# Proposed Draft Unicode Technical Report #25

L2/01-449

# *Unicode Support for Mathematics*

| | |
|---|---|
| Date | 2001-10-10 |
| Authors | Barbara Beeton (bnb@ams.org), Asmus Freytag (asmus@unicode.org), Murray Sargent III (murrays@microsoft.com) |
| Previous version | none |
| This version | |
| Latest version | |
| Version | 1 [This is the last word file before conversion to html] |

**Summary**

*Starting with version 3.2, Unicode includes virtually all of the standard characters used in mathematics. This set supports a variety of math applications on computers, including document presentation languages like TeX, math markup languages like MathML, computer algebra languages like OpenMath, internal representations of mathematics in systems like Mathematica and MathCAD, computer programs, and plain text. This technical report describes the Unicode mathematics character groups and gives some of their default math properties.*

**Status**

*This document has been approved by the Unicode Technical Committee for public review as a* ***Proposed Draft Unicode Technical Report****. Publication does not imply endorsement by the Unicode Consortium. This is a draft document which may be updated, replaced, or superseded by other documents at any time. This is not a stable document; it is inappropriate to cite this document as other than a work in progress.*

*Please send comments to the authors. A list of current Unicode Technical Reports is found on http://www.unicode.org/unicode/reports/. For more information about versions of the Unicode Standard, see http://www.unicode.org/unicode/standard/versions/.*

*The References provide related information that is useful in understanding this document. Please mail corrigenda and other comments to the author(s).*

**Contents**

# 1 Overview

This technical report starts with a discussion of the mathematics character repertoire incorporating the relevant block descriptions of The Unicode Standard [TUS]. Associated character properties are discussed next, including a number of properties that are not yet

part of the Unicode Standard. Character classifications by usage, by typography, and by precedence are given. Some implementation guidelines for input methods and use of Unicode math characters in programming languages are presented next. The final section describes how many mathematical expressions can be rendered using a plain—or at least nearly plain—text format. Mathematical plain text can be handy for down-level text copies, e.g., in email, math input methods, computer programs, and in-line math display. Most mathematical expressions up through calculus can be represented unambiguously in Unicode plain text. Note that the discussion is only intended to show how mathematical plain text might be useful. It is not intended to be a complete specification or to be used for general information interchange at this stage in its development.

# 2 Mathematical Character Repertoire

Unicode 3.2 provides a quite complete set of standard math characters to support publication of mathematics on and off the web. Specifically, Unicode 3.1 introduced 996 new alphanumeric symbols and Unicode 3.2 introduces 591 new symbols, in addition to the 340 math specific symbols already encoded in Unicode 3.0 for a total of 1927 mathematical symbols. This repertoire is the result of input from many sources, notably from the STIX project, and enables the display of virtually all standard mathematical symbols. Nevertheless this work must needs be remain incomplete; mathematicians and other scientists are continually inventing new mathematical symbols and the plan is to add them as they become accepted in the scientific communities.
MathML is a major beneficiary of the increased repertoire for mathematical symbols and the working group lobbied in favor of the inclusion of the new characters. In addition, the new characters lend themselves to a useful plain text encoding of mathematics (see Sec. 4) that is much more compact than MathML or TeX.

## *2.1 Mathematical Alphanumeric Symbols Block*

The Mathematical Alphanumeric Symbols block (U+1D400–U+1D7FF) contains a large extension of letterlike symbols used in mathematical notation, typically for variables. The characters in this block are intended for use only in mathematical or technical notation; they are not intended for use in non-technical text. When used with markup languages, for example with MathML *Mathematical Markup Language (MathML™)* the characters are expected to be used directly, instead of indirectly via entity references or by composing them from base letters and style markup.
***Words Used as Variables***. In some specialties, whole words are used as variables, not just single letters. For these cases, style markup is preferred because in ordinary mathematical notation the juxtaposition of variables generally implies multiplication, not word formation as in ordinary text. Markup not only provides the necessary scoping in these cases, it also allows the use of a more extended alphabet.

## *2.2 Mathematical Alphabets*

***Basic Set of Alphanumeric Characters***. Mathematical notation uses a basic set of mathematical alphanumeric characters which consists of:

- the set of basic Latin digits (0 - 9) (U+0030..U+0039)
- the set of basic upper- and lowercase Latin letters (a - z, A - Z)
- the uppercase Greek letters A - Ω (U+0391..U+03A9), plus the nabla ∇ (U+2207) and the variant of theta Θ given by U+03F4
- the lowercase Greek letters α - ω (U+03B1..U+03C9), plus the partial differential sign ∂ (U+2202) and the six glyph variants of ε, θ, κ, φ, ρ, and π, given by U+03F5, U+03D1, U+03F0, U+03D5, U+03F1, and U+03D6.

Only unaccented forms of the letters are used for mathematical notation, because general accents such as the acute accent would interfere with common mathematical diacritics. Examples of common mathematical diacritics that can interfere with general accents are the circumflex, macron, or the single or double dot above, the latter two of which are used in physics to denote derivatives with respect to the time variable. Mathematical symbols with diacritics are always represented by combining character sequences, except as required by normalization. See Unicode Standard Annex #15, Unicode Normalization Forms for more information.

For some characters in the basic set of Greek characters, two variants of the same character are included. This is because they can appear in the same mathematical document with different meanings, even though they would have the same meaning in Greek text.

***Additional Characters***. In addition to this basic set, mathematical notation also uses the four Hebrew-derived characters (U+2135..U+2138). Occasional uses of other alphabetic and numeric characters are known. Examples include U+0428 CYRILLIC CAPITAL LETTER SHA, U+306E HIRAGANA LETTER NO, and Eastern Arabic-Indic digits (U+06F0..U+06F9). However, these characters are used in only the basic form.

***Semantic Distinctions.*** Mathematics has need for a number of Latin and Greek alphabets that on first thought appear to be mere font variations of one another. For example the letter H can appear as plain or upright (H), bold (**H**), italic (*H*), and script $\mathcal{H}$. However in any given document, these characters have distinct, and usually unrelated mathematical semantics. For example, a normal H represents a different variable from a bold **H**, etc. If these attributes are dropped in plain text, the distinctions are lost and the meaning of the text is altered. Without the distinctions, the well-known Hamiltonian formula

$$\mathcal{H} = \int d\tau (\varepsilon E^2 + \mu H^2),$$

turns into the *integral* equation in the variable H

$$H = \int d\tau (\varepsilon E^2 + \mu H^2).$$

By encoding a separate set of alphabets, it is possible to preserve such distinctions in plain text.

***Mathematical Alphabets***. The alphanumeric symbols encountered in mathematics are given in the following table:

| Math style | Characters from basic set | Plane |
|---|---|---|

| | | |
|---|---|---|
| plain (upright, serifed) | Latin, Greek and digits | BMP |
| bold | Latin, Greek and digits | Plane 1 |
| italic | Latin and Greek | Plane 1* |
| bold italic | Latin and Greek | Plane 1 |
| script (calligraphic) | Latin | Plane 1* |
| bold script (calligraphic) | Latin | Plane 1 |
| fraktur | Latin | Plane 1* |
| bold fraktur | Latin | Plane 1 |
| double-struck | Latin and digits | Plane 1* |
| sans-serif | Latin and digits | Plane 1 |
| sans-serif bold | Latin, Greek and digits | Plane 1 |
| sans-serif italic | Latin | Plane 1 |
| sans-serif bold italic | Latin and Greek | Plane 1 |
| monospace | Latin and digits | Plane 1 |

* Some of these alphabets have characters in the BMP as noted in the following section.

The plain letters have been unified with the existing characters in the Basic Latin and Greek blocks. There are 25 double-struck, italic, Fraktur and script characters that already exist in the Letterlike Symbols block (U+2100 – U+214F). These are explicitly unified with the characters in this block and corresponding holes have been left in the mathematical alphabets.

***Compatibility Decompositions.*** All mathematical alphanumeric symbols have compatibility decompositions to the base Latin and Greek letters -- folding away such distinctions, however, is usually not desirable as it loses the semantic distinctions for which these characters where encoded. See Unicode Standard Annex #15, Unicode Normalization Forms for more information.

## 2.3 Fonts Used for Mathematical Alphabets

Mathematicians place strict requirements on the *specific* fonts being used to represent mathematical variables. Readers of a mathematical text need to be able to distinguish single letter variables from each other, even when they do not appear in close proximity. They must be able to recognize the letter itself, whether it is part of the text or is a mathematical variable, and lastly which mathematical alphabet it is from.

***Fraktur.*** The black letter style is often referred to as *Fraktur* or *Gothic* in various sources. Technically, Fraktur and Gothic typefaces are distinct designs from black letter, but any of several font styles similar in appearance to the forms shown in the charts can be used.

***Math italics.*** Mathematical variables are most commonly set in a form of italics, but not all italic fonts can be used successfully. In common text fonts, the *italic letter v* and *Greek letter nu* are not very distinct. A rounded *italic letter v* is therefore preferred in a mathematical font. There are other characters, which sometimes have similar shapes and require special attention to avoid ambiguity. Examples are shown in the table below.

Theorems are commonly printed in a text italic font. A font intended for mathematical variables should support clear visual distinctions so that variables can be reliably separated from italic text in a theorem. Some languages have common single letter words (English *a*, Scandinavian *i*, etc.), which can otherwise be easily confused with common variables.

***Hard-to-distinguish Letters***. Not all sans-serif fonts allow an easy distinction between *lowercase l*, and *uppercase I* and not all monospaced (monowidth) fonts allow a distinction between the *letter* `l` and the *digit* `1`. Such fonts are not usable for mathematics. In Fraktur, the letters 𝔍 and 𝔍 in particular must be made distinguishable. Overburdened Black Letter forms like 𝕴 and 𝕵 are inappropriate. Similarly, the *digit zero* must be distinct from the *uppercase letter O* for all mathematical alphanumeric sets. Some characters are so similar that even mathematical fonts do not attempt to provide distinguished glyphs for them, e.g. *uppercase A* and *uppercase Alpha* (A). Their use is normally avoided in mathematical notation unless no confusion is possible in a given context,

***Font Support for Combining Diacritics.*** Mathematical equations require that characters be combined with diacritics (dots, tilde, circumflex, or arrows above are common), as well as followed or preceded by super- or subscripted letters or numbers. This requirement leads to designs for *italic* styles that are less inclined, and *script* styles that have smaller overhangs and less slant than equivalent styles commonly used for text such as wedding invitations.

***Typestyle for Script Characters.*** In some instances, a deliberate unification with a non-mathematical symbol has been undertaken; for example, U+2133 is unified with the pre-1949 symbol for the German currency unit *Mark* and U+2113 is unified with the common non-SI symbol for the liter. This unification restricts the range of glyphs that can be used for this character in the charts. Therefore the font used for the reference glyphs in the code charts uses a simplified ‘English Script’ style, as per recommendation by the American Mathematical Society. For consistency, other script characters in the Letterlike Symbols block are now shown in the same typestyle.

***Double-struck Characters.*** The double-struck glyphs shown in earlier editions of the standard attempted to match the design used for all the other Latin characters in the standard, which is based on Times. The current set of fonts was prepared in consultation with the American Mathematical Society and leading mathematical publishers, and shows much simpler forms that are derived from the forms written on a blackboard. However, both serifed and non-serifed forms can be used in mathematical texts, and inline fonts are found in works published by certain publishers. There is no intention to support such stylistic preference via character encoding, therefore only one set of doublestruck mathematical alphanumeric symbols have been encoded.

[ED: this marks the end of the material included in Unicode 3.1]

## 2.4 Locating Mathematical Characters

Mathematical characters can be located by looking in the blocks that contain such characters or by checking the Unicode MATH property, which is assigned to characters that naturally appear in mathematical contexts (see Sec. “Mathematical Character Properties”). Mathematical characters can be found in the following blocks:

**Table 2.1 Locations of Mathematical Characters**

| Block Name | Range | Characters |
|---|---|---|
| Basic Latin | U+0021—U+007E | Variables, operators, digits* |
| Greek | U+0370—U+03FF | Variables* |
| General Punctuation | U+2000—U+206F | Invisible operators* |
| Letterlike Symbols | U+2100—U+214F | Variables* |
| Arrows | U+2190—U+21FF | Arrows, arrow-like operators |
| Mathematical Operators | U+2200—U+22FF | Operators |
| Miscellaneous Technical Symbols | U+2300—U+23FF | Braces, operators* |
| Geometrical Shapes | U+25A0—U+25FF | Symbols |
| Misc. Mathematical Symbols –A | U+27C0—U+27EF | Symbols and operators |
| Supplemental Arrows-A | U+27F0—U+27FF | Arrows, arrow-like operators |
| Supplemental Arrows-B | U+2900—U+297F | Arrows, arrow-like operators |
| Misc. Mathematical Symbols –B | U+2980—U+29FF | Braces, symbols |
| Suppl. Mathematical Operators | U+2A00—U+2AFF | Operators |
| Mathematical Alphanumeric Symbols | U+1D400—U+1D7FF | Variables and digits |
| Other blocks | … | Characters for occasional use |

*This block contains nonmathematical characters as well.

## 2.5 Duplicated Characters

Some Greek letters are re-encoded as technical symbols. These U+00B5 µ MICRO SIGN, U+2126 Ω OHM SIGN, and several characters among the APL functional symbols in the Miscellaneous Technical block. U+03A9 GREEK LETTER CAPITAL OMEGA is the canonical equivalent of U+2126 and its use is preferred. Latin letters duplicated include 212A KELVIN SIGN and U+212B ANGSTROM SIGN. Like the case for the *ohm sign* the corresponding regular Latin letters are the canonical equivalents and therefore their use is preferred.

The *left* and *right angle brackets* at 2328 and 2329 have long been canonically equivalent with the CJK punctuation characters at 3008 and 3009, which implies that the use of the latter code points is preferred and that the characters are 'wide' characters. See Unicode Standard Annex #11, East Asian Width. Unicode 3.2 adds two new *angle bracket* characters that are unequivocally intended for mathematical use.

## 2.6 Accented Characters

Mathematical characters are often enhanced via use of combining marks in the ranges U+0300 – U+036F and the combining marks for symbols in the range U+20D0 – U+20FF. These characters follow the base characters as in nonmathematical Unicode text. This section discusses these characters and preferred ways of representing accented characters in mathematical expressions. If a span of characters is enhanced by a combining mark, e.g., a tilde over AB, typically some kind of higher-level markup is needed as is done in MathML. Unicode does include some combining marks that are designed to be used for pairs of characters, e.g., U+0360 through U+0362. However, their use for mathematical text is not encouraged.

For some Mathematical characters there are multiple ways of the character: as precomposed or a as sequence of base character and combining mark. It would be nice to have a single way to represent any given character, since this would simplify recognizing the character in searches and other manipulations. Selecting a unique representation among multiple equivalent representations is called normalization. Unicode Technical Report 15 discusses the subject in detail, however, due to requirements of non-mathematical software, the normalization forms presented there are not ideal from the perspective of mathematics.

Ideally, on always uses the shortest form of a math operator symbol wherever possible. So U+2260 should be used for the not equal sign instead of the combining sequence U+003D U+0338. This rule concurs with Normalization Form C (NFC) used on the web. If a negated operator is needed that does not have a precomposed form, the character U+0338 COMBINING LONG SOLIDUS OVERLAY can be used to indicate negation. On the other hand, for accented *alphabetic* characters used as variables, ideally only decomposed sequences are used since there are no precomposed math alphanumerical symbols.

Mathematics uses a multitude of combining marks that greatly exceeds the predefined composed characters in Unicode. Accordingly, it is better to have the math display facility handle all of these cases uniformly to give a consistent look between characters that happen to have a fully composed Unicode character and those that do not. The combining character sequences also typically have semantics as a group, so it is handy to be able to manipulate and search for them individually without having to have special tables to decompose characters for this purpose. Note that this approach does not concur with Normalization Form C for the upright alphabetic characters (ASCII letters). To facilitate interchange on the Web, accented characters should conform to NFC when interchanged. However, to achieve consistent results, a mathematical display system should transiently decompose such letters when used in mathematical expressions and use a single algorithm to place embellishments.

## 2.7 Operators

The Unicode blocks U+2200 – U+22FF and U+2A00 – U+2AFF contain many mathematical operators, relations, geometric symbols and other symbols with special usages confined largely to mathematical contexts. In addition to the characters in these blocks, mathematical operators are also found in the Basic Latin (ASCII) and Latin-1 Supplement Blocks. A few of the symbols from the Miscellaneous Technical block and characters from General Punctuation are also used in mathematical notation.

Mathematical operators often have more than one meaning. Therefore the encoding of these blocks is intentionally rather shape based, with numerous instances in which several semantic values can be attributed to the same Unicode value. For example, U+2218 2218 ∘ RING OPERATOR may be the equivalent of *white small circle* or *composite function* or *apl jot*. The Unicode Standard does not attempt to distinguish all possible semantic values that may be applied to mathematical operators or relational symbols.

The Standard does include many characters that appear to be quite similar to one another, but that may well convey different meaning in a given context. On the other hand, mathematical operators, and especially relation symbols, may appear in various standards, handbooks, and fonts with a large number of purely graphical variants. Where variants were recognizable as such from the sources, they were not encoded separately. For relation symbols, the choice of a vertical or forward-slanting stroke typically seems to be an aesthetic one, but both slants might appear in a given context. However, a back-slanted stroke has almost always a distinct meaning compared to the forward-slanted stroke. See Section 13.xx "Variation Selectors" for more information on some particular variants.

***Unifications.*** Mathematical operators such as *implies* ⇔ and *if and only if* ↔ have been unified with the corresponding arrows (U+21D2 RIGHTWARDS DOUBLE ARROW and U+2194 LEFT RIGHT ARROW, respectively) in the Arrows block.

The operator U+2208 element of is occasionally rendered with a taller shape than shown in the code charts. Mathematical handbooks and standards consulted treat these characters as variants of the same glyph. U+220A small element of is a distinctively small version of the *element of* that originates in mathematical pi fonts.

The operators U+226B much greater-than and U+226A much less-than are sometimes rendered in a nested shape. Because no semantic distinction applies, the Unicode Standard provides a single encoding for each operator.

A large class of unifications applies to variants of relation symbols involving equality, similarity, and/or negation. Variants involving one- or two-barred *equal signs*, one- or two-*tilde similarity signs*, and vertical or slanted *negation slashes* and *negation slashes* of different lengths are not separately encoded. Thus, for example, U+2288 neither a subset of nor equal to, is the archetype for at least six different glyph variants noted in various collections.

In two instances, essentially stylistic variants are separately encoded: U+2265 greater-than or equal to is distinguished from U+2267 greater-than over equal to; the same distinction applies to U+2264 less-than or equal to and U+2266 less-than over equal to. This exception to the general rule regarding variation results from requirements for character mapping to some Asian standards that distinguish the two forms.

Several mathematical operators derived from Greek characters have been given separate encodings since they are used differently than the corresponding letters. These operators

may occasionally occur in context with Greek-letter variables. They include U+2206 Δ INCREMENT, U+220F ∏ N-ARY PRODUCT, and U+2211 ∑ N-ARY SUMMATION. The latter two are large operators that take limits. Some typographical aspects of operators are discussed in Section 3.2 “Classification by Typographical Behavior”. For example, the n-ary operators are distinguished from letter variables by their larger size and the fact that they take limit expressions.

The unary and binary minus sign is preferably represented by U+2212 MINUS SIGN rather than by U+002D HYPHEN-MINUS, both because the former is unambiguous and because it is rendered with a more desirable length. (For a complete list of dashes in the Unicode Standard, see *Table 6-2* in TUS). U+22EE – U+22F1 are a set of ellipses used in matrix notation.

***Miscellaneous Symbols*** U+2212 – minus sign is a mathematical operator, to be distinguished from the ASCII-derived U+002D - hyphen-minus, which may look the same as a minus sign, or may be shorter in length. (For a complete list of dashes in the Unicode Standard, see Table 6-2.) U+22EE..U+22F1 are a set of ellipses used in matrix notation.

## 2.8 Superscripts and Subscripts

The Unicode block U+2070 – U+209F plus U+00B2, U+00B3, and U+00B9 contain sequences of superscript and subscript digits and punctuation that can be useful in mathematics. If they are used, it is recommended that they be displayed with the same font size as other subscripts and superscripts at the corresponding nested script level. For example in the Unicode plain text approach of Sec. 4, $a^2$ and $a^{\uparrow}2$ should be displayed the same way when built up. These subscript/superscript characters are not used in MathML and TeX and their use with XML documents is discouraged, see Unicode Technical Reports #20, “Unicode and XML”.

## 2.9 Arrows

Arrows are used for a variety of purposes in mathematics and elsewhere, such as to imply directional relation, to show logical derivation or implication, and to represent the cursor control keys. Accordingly Unicode includes a fairly extensive set of arrows (U+2190 – U+21FF and U+2900 – U+297F), many of which appear in mathematics. It does not attempt to encode every possible stylistic variant of arrows separately, especially where their use is mainly decorative. For most arrow variants, the Unicode Standard provides encodings in the two horizontal directions, often in the four cardinal directions. For the single and double arrows, the Unicode Standard provides encodings in eight directions.

***Unifications*** Arrows expressing mathematical relations have been encoded in the arrows block as well as in Supplemental Arrows-A and Supplemental Arrows-B. An example is U+21D2 _ rightwards double arrow, which may be used to denote *implies*. Where available, such usage information is indicated in the annotations to individual characters in Chapter 14, *Code Charts*.

## 2.10 Geometrical Shapes

The basic geometric shapes (circle, square, triangle, diamond, and lozenge) are used for a variety of purposes in mathematical texts. Because their shapes are distinct and they are easily available in multiple sizes from a variety of widely available fonts, they are also often used in an ad-hoc manner.

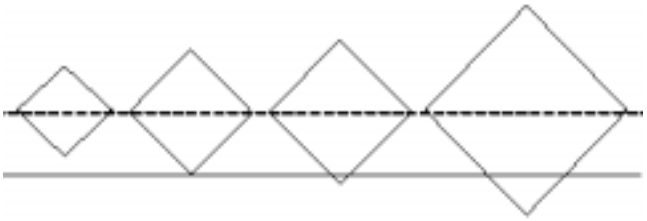
**Ideal sizes**. Mathematical usage requires at least four distinct sizes of simple shapes, and sometimes more. The size gradation must allow each size to be recognized, even when it occurs in isolation. In other words shapes of the same size should ideally have roughly the same visual “impact ” as opposed to same nominal height or width. For mathematical usage simple shapes ideally share a common center. The following diagram shows which size relationship across shapes of the same nominal size is considered ideal.

Please note that neither the current set of glyphs in the standard nor the glyphs from many commonly available non-mathematical fonts show this kind of size relation.

**Actual sizes**. The sizes of existing characters and their names are not always consistent. For mathematical usage, therefore, the MEDIUM SMALL SQUARE should be used together with the MEDIUM size of the other basic shapes, and correspondingly for the other sizes.(The basic shapes from the Zapf Dingbats font match the unmarked size for triangle, diamond and circle and the MEDIUM size for the square.) To achieve the correct size relation, mathematical fonts may need to deviate in minor amounts from the sizes shown in the character charts. [TBD: summary picture]

**Sizes of derived shapes.** Circled and squared operators and similar derived shapes are more constrained in their usage than 'plain' geometric shapes. They tend to occur in two generic sizes based on function: a smaller size for operators and large size for n-ary operators.

**Positioning.** For a mathematical font, the centerline should go through the middle of a parenthesis, which should go from bottom of descender to top of ascender. This is the same level as the minus or the middle of the plus and equal signs. For correct positioning, the glyph will descend below the baseline for the larger sizes of the basic shapes as in the following schematic diagram:



The standard triangles used for mathematics are also center aligned. This is different from the positioning for the reference glyphs of existing characters shown in the charts. Mathematical fonts may need to deviate in positioning of these triangles.

## *2.11 Other Symbols*

Other symbols of use in mathematics are contained in the Miscellaneous Technical block (U+2300–U+23FF), the Geometric Shapes block (U+25A0–U+25FF), the Miscellaneous Symbols block (U+2600–U+267F), and the General Punctuation block (U+2000–U+206F).

Generally any easily recognized and distinct symbol is fair game for mathematicians faced with the need of creating notations for new fields of mathematics. For example, the card suits ♥,♠, *etc.* can be found as operators as well as subscripts.

## 2.12 Symbol Pieces

Fences are ordinarily sized to the content that they enclose. However, in creating a large fence, the glyph is not scaled proportionally, in particular the displayed stem weights must remain compatible with the accompanying smaller characters. Thus, simple scaling of font outlines cannot be used to create tall brackets. Instead, a common technique is to build up the symbol from pieces. In particular, the characters U+239B through U+23AD Miscellaneous Technical block contains a set of pieces for building up large versions of the fences (, ), [, ], {, and }, and of the large operators ∑, and ∫. These brace and operator pieces are not used in stored mathematical text, but are often used in the data stream created by display and print drivers. Some of these symbol pieces also occur in legacy character sets.

The following table shows which pieces are intended to be used together for what symbol.

| | 2-row | 3-row | 5-row |
|---|---|---|---|
| Summation | 23B2, 23B3 | | |
| Integral | 2320, 2321 | 2320, 23AE, 2321 | 2320, 3×23AE, 2321 |
| Left Parenthesis | 239B, 239D | 239B, 239D | 239B, 3×239C, 239D |
| Right Parenthesis | 239E, 23A0 | 239E, 239F, 23A0 | 239E, 3×239F, 23A0 |
| Left Bracket | 23A1, 23A3 | 23A1, 23A2, 23A4 | 23A1, 3×23A2, 23A3 |
| Right Bracket | 23A4, 23A6 | 23A4, 23A5, 23A6 | 23A4, 3×23A5, 23A6 |
| Left Brace | 23B0, 23B1 | 23A7, 23A8, 2389 | 23A7, 23AA, 23A8, 23AA, 2389 |
| Right Brace | 23B1, 23B0 | 23AB, 23AC, 23AD | 23AB, 23AA, 23AC, 23AA, 23AD |

**Table 2.2 Use of symbol pieces**

For example, an instance of U+239B can be positioned relative to instances of U+239C and U+2390 to form an extra-tall (three or more line) left-parenthesis. The center sections encoded here are meant to be used only with the top and bottom pieces encoded adjacent to them, since the segments are usually graphically constructed within the fonts so that they match perfectly when positioned at the same *x* coordinates.

## 2.13 Invisible Operators

In mathematics some operators or punctuation are often implied, but not displayed. U+2063 INVISIBLE SEPARATOR or *invisible comma* is intended for use in index expressions and other mathematical notation where two adjacent variables form a list and are not implicitly multiplied. In mathematical notation, commas are not always explicitly present, but need to be indicated for symbolic calculation software to help it disambiguate a sequence from a multiplication. For example, the double $_{ij}$ subscript in the variable $a_{ij}$

means $a_{i,j}$, i.e., the *i* and *j* are separate indices and not a single variable with the name *ij* or even the product of *i* and *j*. Accordingly to represent the implied list separation in the subscript $_{ij}$ one can insert a nondisplaying *invisible separator* between the *i* and the *j*. In addition, use of the invisible comma would hint to a math layout program to typeset a small space between the variables.
Similarly an expression like $mc^2$ implies that the mass *m* multiplies the square of the speed *c*. To represent the implied multiplication in $mc^2$, one inserts a nondisplaying U+2061 INVISIBLE TIMES between the *m* and the *c*. A related case is the use of U+2062 FUNCTION APPLICATION for an implied function dependence as in $f(x + y)$. To indicate that this is the function *f* of the quantity $x + y$ and not the expression $fx + fy$, one can insert the nondisplaying *function application symbol* between the *f* and the left parenthesis.
Another example is the expression $f^{i,j}(\cos(ab))$, which means the same as $f^{i,j}(\cos(a \times b))$, where × represents *multiplication*, not the *cross product.* Note that the spacing between characters may also depend on whether the adjacent variables are part of a list or are to be concatenated, i.e., multiplied.

## 2.14 Other Characters

These include all remaining Unicode characters. They may appear in mathematical expressions, typically in spelled-out names for variables in fractions or simple formulae, but they most commonly appear in ordinary text. An English example is the equation

distance = rate × time,

which uses ordinary ASCII letters to aid in recognizing sequences of letters as words instead of products of individual symbols. Such usage corresponds to identifiers, discussed elsewhere.

## 2.15 Variation Selector

The variation selector VS1 is used to represent well-defined variants of particular math symbols. The variations include: different slope of cancellation element in some negated symbols, changed orientation of an equating or tilde operator element, and some well-defined different shapes. To select one of the predefined variation follow the base by the variation selector. The only characters defined for use with the variant selector are the ones given in the following table.
It is important to not that the variation selector only selects a different *appearance* of an already encoded character. It is not intended as a general code extension mechanism. At this time the variations encoded with the variation selector are thought to be primarily glyphic variation. Should their usage or interpretation change – over time, or because of better evidence about how these shapes are actually used in mathematical notation – it is likely that another character would be coded so that the distinction in meaning can be kept directly in the character code.
In extremis, the Unicode Standard considers the variation selector somewhat optional. Processes or fonts that cannot support it should yield acceptable results by ignoring the variation selector.

| | |
|---|---|
| 2268 + VS1 | Less-than and not double equal - with vertical stroke |
| 2269 + VS1 | Greater-than and not double equal - with vertical stroke |
| 22DA + VS1 | Less-than above slanted equal above greater-than |
| 22DB + VS1 | Greater-than above slanted equal above less-than |
| 2272 + VS1 | Less-than or similar - following the slant of the lower leg |
| 2273 + VS1 | Greater-than or similar - following the slant of the lower leg |
| 2A9D + VS1 | similar - following the slant of the upper leg - or less-than |
| 2A9E + VS1 | similar - following the slant of the upper leg - or greater-than |
| 2AAC + VS1 | Smaller than or slanted equal |
| 2AAD + VS1 | larger than or slanted equal |
| 228A + VS1 | subset not equals - variant with stroke through bottom members |
| 228B + VS1 | superset not equals - variant with stroke through bottom members |
| 2ACB + VS1 | subset not two-line equals - variant with stroke through bottom members |
| 2ACC + VS1 | superset not two-line equals - variant with stroke through bottom members |
| 2A3B + VS1 | Interior product - tall variant with narrow foot |
| 2A3C + VS1 | righthand interior product - tall variant with narrow foot |
| 2295 + VS1 | circled plus with white rim |
| 2297 + VS1 | circled times with white rim |
| 229C + VS1 | equal sign inside and touching a circle |
| 2225 + VS1 | Slanted parallel |
| 2225 + VS1 + 20E5 | Slanted parallel with reverse slash |
| 222A + VS1 | union with serifs |
| 2229 + VS1 | intersection with serifs |
| 2293 + VS1 | Square intersection with serifs |
| 2294 + VS1 | Square union with serifs |

### *2.16 Novel Symbols not yet in Unicode*

Mathematicians are by their nature inventive people and will continue to invent new symbols to express their theories. Until these symbols are used by a number of people, they should not be standardized. Nevertheless, one needs a way to handle these novel symbols even before they are standardized.

The private use area (0xE000 – 0xF8FF) can be used for such nonstandard symbols. It is a tricky business, since the PUA is used for many purposes. Hence when using the PUA, it is a good idea to have higher-level backup to define what kind of characters are involved. If they are used as math symbols, it would be good to assign them a math attribute that is maintained in a rich-text layer parallel to the plain text.

# 3 Mathematical Character Properties

Unicode assigns a number of mathematical character properties to aid in the default interpretation and rendering of these characters. Such properties include the classification of characters into operator, digit, delimiter, and variables. These properties may be overridden, or explicitly specified in some environments, such as MathML, which uses specific tags to indicate how Unicode characters are used, such as <mo> for operator, <md> for one or more digits comprising a number, and <mi> for identifier. TeX is a higher-level composition system that uses implicit character semantics. In the following, these properties are described in greater detail.

In particular, many Unicode characters nearly always appear in mathematical expressions and are given the generic mathematics property. For example, they include the math operators in the ranges U+2200 – U+22FF and U+29B0 – U+2AFF, the math combining marks U+20D0 – U+20FF, the math alphanumeric characters (some of the Letterlike symbols and the mathematics alphanumerics range U+1D400 – U+1D7FF). Other characters may occur in mathematical usage depending on context. The math property is useful in heuristics that seek to identify mathematical expressions in plain text.

## *3.1 Classification by Usage Frequency*

Editors' note: this classification is a work in progress

### 3.1.1 Strongly Mathematical Characters

Strong mathematical characters are all characters that are primarily used for mathematical notation. This includes all characters with the math property [Sec. 4.9 of TUS] {check that this is true after extension of the properties to the new characters} with the following exceptions:

002D HYPHEN-MINUS

and the following additions {any?}

### 3.1.2 Weakly Mathematical Characters

These characters often appear in mathematical expressions, but they also appear naturally in ordinary text. They include the ASCII letters, punctuation, as well as the arrows and many of the geometric and technical shapes. The ASCII hyphen minus (U+002D) is a weakly mathematical character that may be used for the subtraction operator, but U+2212 is preferred for this purpose and looks better. Geometric shapes are frequently used as mathematical operators.

### 3.1.3 Other

All other Unicode characters. Many of these may occur in mathematical texts, though often not as part of the mathematical expressions themselves.

## *3.2 Classification by Typographical Behavior*

Math characters fall into a number of subcategories, such as operators, digits, delimiters, and identifiers (constants and variables). This section discusses some of the typographical characteristics of these subcategories. These characteristics and classifications are useful in the absence of overriding information. For example, there is at least one document that uses the letter *P* as a relational operator.

### 3.2.1 Alphabetic

In general italic Latin characters are used to represent single-character Latin variables. In contrast, mathematical function names like sin, cos, tan, tanh, etc., are represented by upright serifed text to distinguish them from products of variables. Such names should not use the math alphanumeric characters. The upright uppercase Greek are favored over the italic ones. In Europe, upright d, D, e, and i are used for the two differential, exponential, and imaginary part functionalities, respectively. In common American mathematical prac-

tice, these quantities are represented by italic quantities. Products of italicized variables have slightly wider spacing than the letters in italicized words in ordinary text.

### 3.2.2 Operators

Operators fall into one or more categories. These include:

| | |
|---|---|
| binary | some spacing around binary operators |
| unary | closer to modified character than binary operators |
| n-ary | often called “large” operators, take limits ordinarily above/below when displayed out-of-line and right to/bottom when displayed inline |
| arithmetic | includes binary and unary operators |
| logical | unary not and binary and, or, exclusive or in a host of guises |
| set-theoretic | inclusion, exclusion, in a variety of guises |
| relational | binary operators like less/greater than in many forms |

### 3.2.3 Large Operator

These include n-ary operators like summation and integration. These may expand in size to fit their associated expressions. They generally also take limits. The placement of the limits of an operator is different when they are used in-line compared to their use in displayed formulae. For example $\sum_{n=1}^{\infty} a_n$ versus

$$\sum_{n=1}^{\infty} a_n .$$

Selection of a particular layout for limit expressions is outside the scope of the Unicode Standard.

### 3.2.4 Digits

Digits include 0-9 in various styles. These have the same widths as one another.

### 3.2.5 Delimiters

Delimiters include punctuation, opening/closing delimiters such as parentheses and brackets, braces, and fences. Opening and closing delimiters and fences may expand in size to fit their associated expressions. Some bracket expressions do not appear to be “logical” to readers unfamiliar with the notation, e.g., $]x,y[$.

### 3.2.6 Fences

Fences are similar to opening and closing delimiters, but are not paired. In addition, they include “mid” delimiters, which are not opening or closing in character.

### 3.2.7 Combining Marks

Combining marks are used with mathematical alphabetic characters (see Sec. “Accented Characters”), instead of precomposed characters. Use U+0061 U+0308 for the second derivative of acceleration with respect to time, not the precomposed letter ä. On the other hand, precomposed characters are used for operators whenever they exist. Combining

slash (solidus) or vertical overlays can be used to indicate negation for operators that do not have precomposed negated forms.
Where both long and short combining marks exist, use the long, e.g., use U+0338, not U+0337 and use U+20D2, not U+20D3. The actual shape or position of a combining is a typesetting problem and not specified in plain text. When using combining marks, the composite characters have the same typesetting class as the base character.

### 3.3 Classification of Operators by Precedence

Operator precedence reduces the notational complexity of expressions and is commonly used for this purpose in computer programming languages, calculus, and algebra. A simple precedence table is used in Sec. 4-2 to convert the Unicode plain-text notation into a prefix notation used in two-dimensional display code. Although that table has some unusual precedence assignments, it shares with ordinary algebra the concept that addition and subtraction have lower precedence than multiplication and division. Some display engines, e.g., TeX's and MathML's, do not use precedence and instead rely on complete specification of operator order via explicit bracketing, either with {} as in TeX or XML tags as in MathML.
[TBD: property files that specify the actual classification]

## 4 Implementation Guidelines

### 4.1 Use of Normalization with Mathematical Text

If Normalization Form C is applied to mathematical text, some accents or overlays used with BMP alphabetic characters may be incorrectly composed with their base character. Parsers should allow for this. Normalization forms KC or KD remove the distinction between different mathematical alphabets. These forms *cannot* be used with mathematical texts. For more details on Normalization see Unicode Standard Annex #15 *Normalization* and the discussion in Sec. "Accented Characters".

### 4.2 Input of Mathematical and Other Unicode Characters

In view of the large number of characters used in mathematics, it is useful to give some discussion of input methods. The ASCII math symbols are easy to find, e.g., + - / * [ ] ( ) { }, but often need to be used as themselves.
**Post-entry correction**. From a syntax point of view, the official Unicode minus sign (U+2212) is certainly preferable to the ASCII hyphen-minus (U+002D) and the prime (U+2032) is preferable to the ASCII apostrophe (U+0027), but users may find the ASCII characters more easily. Similarly it is easier to type ASCII letters than italic letters, but when used as mathematical variables, such letters are traditionally italicized in print. Accordingly a user might want to make italic the default alphabet in a math context, reserving the right to overrule this default when necessary. Other post-entry enhancements include automatic-ligature and left-right quote substitutions, which can be done automatically by some word processors. Suffice it to say that intelligent input algorithms can dramatically simplify the entry of mathematical symbols.
**Math keyboards.** A special math shift facility for keyboard entry could bring up proper math symbols. The values chosen can be displayed on an *on-screen keyboard*. For exam-

ple, the left Alt key can access the most common mathematical characters and Greek letters, the right Alt key could access italic characters plus a variety of arrows, and the right Ctrl key could access script characters and other mathematical symbols. The numeric keypad offers locations for a variety of symbols, such as sub/superscript digits using the left Alt key. Left Alt CapsLock could lock into the left-Alt symbol set, etc. This approach yields what one might call a "sticky" shift. Other possibilities involve the NumLock and ScrollLock keys in combinations with the left/right Ctrl/Alt keys. Pretty soon one realizes that this approach rapidly approaches literally billions of combinations, i.e., several orders of magnitude more than Unicode can handle!

**Macros.** The autocorrect and keyboard macro features of some word processing systems provide other ways of entering mathematical characters for people familiar with TeX. For example, typing \alpha inserts α if the appropriate autocorrect entry is present. This approach is noticeably faster than using menus.

**Hexadecimal input.** A handy hex-to-Unicode entry method works with recent Microsoft text software (similar approaches are available on other systems) to insert Unicode characters in general and math characters in particular. Basically one types a character's hexadecimal code (in ASCII), making corrections as need be, and then types Alt+x. The hexadecimal code is replaced by the corresponding Unicode character. The Alt+x can be a toggle, that is, type it once to convert a hex code to a character and type it again to convert the character back to a hex code. If the hex code is preceded by one or more hexadecimal digits, one needs to "select" the code so that the preceding hexadecimal characters aren't included in the code. The code can range up to the value 0x10FFFF, which is the highest character in the 17 planes of Unicode.

**Pull-down menus**. Pull-down menus are a popular method for handling large character sets, but they are slow. A better approach is the *symbol box*, which is an array of symbols either chosen by the user or displaying the characters in a font. Symbols in symbol boxes can be dragged and dropped onto key combinations on the on-screen keyboard(s), or directly into applications. On-screen keyboards and symbol boxes are valuable for entry of mathematical expressions and of Unicode text in general.

**Unicode plain-text mathematics.** One use of the plain-text format is as a math input method, both for search text and for general editing.

## *4.3 Use of Math Characters in Computer Programs*

It can be very useful to have typical mathematical symbols available in computer programs (see Sec. 5.3 for a more detailed discussion). A key point is that the compiler should display the desired characters in both edit and debug windows. A preprocessor can translate MathML, for example, into C++, but it will not be able to make the debug windows use the math-oriented characters unless it can handle the underlying Unicode characters. Java has made an important step in this direction by allowing Unicode variable names. The mathematical alphanumeric symbols allow this approach to go further with relatively little effort for compilers.

The advantages of using the Unicode plain text in computer programs are at least threefold: 1) many formulas in document files can be programmed simply by copying them into a program file and inserting appropriate multiplication dots. This dramatically reduces coding time and errors. 2) The use of the same notation in programs and the associated journal articles and books leads to an unprecedented level of self-documentation. 3)

In addition to providing useful tools for the present, these proposed initial steps should help one figure out how to accomplish the ultimate goal of teaching computers to understand and use arbitrary mathematical expressions.

# 5 Unicode Plain Text Encoding of Mathematics

Getting computers to understand human languages is important in increasing the utility of computers. Natural-language translation, speech recognition and generation, and programming are typical ways in which such machine comprehension plays a role. The better this comprehension, the more useful the computer, and hence there has been considerable current effort devoted to these areas since the early 1960s.

Ironically one truly international human language that tends to be neglected in this connection is mathematics itself.

With a few conventions, Unicode can encode most mathematical expressions in readable (near-)plain text. The format is linear, but can be displayed in built-up form. The approach uses heuristics based on the Unicode math properties to recognize mathematical expressions without the aid of explicit math-on/off commands. This is facilitated by Unicode's strong support for mathematical symbols. This plain-text representation is substantially more compact and easy to read compared to the LaTeX dialect of TeX, "Unicode TeX", or MathML. Most mathematical expressions up through calculus can be represented unambiguously in Unicode plain text. The plain-text encoding can be easily exported to (La)TeX, MathML, C++, and symbolic manipulation programs.

Note that this discussion is intended to illustrate how mathematical plain text might be useful, for example, in math input and computer programs. This is intended to be a preliminary draft of a specification of how to encode mathematical expressions in plain text. Readers are encouraged to comment on the proposed scheme.

Given the power of Unicode relative to ASCII, how much better can a plain-text encoding of mathematical expressions look using Unicode? The most well-known plain-text ASCII encoding of such expressions is that of TeX, so one uses it for comparison. MathML is considerably more verbose than TeX, so some of the comparisons apply to it as well. Notwithstanding TeX's phenomenal success in the science and engineering communities, a casual glance at its representation of mathematical expressions reveals that they do not look very much like the expressions they represent. It is certainly not easy to make algebraic calculations directly using TeX's notation. With Unicode, one can represent mathematical expressions more readably, and the resulting plain text can be used directly for such calculations.

For example, one way to specify a TeX fraction numerator consists of the expression \frac{numerator}{denominator}. In both the fraction and subscript/superscript cases, the { } are not printed. These simple rules immediately give a "plain text" that is unambiguous, but looks quite different from the corresponding mathematical notation, thereby making it hard to read.

Instead, suppose one defines a simple operand to consist of all consecutive letters and digits, i.e., a span of characters belong to the L*x* and Nd General Categories. As such, a simple numerator or denominator is terminated by any operator, including, for example, arithmetic operators, the blank, all Unicode characters with codes U+22xx, and a special argument "break" operator consisting of a small raised dot. The fraction operator is given

by the Unicode fraction slash operator U+2044, which is depicted here with the glyph ⁄. So the simple built-up fraction

$$\frac{abc}{d} \ .$$

appears in plain text as *abc⁄d*.
For more complicated operands (such as those that include operators), parentheses ( ), brackets [ ], or braces { } can be used to enclose the desired character combinations. If parentheses are used and the outermost parenthesis set is preceded and followed by operators, that set is not displayed in built-up form, since usually one does not want to see such parentheses. So the plain text (*a* + *b*)⁄c displays as

$$\frac{a + c}{d} \ .$$

In practice, this approach leads to plain text that is significantly easier to read than TeX's, e.g., \frac{a + c}{d} , since in many cases, outermost parentheses are not needed, while TeX requires { }'s. To force the display of an outermost parenthesis set, one encloses the set, in turn, within parentheses, which then become the outermost set. For example, ((*a* + *b*))/*c* displays as

$$\frac{(a + c)}{d} \ .$$

A really neat feature of this notation is that the plain text is, in fact, a legitimate mathematical notation in its own right, so it is relatively easy to read. In MathML, this fraction reads as

```
<mfrac>
    <mrow>
        <mi>a</mi>
        <mo>+</mo>
        <mi>c</mi>
    </mrow>
    <mrow>
          <mi>d</mi>
    </mrow>
</mfrac>
```

Subscripts and superscripts are a bit trickier, but they are still quite readable. Specifically, one can introduce a subscript by a subscript operator, which is displayed as the subscripted down arrow ↓. A simple subscript operand consists of the string of one or more characters with the General Categories L*x* (alphabetic) and Nd (decimal digits), as well as the invisible comma. For example, a pair of subscripts, such as $\delta_{\mu\nu}$ is written as δ↓μν. Similarly, superscripts are introduced by a superscript operator, which is displayed here as the superscripted up arrow ↑. So $a^{\uparrow}b$ means $a^b$.
Compound subscripts and superscripts include expressions within parentheses, square brackets, and curly braces. So $\delta_{\mu+\nu}$ is written as δ↓(μ+ν). In addition it is worthwhile to treat two more operators, the comma and the period, in special ways. Specifically, if a subscript operand is followed directly by a comma or a period that is, in turn, followed by whitespace, then the comma or period appears on line, i.e., is treated as the operator that

terminates the subscript. However a comma or period followed by a non-operator is treated as part of the subscript. This refinement obviates the need for many overriding parentheses, thereby yielding a more readable plain text.
Another kind of compound subscript is a subscripted subscript, which works using right-to-left associativity, e.g., $a\downarrow b\downarrow c$ means $a_{b_c}$. Similarly $a^{\uparrow}b^{\uparrow}c$ means $a^{b^c}$.

Parentheses are needed for constructs such as a subscripted superscript like $a^{b_c}$, which is given by $a^{\uparrow}(b\downarrow c)$.

As an example of a slightly more complicated example, consider the expression $W^{3\beta}_{\delta_1\rho_1\sigma_2}$ has the plain-text format $W^{\uparrow}3\beta\downarrow\delta_1\rho_1\sigma_2$. In contrast, for TeX, one types

$W^{3\beta}_{\delta_1\rho_1\sigma_2}$ ,

which is hard to read. The TeX version looks distinctly better using Unicode for the symbols, namely $W^{3β}_{δ_1ρ_1σ_2}$ or $W^{3β}_{δ₁ρ₁σ₂}$, since Unicode has a full set of decimal subscripts and superscripts. However the need to use the {}, not to mention the $'s, makes even the last of these harder to read than the plain-text version $W^{\uparrow}3\beta\downarrow\delta_1\rho_1\sigma_2$.
For the ratio

,

the Unicode plain text reads $\alpha_2^3/(\beta_2^3 + \gamma_2^3)$, while the standard TeX version reads as

${\alpha^3_2 \over \beta^3_2 + \gamma^3_2}$.

The Unicode plain text is a legitimate mathematical expression, while the TeX version bears no resemblance to a mathematical expression.

TeX becomes very cumbersome for longer equations such as

$$W^{3\beta}_{\delta_1\rho_1\sigma_2} = U^{3\beta}_{\delta_1\rho_1} + \frac{1}{8\pi^2}\int_{\alpha_1}^{\alpha_2} d\alpha_2' \left[\frac{U^{2\beta}_{\delta_1\rho_1} - \alpha_2' U^{1\beta}_{\rho_1\sigma_2}}{U^{0\beta}_{\rho_1\sigma_2}}\right].$$

The Unicode plain-text version of this reads as

$W^{\uparrow}\beta\downarrow\delta_1\rho_1\sigma_2 = U^{\uparrow}3\beta\downarrow\delta_1\rho_1 + 1/8\pi^2 \int\downarrow\alpha_1{}^{\uparrow}\alpha_2\ d\alpha_2' \ [(U^{\uparrow}2\beta\downarrow\delta_1\rho_1 - \alpha_2' U^{\uparrow}1\beta\downarrow\rho_1\sigma_2)/U^{\uparrow}0\beta\downarrow\rho_1\sigma_2]$

while the standard TeX version reads as

```
${W^{3\beta}_{\delta_1\rho_1\sigma_2}
        = U^{3\beta}_{\delta_1\rho_1} + {1 \over 8\pi^2}
        \int_{\alpha_1}^{\alpha_2} d\alpha_2' \left[
```

```
{U^{2\beta}_{\delta_1\rho_1} - \alpha_2'
U^{1\beta}_{\rho_1\sigma_2} \over
U^{0\beta}_{\rho_1\sigma_2}} \right] }$ .
```

In a “Unicoded” TeX, it could read as

```
${W^{3β}_{δ₁ρ₁σ₂} = U^{3β}_{δ₁ρ₁} + {1 / 8π²}
        ∫_{α₁}^{α₂} dα₂′ \left[{U^{2β}_{δ₁ρ₁} - α₂′U^{1β}_{ρ₁σ₂}
        / U^{0β}_{ρ₁σ₂}} \right] }$ ,
```

which is significantly easier to read than the ASCII TeX version, although still much harder to read than the Unicode plain-text version.

Brackets [ ], braces { }, and parentheses ( ) represent themselves in the Unicode plain text, and a word processing system capable of displaying built-up formulas should expand them to fit around what’s inside them. Here U+2032 is used for \prime and U+2044 for \over.

Certain operators like brackets, braces, parentheses, superscript, subscript, integral, etc. have special meaning in this notation. To treat them as normal characters, one precedes them by the “literal operator”, for which the *percent sign* % is handy. So %[ is displayed as an ordinary left square bracket, with no attempt by the software to match a corresponding right square bracket. Table 5.1 lists operators that have special meanings.

## *5.1 Recognizing Mathematical Expressions*

Unicode plain-text encoded mathematical expressions can be used “as is” for simple documentation purposes. Use in more elegant documentation and in programming languages requires knowledge of the underlying mathematical structure. This section describes some of the heuristics that can distill the structure out of the plain text.

Many mathematical expressions identify themselves as mathematical, obviating the need to declare them explicitly as such. One well-known TeX problem is TeX’s inability to detect expressions that are clearly mathematical, but that are not enclosed within $’s. If one leaves out a $ by mistake, one gets a slew of error messages because TeX interprets subsequent text in the wrong mode.

An advantage of recognizing mathematical expressions without math-on/math-off syntax is that it is much more tolerant to user errors of this sort. Resyncing is automatic, while in TeX one basically has to start up again from the omission in question. Furthermore, this approach should be useful in an important related endeavor, namely in recognizing and converting the mathematical literature that is not yet available in an object-oriented machine-readable form, into that form.

It is possible to use a number of heuristics for identifying mathematical expressions and treating them accordingly. These heuristics are not foolproof, but they lead to the most popular choices. Special commands discussed at the end of this section can be used to overrule these choices. Ultimately the approach could be used as an autoformat style wizard that tags expressions with a rich-text math style. The user could then override cases that were tagged incorrectly. A math style would connect in a straightforward way to appropriate MathML tags.

The basic idea is that math characters identify themselves as such *and* potentially identify their surrounding characters as math characters as well. For example, the fraction (U+2044) and ASCII slashes, symbols in the range U+2200 through U+22FF, the symbol combining marks (U+20D0 - U+20FF), and in general, Unicode characters with the mathematics property, identify the characters immediately surrounding them as parts of math expressions.

If English letter mathematical variables are already given in one of the math alphabets, they are considered parts of math expressions. If they are not, one can still have some recognition heuristics as well as the opportunity to italicize appropriate variables. Specifically ASCII letter pairs surrounded by whitespace are often mathematical expressions, and as such should be italicized in print. If a letter pair fails to appear in a list of common English and European two-letter words, it is treated as a mathematical expression and italicized. Many Unicode characters are not mathematical in nature and suggest that their neighbors are not parts of mathematical expressions.

Strings of characters containing no whitespace but containing one or more unambiguous mathematical characters are generally treated as mathematical expressions. Certain two-, three-, and four-letter words inside such expressions should *not* be italicized. These include trigonometric function names like sin and cos, as well as ln, cosh, etc. Words or abbreviations, often used as subscripts (see the program in Sec. 5.3), also should not be italicized, even when they clearly appear inside mathematical expressions.

Special cases will always be needed, such as in documenting the syntax itself. The literal operator introduced earlier (%) causes the character that follows it to be treated as an ordinary character. This allows the printing of characters without modification that by default are considered to be mathematical and thereby subject to a changed display. Similarly, mathematical expressions that the algorithms treat as ordinary text can be sandwiched between math-on and math-off symbols. For this purpose, it is tempting to use Asian brackets such as the white lenticular brackets (U+3016 and U+3017), a matched pair that is not common in mathematical expressions. Such “overhead” symbols clutter up the text and hopefully will be rarely needed in Unicode plain text. This does complicate the preparation of technical documents and although one can get very good at it, it is not the most user-friendly way of doing things. On the other hand, identifying the beginning and end of math expressions using $’s or use of extensive markup tags is not user friendly either.

## 5.2 Minimal Operator Summary

Operands in subscripts, superscripts, fractions, roots, boxes, etc. are defined in part in terms of operators and operator precedence. While such notions are very familiar to mathematically oriented people, some of the symbols that are defined here as operators might surprise one at first. Most notably, the space (ASCII 32) is an important operator in the plain-text encoding of mathematics. A small but common list of operators

**Table 5.1 A list of common operators ordered by precedence**

| |
|---|
| FF CR \ |
| ( [ { |
| ) ] } \| |

| Space " . , = - + LF Tab |
|---|
| / * × · • • \| |
| ■ √ |
| ∫ Σ Π |
| ↓ ↑ |

where Tab = U+0009, LF = U+000A, FF = U+000C, and CR = U+000D.
As in arithmetic, operators have precedence, which streamlines the interpretation of operands. The operators are grouped above in order of increasing precedence, with equal precedence values on the same line. For example, in arithmetic, 3+1/2 = 3.5, not 2. Similarly the plain-text expression α + β/γ means

$$\alpha + \frac{\beta}{\gamma} \quad \text{not} \quad \frac{\alpha + \beta}{\gamma}.$$

As in arithmetic, precedence can be overruled, so (α + β)/γ gives the latter.
The following gives a list of the syntax for a variety of mathematical constructs.

| | |
|---|---|
| $exp_1/exp_2$ | Create a built-up fraction with numerator $exp_1$ and denominator $exp_2$. Numerator and denominator expressions are terminated by operators such as /*])↑↓· and blank (can be overruled by enclosing in parentheses). The "/" is given by U+2044. |
| $^{\uparrow}exp_1$ | Superscript expression $exp_1$. The superscripts $^{0\text{-}9+\text{-}()}$ exist as Unicode symbols. Sub/superscripts expressions are terminated by /*])↑↓· and blank. Sub/superscript operators associate right to left. |
| $\downarrow exp_1$ | Subscript expression $exp_1$. The subscripts $_{0\text{-}9+\text{-}()}$ exist as Unicode symbols. |
| $[exp_1]$ | Surround $exp_1$ with built-up brackets. Similarly for { } and ( ). |
| $[exp_1]^{\uparrow}exp_2$ | Surround $exp_1$ with built-up brackets followed by superscripted $exp_2$ (moved up high enough). Similarly for { } and ( ). |
| $\sqrt{}exp_1$ | Square root of $exp_1$. |
| · | Small raised dot that is not intended to print. It is used to terminate an operand, such as in a subscript, superscript, numerator, or denominator, when other operators cannot be used for this purpose. Similar raised dots like • and • also terminate operands, but they are intended to print. |
| $\Sigma\downarrow exp_1{}^{\uparrow}exp_2$ | Summation from $exp_1$ to $exp_2$. $\downarrow exp_1$ and $^{\uparrow}exp_2$ are optional. |
| $\Pi\downarrow exp_1{}^{\uparrow}exp_2$ | Product from $exp_1$ to $exp_2$. |

$\int_{\downarrow exp_1}^{\uparrow exp_2}$ Integral from $exp_1$ to $exp_2$.

$exp_1 \mid exp_2$ Align $exp_1$ over $exp_2$ (like fraction without bar). Useful for building up matrices as a set of columns.

Diacritics are handled using Unicode combining marks (U+0300 - U+036F, U+20D0 - U+20FF). Note that many more operators can be added to fill out the capabilities of the approach in representing mathematical expressions in Unicode plain (or almost plain) text.

## 5.3 Using Plain-Text Mathematics in Programming Languages

In the middle 1950's, the authors of FORTRAN named their computer language after FORmula TRANslation, but they only went part way. Arithmetic expressions in Fortran and other current high-level languages still do not look like mathematical formulas and considerable human coding effort is needed to translate formulas into their machine comprehensible counterparts. For example, Fortran's superscript a**k isn't as readable as $a^k$ and Fortran's subscript a(k) isn't as readable as $a_k$. Bertrand Russell once said that notation is a great teacher and that a perfect notation would be a substitute for thought. From this point of view, modern computer languages are badly lacking. Specialized mathematical applications such Mathematica are substantially better in this regard.

Using real mathematical expressions in computer programs would be far superior in terms of readability, reduced coding times, program maintenance, and streamlined documentation. In studying computers we have been taught that this ideal is unattainable, and that one must be content with the arithmetic expression as it is or some other non-mathematical notation such as TeX's. It is time to reexamine this premise. Whereas true mathematical notation clearly used to be beyond the capabilities of machine recognition, we feel it no longer is.

In general, mathematics has a very wide variety of notations, none of which look like the arithmetic expressions of programming languages. Although ultimately it would be desirable to be able to teach computers how to understand all mathematical expressions, it is useful to start with the Unicode plain-text format presented here.

In raw form, these expressions look very like traditional mathematical expressions. With use of the heuristics described above, they can be printed or displayed in traditional built-up form. On disk, they can be stored in pure-ASCII program files accepted by standard compilers and symbolic manipulation programs like Derive, Mathematica, and Macsyma. The translation between Unicode symbols and the ASCII names needed by ASCII-based compilers and symbolic manipulation programs is carried out via table-lookup (on writing to disk) and hashing (on reading from disk) techniques.

Hence formulas can be at once printable in manuscripts *and* computable, either numerically or analytically. Note that this is a goal of MathML as well, but attained in a relatively complex way using specialized tools. The idea here is that regular programming languages can have expressions containing standard arithmetic operations and special characters, such as Greek, italics, script, and various mathematical symbols like the square root. Two levels of implementation are envisaged: scalar and vector. Scalar opera-

tions can be performed on traditional compilers such as those for C and Fortran. The scalar multiply operator is represented by a raised dot, a legitimate mathematical symbol, instead of the asterisk. To keep auxiliary code to a minimum, the vector implementation requires an object-oriented language such as C++.

The advantages of using the Unicode plain text are at least threefold:

1) many formulas in document files can be programmed simply by copying them into a program file and inserting appropriate multiplication dots. This dramatically reduces coding time and errors.
2) The use of the same notation in programs and the associated journal articles and books leads to an unprecedented level of self documentation. In fact, since many programmers document their programs poorly or not at all, this enlightened choice of notation can immediately change nearly useless or nonexistent documentation into excellent documentation.
3) In addition to providing useful tools for the present, these proposed initial steps should help one figure out how to accomplish the ultimate goal of teaching computers to understand and use arbitrary mathematical expressions. Such machine comprehension would greatly facilitate future computations as well as the conversion of the existing paper literature and Pen-Windows input into machine usable form.

The concept is portable to any environment that supports a large character set, preferably Unicode, and it takes advantage of the fact that high-level languages like C and Fortran accept an "escape" character ("_" and "$", respectively) that can be used to access extended symbol sets in a fashion similar to TeX. In addition, the built-in C preprocessor allows niceties such as aliasing the asterisk with a raised dot, which is a legitimate mathematical symbol for multiplication. The Java and C# languages allow direct use of Unicode variable names, which is a major step in the right direction. Compatibility with unenlightened ASCII-only compilers can be done via an ASCII representation of Unicode characters.

## *5.4 Comparison of Programming Notations*

To get an idea as to the differences between the standard way of programming mathematical formulas and the proposed way, compare the following versions of a C++ routine entitled IHBMWM (inhomogeneously broadened multiwave mixing)

```
void IHBMWM(void)
{
        gammap = gamma*sqrt(1 + I2);
        upsilon = cmplx(gamma+gamma1, Delta);
        alphainc = alpha0*(1-(gamma*gamma*I2/gammap)/(gammap + upsilon));

        if (!gamma1 && fabs(Delta*T1) < 0.01)
                alphacoh = -half*alpha0*I2*pow(gamma/gammap, 3);
        else
        {
                Gamma = 1/T1 + gamma1;
                I2sF = (I2/T1)/cmplx(Gamma, Delta);
```

```
                betap2 = upsilon*(upsilon + gamma*I2sF);
                beta = sqrt(betap2);
                alphacoh = 0.5*gamma*alpha0*(I2sF*(gamma + upsilon)
                                /(gammap*gammap - betap2))
                                *((1+gamma/beta)*(beta - upsilon)/(beta + upsilon)
                                - (1+gamma/gammap)*(gammap - upsilon)/
                                (gammap + upsilon));
        }
        alpha1 = alphainc + alphacoh;
}

void IHBMWM(void)
{
        γ′ = γ•√(1 + I2);
        υ = γ + γ1 + i•Δ;
        αinc = α0•(1 - (γ•γ•I2/γ′)/(γ′ + υ));
        if (!γ1 || fabs(Δ•T1) < 0.01)
        αcoh = -.5•α0•I2•pow(γ/γ′, 3);
        else
        {
                Γ = 1/T1 + γ1;
                I2ℱ = (I2/T1)/(Γ + i•Δ);
                β2 = υ•(υ + γ•I2ℱ);
                β = √β2 ;
                αcoh = .5•γ•α0•(I2ℱ(γ + υ)/(γ′•γ′ - β2))
                        ×((1+γ/β)•(β - υ)/(β + υ) - (1+γ/γ′)•(γ′ - υ)/(γ′ + υ));
        }
        α1 = αinc + αcoh ;
}
```

[TBD: *Discuss the interaction of C operators like || with math operators*]

The above function runs fine with current C++ compilers, but C++ does impose some serious restrictions based on its limited operator table. For example, vectors can be multiplied together using dot, cross, and outer products, but there's only one asterisk to overload in C++. In built-up form, the function looks even more like mathematics, namely

```
void IHBMWM(void)
{
        γ′ = γ•√(1 + I2) ;
        υ = γ + γ1 + i•Δ;
        αinc = α0•(1 - (γ•γ•I2/γ′)) / (γ′ + υ) ;
        if (!γ1 || fabs(Δ•T1) < 0.01)
        αcoh = -.5•α0•I2•pow(γ/γ′, 3);
        else
```

{

$\Gamma = 1/T_1 + \gamma_1;$

$I_2\mathcal{F} = \dfrac{I_2/T_1}{\Gamma + i\bullet\Delta}\ ;$

$\beta^2 = \upsilon\bullet(\upsilon + \gamma\bullet I_2\mathcal{F});$

$\beta = \sqrt{\beta^2}\ ;$

$\alpha_{coh} = .5\bullet\gamma\bullet\alpha_0\bullet\dfrac{I_2\mathcal{F}(\gamma + \upsilon)}{\gamma'\bullet\gamma' - \beta^2} \times\left(\left(1+\dfrac{\gamma}{\beta}\right)\bullet\dfrac{\beta - \upsilon}{\beta + \upsilon} - \left(1+\dfrac{\gamma}{\gamma'}\right)\bullet\dfrac{\gamma' - \upsilon}{\gamma' + \upsilon}\right);$

}

$\alpha_1 = \alpha_{inc} + \alpha_{coh}\ ;$

}

The ability to use the second and third versions of the function was built into the PS Technical Word Processor circa 1988. With it we already came much closer to true formula translation on input, and the output is displayed in standard mathematical notation. Lines of code can be previewed in built-up format, complete with fraction bars, square roots, and large parentheses. To code a formula, one copies (cut and paste) it from a technical document into a program file, insert appropriate raised dots for multiplication and compile. No change of variable names is needed. Call that 70% of true formula translation! In this way, the C++ function on the preceding page compiles without modification. The code appears nearly the same as the formulas in print [see Chaps. 5 and 8 of Meystre and Sargent].

**Open issues and future.** Questions remain, such as to whether subscript expressions in the Unicode plain text should be treated as part of program-variable names, or whether they should be translated to subscript expressions in the target programming language. Similarly, it would be straightforward to automatically insert an asterisk (indicating multiplication) between adjacent symbols, rather than have the user do it. However here there is a major difference between mathematics and computation: symbolically, multiplication is infinitely precise and infinitely fast, while numerically, it takes time and is restricted to a binary subset of the rationals with very limited (although often adequate) precision. Consequently for the moment, at least, it seems wiser to consider adjacent symbols as part of a single variable name, just as adjacent ASCII letters are part of a variable name in current programming languages. Perhaps intelligent algorithms will be developed that decide when multiplication should be performed and insert the asterisks optimally. Export to TeX is similar to that to programming languages, but has a modified set of requirements. With current programs, comments are distilled out with distinct syntax. This same syntax can be used in the Unicode plain-text encoding, although it is interesting to think about submitting a mathematical document to a preprocessor that can recognize and separate out programs for a compiler. In this connection, compiler comment syntax is not particularly pretty; ruled boxes around comments and vertical dividing lines between code and comments are noticeably more readable. So some refinement of the ways that comments are handled would be very desirable. For example, it would be nice to have a vertical window-pane facility with synchronous window-pane scrolling and the ability to display C code in the left pane and the corresponding // comments in the right pane. Then if one wants to see the comments, one widens the right pane accordingly. On the other hand, to view lines with many characters of code, the // comments need not get in the

way. Such a dual-pane facility would also be great for working with assembly-language programs.

**Export to TeX.** With TeX, the text surrounding the mathematics is part and parcel of the technical document, and TeX needs its $'s to distinguish the two. These can be included in the plain text, but we have repeatedly pointed out how ugly this solution is. The heuristics described above go a long way in determining what is mathematics and what is natural language. Accordingly, the export method consists of identifying the mathematical expressions and enclosing them in $'s. The special symbols are translated to and from the standard TeX ASCII macros via table lookup and hashing, as for the program translations. Better yet, TeX should be recompiled to use Unicode.

### *5.5 Conclusions*

This section has shown how, with a few additions to Unicode, mathematical expressions can usually be represented with a readable Unicode plain-text format. The text consists of combinations of operators and operands. A simple operand consists of a span of non-operators, a definition that dramatically reduces the number of parenthesis-override pairs and thereby increases the readability of the plain text. The only disadvantage to this approach versus TeX's ubiquitous { } pairs is that the user needs to know what characters are operators. To reveal the operators, operator-aware editors could be instructed to display operators with a different color or some other attribute. To simplify the notation, operators have precedence values that control the association of operands with operators unless overruled by parentheses. Heuristics can be applied to the Unicode plain text to recognize what parts of a document are mathematical expressions. This allows the Unicode plain text to be used in a variety of ways, including in technical document preparation, symbolic manipulation, and numerical computation.

The heuristics given for recognizing mathematical expressions work well, but they are not infallible. An effective use of the heuristics would be as an autoformatting wizard that delimits what it thinks is mathematics with mathematics on/off codes. The user could then overrule incorrect choices. Once marked unequivocally as mathematics (an alternative to TeX's $'s), export to MathML, compilers, and other consumers of mathematical expressions is straightforward. We have a workable plain-text encoding of mathematics that looks very much like mathematics even with the most limited display capabilities. Appropriate display software can make it look like the real thing.

## 6 References

[MathML] http://www.w3.org/mathml

[Meystre and Sargent] P. Meystre and M. Sargent III (1991), *Elements of Quantum Optics*, Springer-Verlag

[TeX] http://www.ams.org/tex/publications.html

[LaTeX]

[STIX] http://www.ams.org/STIX.

## 7 Modifications

This is the first draft version for public review.