines, plus one C1 control code (U+0084, ex-INDEX control) as operators. The operands are as defined in 4.2.1, 4.4.1, or 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 infix). 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 centred above/below that line. If *both* operands are ***un****bracketed* (no **MEO** ... **MEC**) digit sequences (no punctuation, no spaces) in default style (explicitly or using the default mechanism), then the digits are rendered small (approx. index-size). For example,
  “**MEO MEO** a+b **MEC MHD** c **MEC**” 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).

- **DIVISION SLASH** (U+2215). Inside of a math expression, this is the (possibly big) *slanted* division stroke with “super” nominator (left argument), and “sub” denominator (right argument) operator. If *both* operands are ***un****bracketed* (no **MEO** ... **MEC**) digit sequences (no punctuation, no spaces) in default style (explicitly or using the default mechanism), then the digits are rendered small (approx. index-size). Note that this automatic size reduction only applies to pure digits arguments.

For example, "**MEO MEO** a+b **MEC** <U+2215> c **MEC**" should display like this: $a + {}^{b}/_{c}$.
This means that we regard **DIVISION SLASH** as a control character, not an ordinary symbol. Likewise for its "mirror": **REVERSE SOLIDUS OPERATOR**.

- **BIG SOLIDUS** (U+29F8). Inside a math expression, this is the (possibly big) *slanted* stroke with *level* nominator (dividend) and denominator (divisor) operator. 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 (**MEO** ... **MEC**) never generate visible parentheses or brackets of any kind (except perhaps in a "view invisible controls" mode in a text editor). "**MEO MEO** a+b **MEC**⁄ c **MEC**" should display like this: $a + b/c$. Note that despite the **MEO**…**MEC** there are no parentheses. They need to be written explicitly, inside of the **MEO**…**MEC**, by the author of the math expression. "**MEO MEO** (a+b **MHD** c) **MEC** <U+29F8> d **MEC**" should display similar to this (except that parenthesis stretch is glossed over here, see below regarding parenthesis stretch, and there should be no size changes): $\left(a + \frac{b}{c}\right)/d$. Indeed "**MEO** (a+ b **MHD** c)<U+29F8>d **MEC**" 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**.
- U+2044 (FRACTION SLASH) and U+215F 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.

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.

4.8.2 Using HTML/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/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)/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, or 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/HTML compatible variant. SOLIDUS can be used directly, but acts as an ordinary operator symbol, no special (and stretchy) layout.

As above, also here, when both arguments are pure digit sequences (no **<me>** brackets) with default style (explicit or using the defaulting 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*. Likewise, the notations for “manually” doing multiplication of numbers, or addition or subtraction of numbers, 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. They *may* be included in a future revision, but probably not, as they are a separate category of notation.

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 $\begin{bmatrix} a & \cdots & b \\ \vdots & \ddots & \vdots \\ c & \cdots & d \end{bmatrix}$. 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.

4.10.1 Using C0 and C1 control characters

Within **MEO** … **MEC** bracketing **SP**, **HT**, **LF** and more are used as separators and code layout only. They are not rendered. But math expressions also needs to have matrix layouts. It need not semantically be a matrix of any kind; it is just the layout.

A math matrix layout is started with **SCI ,** (SCI, COMMA), **SCI .** (SCI, FULL STOP), **SCI ;** or **SCI :** and 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.

- 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 space.
- Terminating **SCI HT**: the content is horizontally centred in the cell 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).

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. 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 **MEO**…**MEC** 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 **MEO**:s that are not closed within the matrix layout, they are all closed by the end matrix layout **SCI#** control. Any "surplus" **MEC** controls in a matrix layout cell is ignored (i.e., they cannot close an **MEO** 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/HTML compatible variant, the XML/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 **MEO**…**MEC** 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 compatible control tags

For the HTML/XML 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).

A math matrix row has cells that have horizontal alignment attributes:
<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>**.

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".

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.

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.

4.11.2 Using HTML/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).

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 0 or 4.3.2 or 4.4.2.

4.12 LTR layout

Math expressions 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, see below).

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

There is no parsing of parenthesis or other fences, no parsing using (visible) operator precedence. 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 precedencies 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.

4.12.1 Using C1 control characters

A sequence of zero or more math expressions according to 4.2.1, 4.4.1, 4.7.1, 4.8.1, 4.10.1 or 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. Whitespace is collapsed, effectively though maybe not explicitly, to a single WJ.

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

4.12.2 Using HTML/XML compatible control tags

A sequence of zero or more math expressions according to 4.2.2, 4.4.2, 4.7.2, 4.8.2, 4.10.2 or 4.11.2. Whitespace, **SP**, **HT**, *NLF*, may occur around the math expressions, and may be needed to break letter sequences into multiple letter sequences. Whitespace is collapsed, effectively though maybe not explicitly, to a single WJ (which is actually a word *separator*).

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 explicitly requested or unrequested by controls. The glyphs in the fonts are always without the extra space.

5.1 Bounding boxes

For each layout subexpression, a bounding box is calculated. A bounding box here 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.

The bounding box should "closely circumscribe" the (would be, if phantom) display of the enclosed math expression. It should not be too tight, needing a little bit if typographical spacing.

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

5.2 Binary operators

Conventionally, binary operators, except division operators and set minus, in a math expression get a little bit of extra spacing before and after it. This must not be built into the "glyphs" in fonts, since the same symbol may be used as prefix or postfix operator, and then not get the extra spacing.

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 binary operator extra spacing for a symbol, as well as turn it off. 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 either a default closing fence (i.e., general category Pe), an **MEC** or an **</me>** (depending on C1 or XML/HTML variant), a number, an identifier, or a non-leftmost matrix cell start, and
- just after it (skipping "ignored whitespace" in between) is either a default opening fence (general category Ps), an **MEO** or an **<me>** (depending on C1 or XML/HTML variant), a number, an identifier, or a non-rightmost matrix cell end.

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

Note that this does not consider superscripts/subscripts/etc. I.e., this heuristic does not take into account the parsing into a layout structure. So it may "guess" wrong. Likewise, sometimes a default opening fence is used as a closing fence and v.v. Thus, trying to consider superscripts, division etc. will still not make this perfect. To correct a "wrong guess" (by that heuristic), one can add **MEO**/**MEC** 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 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 as 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”. Note that commas that occur in numerals are considered part of the numeral, not a separate symbol, and thus does not get any spacing after them. **SCI -** can be used to remove this extra space for a list item separator punctuation symbol.

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”.

1. A numeral adjacent to (either order) an upright identifier, the symbols %, ‰, ‱ or a currency symbol: automatic four-per-em space between.
2. Italic/slanted identifier adjacent to (either order) an upright identifier, the symbols %, ‰, ‱ or a currency symbol: automatic four-per-em space between.
3. A numeral adjacent to a numeral: automatic 1 em space between. Placing numerals adjacent to each other without an explicit visible operator is not recommended.

If that automatic space is not desired, one can use, e.g., an empty math expression in between.

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 box, which helps to position a math subexpression with other subexpressions, is not always a rectangle. It is often a parallelogram (for italic/slanted ids, numbers, letter-like symbols), or even a trapezoid with horizontal top and base (of which a parallelogram, with horizontal top and base, is a special case). 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 **MEO**…**MEC**, and then have **SCI @** (RECT) directly after the **MEO**. For the HTML/XML compatible variant, we use the attribute to **<me>**: **<me t="RECT">**.

5.7 Phantom expressions

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 suitable. Then 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, 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).

In the case of italicised content, the spacing is actually a parallelogram, and in general it is a trapezoid (with horizontal top and bottom). This affects the positioning of superscripts and subscripts.

In the C1 version, we use the prefix **SCI *** (PHA) just after **MEO** to keep just the trapezoid (with horizontal top and bottom) space, not displaying the math expression. For the HTML/XML 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. In the C1 version, we use the prefix **SCI |** (VPHA) just after **MEO**. For the HTML/XML 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. 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 and may cause an error message.

For the HTML/XML compatible version, we add an attribute to the **<mr>** (math matrix row) tag: **<mr a="*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 SP,** 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 SP**. 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 **SCI BS** to decrease the spacing between the columns by 1/6 em per **SCI BS**; but using **SCI .** or **SCI ,** with **SCI SP** is preferred in this case, avoiding to use **SCI BS**. However, the spacing will never be negative, so cells will never overlap. All other occurrences of **SCI SP** or **SCI BS** (in a math expression) are ignored and may cause an error message. Tabulation characters actually terminate cells in math expressions as defined here, but for the rightmost column any **SCI SP** or **SCI BS** are ignored. As a shortcut, **SCI IS1** gives an increase of 3 em (18 **SCI SP**).

For the HTML/XML compatible version, we add an attribute to the **<mc>** (math matrix cell) tag: **<mc a="*n*">**, where *n* is a small integer number (including negative numbers), where the number corresponds to the number of **SCI SP**s (negative value refers to **SCI 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 (unequal number of cells in the rows) 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

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.

6.1 Vertical math centre for styled elements

The vertical math centre for styled elements (identifiers, decimal numerals, symbols) and for embedded texts is the vertical position of the MINUS SIGN glyph (at the current font size).

6.2 Vertical math centre for sequences

The sequences of math expressions that occur inside **MEO**…**MEC**/**<me>**...**</me>** are aligned on their vertical math centre, which becomes the vertical math centre of the sequence. The same goes for the sequences in 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”).

6.5 Vertical math centre for matrices

The vertical centre of a math matrix layout is the true vertical centre of the matrix layout.

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 math expression (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’s (0 (bottom row), …, 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+229F9**, a **SCI 0** after the fraction control character selects the denominator’s vertical math centre as the vertical math centre for the fraction. A **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, a **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. A **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/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 naively 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. There cannot be an unreliable change of order of arguments, nor an unreliable change in which operator or fence is used. This means that the bidi algorithm has to be strongly restricted in a math expression, and so-called bidi mirroring (via the bidi algorithm) must be completely ruled out in math expressions.

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. Inside of a math expression, bidi applies *only* to individual identifiers (and numerals) as well as embedded text items (**TXTO … TXTC** (**<txt>**…**</txt>**), *each in isolation*. And, bidi mirroring shall *never* be applied inside a math expression, not even inside **TXTO … TXTC** (**<txt>**…**</txt>**) embedded text blocks. That goes for all kinds 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.

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 for bidi is always LTR for an identifier/ numeral (given the restrictions, that should not matter, but just to have it defined) 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. **TXTO … TXTC** 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 **MEO**…**MEC** 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". 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 (**TXTO**…**TXTC**, **<txt>**…**</txt>**). Symbols (including punctuation), in math expressions, always have their 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", arrows and arrowlike symbols have bidi-mirrored set to "false". For math expression use, that is, well, incongruent. But as mentioned, that is solved by never bidi-mirror any symbol (S*, P*) at all. Further, we cannot allow any "display time" mirroring of any symbols because that is way too unreliable and may inadvertently 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 or not, to make it easier for the math expression author).

Another reason for not doing argument "reversal" 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 and 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 arguments).

"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 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

As explained above all mirroring must be done as character substitutions, as edit operations. To aid in this edit mirroring, 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 for right to left math expressions: Some symbols (here we include punctuation and arrows) 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", though most such symbols do have an allocated mirror symbol. These are called out in *BidiMirroring.txt* as comments at the end of the file, and likewise in *ExtraMirroring.txt*.

One would expect 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 A). 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>, or as we will notate it: **MMS** ∾.

A disadvantage with this "visual" encoding of math expressions is that it is a bit harder to do the input. 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. Math expressions are not "running text" after all, so editing math expressions is a bit special anyway.

Annex A gives a list of **MMS**-mirrorable characters.

8 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 edit operation that is not directly related to case mapping.

The reason is that the "same" letter of different case are different "mathematical identifiers" (be that variables, constants, or unit designations, etc.). Same letter (of 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 **TXT**/**<txt>** embedded texts, though it might not be so clear-cut then.

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 compatible version) '>' need be shielded from a combining solidus stroke if the '>' is part of a tag (ending the tag). The latter "should not happen" (it is inappropriate according to this specification).

Note, however, that mapping to NFKD or NFKC may turn an unpermitted math expression into a permitted one, do inappropriate mapping for "GREEK * SYMBOL" and "HANGUL LETTER *", and may split some "composed" symbols. (It also maps isolated form Arabic letter after a **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 NFKD is thus not recommended, since it may have undesirable effects.

The "context" of the math expression may have a mechanism for character references, like XML and HTML do. Whether the normalisation (NFD, NFC) affect also the characters referenced via character references or not depends on the point when the normalisation is applied. For the HTML/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/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 NLFs, in particular between <**CR**,**LF**> and **LF**, but sometimes also **CR**. This is unproblematic for both variants of math expression representation proposed here.

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 NLFs), 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+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 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.
- 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.

- 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 should 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 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 lexer 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 **TXT/<txt>** group (explicitly bracketed), where it is interpreted as NO-BREAK HYPHEN (HYPHEN is also interpreted as NO-BREAK HYPHEN).
- ASCII TILDE, ~, if directly followed by an **SP**, is mapped to TILDE OPERATOR.
- 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 HYPHEN-MINUS in a row, ---, is mapped to **MHD** (MATH HORIZONTAL DIVISION) (C1 version) or **<dv t='t0'/>** (XML/HTML compatible version) or just **<dv/>** (as **t='t0'** is default). Note that this maps to a control code or control tag, not to a symbol.
- Two SOLIDUS in a row, //, is mapped to DIVISION SLASH (a control…) or **<dv t='t1'/>**.
- <= 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 (∓); not very common but included for symmetry.
- =/ 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 is 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: $\#\{n : n > 1\}$
- {{ is mapped to **MEO**, only for the C1 variant and not for an outer **MEO**.
- }} is mapped to **MEC**, only for the C1 variant and not for an outer **MEC**.

Notes on some other characters:

- Invisible operators, there are a few defined in Unicode, should *not* be used (but if used, they are interpreted as SPACE, which in turn is basically ignored).
- Space characters (except for SPACE itself), like MEDIUM MATHEMATICAL SPACE (MMSP), may be used to adjust the position of subexpressions (though 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 those reinterpreted as SPACE) are interpreted as (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): use **SUPL** (**<lsp/>**) and **SUPR** (**<rsp/>**) 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. MIDDLE DOT and DOT OPERATOR can be used for multiplication, esp. between numbers as space is then more commonly used for digit grouping.
- Sometimes, × is used for multiplication between numerals (but next to never when there is a variable), and sometimes ÷ is used for division between numerals.

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 (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). This is a general issue, and the proposals here should not be introducing any new security issues (not even for the symbol mappings above).

In addition, math expressions should be limited in coding size. Otherwise, an (outermost, first) **MEO** followed by whatever but no (matching) **MEC** 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 **MEO** to “gobble up” all the rest of the text. The parsing of a math expression can also be terminated by any syntax error, not enabling math expression reading (starting with interpreting an **MEO**) for, say, another 100 characters.

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 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 details. That is also needed for the structure “reconstruction” (i.e. parsing) at an extra level of parsing after the XML/HTML parsing. Requiring such an extra level of parsing is vital for enabling a succinct expression in the XML/HTML compatible variant. Explicitly XML/HTML bracketing everything would create the same (or worse) verbosity that MathML and OMML suffers from.

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

9.1 Character classifications

The characters defined in Unicode have properties to classify them. We will here use these properties for indicating which characters are allowed where, without enumerating them all 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**

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. However, we will not cover this here, except to say that numerals in bases greater than 10 will be "relegated" to be expressed using **TXT**/**<txt>**.

As *decimal-digit*s we allow for all characters that have general category Nd *and* do not have a compatibility decomposition. In particular, 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 otherwise, though a script change does *not* break up the numeral into several numerals. As usual, longest match rule applies.

num ::= (*decimal-digit*)+((","|"."|**PSP**)(*decimal-digit*)+)*
// Post-lexical check: 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).

This syntax cannot handle numerals in bases greater than ten. A numeral in hexadecimal, base 16, can be given as **TXTO**<hexadecimal numeral>**TXTC**. Math styling is not available for numerals given as embedded text.

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 **MEO**, **MEC**, **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 Wp** 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; 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 *”. In particular, that excludes the so-called “MATHEMATICAL” letters. We have other ways of “styling” identifiers in a math context. There *should* be no mixing of scripts in a single identifier. Though it is common to mix Latin letters with Greek letters, and even Hebrew letters, it is usually not in a single identifier (but some SI units do mix scripts). 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*(*letter* (*combining-mark*)*)+)*
// Post-lexical check: the letters and combining marks should all be of the same script.
// Note that hyphen cannot be given as HYPHEN-MINUS, since that is mapped to MINUS SIGN.

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 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 general category and compatibility decomposition:

```
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
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
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)
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 section 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
2A18;INTEGRAL WITH TIMES SIGN
2A19;INTEGRAL WITH INTERSECTION
2A1A;INTEGRAL WITH UNION
2A1B;INTEGRAL WITH OVERBAR
2A1C;INTEGRAL WITH UNDERBAR
3131..3163,3165..318E HANGUL LETTER *

The following symbols should be avoided, and instead use the compositions given (C1 variant, spaces 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* and

// certain Ll. We also allow some mappings of a symbol sequence to a symbol; see
// section 7.5. These (pre-processing) mappings are not included in the grammar here.
// (Note: some of the mappings map to control characters or control tags, not symbols.)

Symbols, and in **TXT**/**<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 lexer, which is not given).

me ::= **MEO**(**PHA**|**VPHA**)?**RECT**?(**SP**|**HT**|***NLF***)*(*stretch*(**SP**|**HT**|***NLF***)*)***MEC**

// Outside of the outer **MEO**…**MEC**, all math controls except **MEO** are ignored.
// If a **MEO** has a **PHA** or **VPHA**, that is inherited to all inner parts of the **MEO**…**MEC**;
// there is no “unphantom” control; selection of the “vertical” part is done at the
// outermost **VPHA**-marked **MEO**…**MEC**, and that implies **RECT**. Sequences
// of no-break spaces are automatically embraced by **TXTO**…**TXTC**. 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 ::= **TXTO** (*plaintext*|*me*)* **TXTC**

// *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 **TXTC** nor **MEO**, other math controls are ignored.
// Non-empty longest sequences of P*, non-script-specific 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 **MEO**…**MEC**. 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 **TXTC**.

closed ::= *me* | *st* | *txt*

```
stack ::= closed ((BLW|BLWS|BLWT|BLWU) closed)* ((ABV|ABVS|ABVT|ABVU) closed)*
around ::= (closed SUPL)? (closed SUBL)? stack (SUBR closed)? (SUPR closed)?
frac ::= around(MHD|U+2215|U+29F5|U+29F8|U+229F9)(SCI [01])?around
stretch ::= (around (LSS|LSST|LSSU|LSSV))*(around|frac|mx)((RSS|RSST|RSSU|RSSV) around)*
        // the “division operator” controls (MHD, …) bind harder than the stretch operator controls.
        // The LSS operators associate to the right, the RSS operators associate to the left

mc ::= ((SCI SP|SCI IS1)*|(SCI BS)*)?SP*(stretch SP*)*(HT|HTJ|SCI HT|SCI VT|SCI LS)
mr ::= mc*      // the ((SCI SP|SCI IS1)*|(SCI BS)*)? parts are interpreted only for the top row
mx ::= SCI(,|.|;|:)(SCI [0-9])?mr (NLF(PLD*|PLU*)mr)*SCI #  // more “powerful” than MEO…MEC
                   // in case of nesting imbalance (a syntax error), for syntax error recovery;
                   // SCI 0 refers to the bottom row, and then upwards for SCI [1-9]
```

9.3 DTD for the XML/HTML compatible version for math expressions

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 not 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 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 an additional level, 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 (%stretch;)*>     <!ATTLIST me
                                   d (Y|PHANTOM|VPHANTOM) "Y"
                                   t (T|RECT) "T"
           -- Size and colours 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 compatible version of math expression representation defined here we here illustrate the desired integration with example DTD snippets. There is no proposal here to do updates to the DTDs.

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">
```

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>**.

For HTML 5, **<txt>** would have the (informal) HTML 5 ELEMENT declaration:

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

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

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

Further, any sequences of symbols and punctuation (also with spaces in between) must automatically be surrounded by **<me>**…**</me>**. That way such sequences are insulated from bidi mirroring.

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).

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, 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 character encoded.
            Symbols that are not letter-like have only one style: s="E", no matter what the s attribute is.   -->

<!ENTITY % st "(#PCDATA | st)"
            -- Note that this declares an entity %st; which is different from the element <st>
```

```
         The PCDATA must be a num, id or sym as per above, nothing beyond except ignored whitespace.
         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)"
        -- 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 "(%around;, lss)*, (%around; | %frac; |mx), (rss, %around;)*"
        --  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"      a 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     a 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, cc: use true vertical centre for math centre,
         c0: use bottom row centre as vertical math centre for the matrix, c1: …
         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>.

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 **MEO**…**MEC** 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 was we cannot foresee. However, if there are explicit **MEO**…**MEC** (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 examples

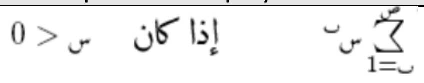
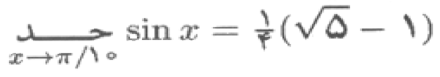
LaTeX	Proposed control code representation (C1 version)
\\(x^2 + y^2 = z^2\\)	**MEO** x **SUPR** 2 + y **SUPR** 2 = z **SUPR** 2 **MEC**
\\(E=mc^2\\)	**MEO** E=m c **SUPR** 2 **MEC** (note the important space between m and c)
\\(\sqrt{x^2+1}\\)	**MEO** √ **LSS MEO** x **SUPR** 2+1 **MEC MEC**

12.2 Some more examples (C1 and HTML compat. versions)

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
$f(x) = \begin{cases} \sum_{i=1}^{s} x^i & \text{if } x < 0 \\ \int_1^s x^i \, dx & \text{if } x \in \mathrm{S} \\ \tan \pi & \text{otherwise (with } \pi \simeq 3.141) \end{cases}$	**MEO** f(x)={**LSS SCI ;** ∑ **BLW MEO** i=1 **MEC ABV** s **LSSV** x **SUPR** i **HT SCI IS1 TXTO** if **SP TXTC** x<0 **LF** **MIT** ∫ **SUBR** 1 **SUPR** s **LSSV** x **SUPR** i **NNBSP** d x **HT TXTO** if **SP TXTC** x∈ **MSU** S **LF** tan **SP** π **HT TXTO** otherwise **SP** (with **SP MEO** π≃3.141 **MEC**) **TXTC SCI # MEC** (spaces and line breaks are for readability only), or **<me>** f(x)={**<lss/><mx c='cc'>** **<mr><mc>**∑**<blw/><me>**i=1**</me>** **<abv/>**s**<lss s='v'/>**x**<rsp/>**i**</mc>** **<mc a='18'><txt>**if **</txt>**x<0 **</mc></mr>** **<mr><mc><st s='C'>**∫**</st><rsb/>** 1 **<rsp/>**s**<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>**

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
	**MEO** 0 > س **NBSP NBSP TXTO** إذاكان **TXTC** **NBSP NBSP NBSP MEO** ب **SUPL** س **MEC RSS** **MEO MMS** ∑ **BLW MEO** 1=ب **MEC ABV** ص **MEC** or **<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>**
	**MEO** حد **BLW MEO** x ⟶ π/١٠ **MEC** sin x=١ **MHD** ۴ (√ **LSS** ۵ - ١) **MEC** or **<me>** حد **<blw/><me>**x ⟶ π/١٠**</me>**sin x=١ **<dv/>** ۴ (√ **<lss/>** ۵ - ١) **</me>**

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 ….) 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.

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 and HTML(5), 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 other documents (using one or the other representation proposed here) where math expressions are given.

14 REFERENCES

[AsciiMath]	*ASCII Math*, http://asciimath.org/.
[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.txt]	http://www.unicode.org/Public/UNIDATA/BidiMirroring.txt.
[ECMA-48]	*Control Functions for Coded Character Sets*, 5th ed., June 1991, https://www.ecma-international.org/publications/files/ECMA-ST/Ecma-048.pdf. (Earlier editions as well.)
[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.txt]	Draft in https://www.unicode.org/L2/L2022/22026r-non-bidi-mirroring.pdf.
[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://standards.iso.org/ittf/PubliclyAvailableStandards/c069119_ISO_IEC_10646_2017.zip. Same coded characters as in [Unicode].
[LaTeX]	*The LaTeX Companion*, Goossens, Mittelbach, Samarin, 1994.
[LaTeXW]	*The LaTeX Web Companion*, Michel Goossens, Sebastian Rahtz, 1999.
[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).
[MathML]	*MathML* – World Wide Web Consortium (W3C), https://www.w3.org/Math/.
[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 – MMS mirroring

The following symbols can be mirrored by **MMS** (or the corresponding attribute in the HTML/XML 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: 1)
⨤	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 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 TXTO...TXTC apart from embedded MEO...MEC. */

/* 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_MEO is for TXTO...TXTC content.   */
%token ID NUM SYM char-non-meo

/* Controls for math expression layout.                                          */
/* The lexer needs to map certain character sequences to these tokens.           */
%token MEO MEC PHA VPHA RECT SP HT HTJ NLF STYLE BINOP NONBINOP MMS TXTO TXTC
%token BLW BLWS BLWT BLWU ABV ABVS ABVT ABVU SUPL SUBL SUBR SUPR
%token LSS LSST LSSU LSSV RSS RSST RSSU RSSV MHD U2215 U29F5 U29F8 U229F9
%token SCIHT SCIVT SCISP SCIIS1 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 TXTO...TXTC around strings of no-break spaces. */
me : MEO /* Push 'math expression' lexing state for lexer */
       phantom rectangular ws me-seq
     MEC /* Pop 'math expression' lexing state for lexer */ ;
me-seq : me-seq stretch ws | ;
phantom : PHA | VPHA | ;
rectangular : RECT | ;
ws : ws SP | ws HT | ws NLF | ;

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 MEO...MEC around strings of symbols,  */
/* including punctuation and spaces, except pure spaces.                */
/* Like elsewhere in math expressions, bidi controls are not permitted. */
txt : TXTO /* Push 'embedded text' lexing state for lexer */
        texts
      TXTC /* Pop 'embedded text' lexing state for lexer */ ;
texts : texts char-non-meo | texts me | ;

closed : me | st | txt ;
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 | U229F9 ;
vc-adjf : SCI0 | SCI1 | ;

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

/* 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 SCISP longer mc
   | mr SCIIS1 longer mc
   | mr SCIBS shorter mc
   | ;
longer : longer SCISP | longer SCIIS1 | ;
shorter : shorter SCIBS | ;

mrs : NLF mr
    | NLF PLD plds mr
    | 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 – “markdown” style 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 C1 control codes. The XML/HTML 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 C0/C1 control codes nor XML/HTML 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 (between an outermost **MEO** and an outermost **MEC**):

- **${** for outermost **MEO**, **{** for non-outermost **MEO**, **\{** for {,
- **}** for **MEC**, **\}** for },
- **$** for **SCI** (note: these cover all the math style controls), **$$** for $,
- \\ for \,
- **\/** for **MHD**,
- **\HT** for **HTJ**, **$HT** for **SCI HT**, **$VT** for **SCI VT** (**VT** is usually input via *shift-return*),
- **\&** for **SCI IS1**,
- **\%** for **MMS**,
- **$^** or ^ for **SCI ^**, **\^** for ^,
- **$_** or **_** for **SCI _**, **_** for _,
- **$[**[**.,:;**] for **SCI**[**.,:;**] (rather than just **$**[**.,:;**]),
- **$]** for **SCI #** if within a math matrix (if started with **$[**[**.,:;**], otherwise **$#**).

It will have all the quirks of the C1 version (apart from the use of C1 control characters), but input from a common keyboard is possible. Special symbols may be inputable via a palette, or by using name references, e.g., \alpha. Unlike most other (non-math expression) markdowns, this is fully equivalent to the nominal formats.

Note that this is just a hint, not part of the proposed formats.