L2/00-409

Unicode Support for Mathematics

12-Nov-2000

Barbara Beeton (bnb@ams.org)
Asmus Freytag (asmusf@ix.netcom.com)
Murray Sargent III (murrays@microsoft.com)

[*This is a preliminary draft of a forthcoming Unicode TR on mathematics. Updates are being made continually, but people might find this draft of interest. At this stage, only a Word document is available, but a PDF version will be made available when the document becomes more stable.*]

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 math in systems like Mathematica and MathCAD, computer programs, and plain text. In this paper, we describe the Unicode mathematics character groups and give some of their default math properties. 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.

The paper 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 isn't intended to be a complete specification or to be used for general information interchange at this stage in its development.

1 Mathematical Character Repertoire

Unicode 3.2 has a quite complete set of standard math characters to support math publication on and off the web. Specifically there are 591 new symbols (since Unicode 3.0) and 996 new alphanumeric symbols in addition to 340 symbols already encoded in Unicode 3.0 for a total of 1927 math symbols. This repertoire is the result of input from many sources, notably from the STIX project, and enables one to display virtually all standard mathematical symbols. MathML is a major beneficiary of this support and lobbied in favor of the inclusion of the new characters. In addition, this math support lends itself to a useful plain-text encoding (see Sec. 4) that's much more compact than MathML or TeX.

1.1 Mathematical Alphanumeric Characters

Mathematics uses the basic Latin letters and digits (a – z, A – Z, and 0 – 9), as well as the uppercase Greek letters A-Ω (U+0391 - U+03A9), plus the nabla ∇ (U+2207) and

Θ variant (U+03F4), and the lowercase Greek 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. For each of these six characters, both glyph variants can appear in the same document with different meaning.

Therefore the basic set of mathematical alphanumerical characters consists of 52 Latin letters, ten digits, plus 24+2 uppercase Greek characters and 25+7 lowercase Greek characters. In addition to this basic set, mathematics uses the four Hebrew-derived characters (U+2135 – U+2138). Occasional use of Cyrillic uppercase Shah (U+0428) is also known as well as the Hiragana no (U+306E).

1.2 Mathematical Alphabets

Mathematics has need for a number of Latin and Greek alphabets that on first thought appear to be font variations of one another, e.g., normal, bold, italic and script H. However in any given document, these characters have distinct mathematical semantics. For example, a normal H represents a different variable from a bold H, etc. If one drops these distinctions in plain text, one gets gibberish. Instead of the well-known Hamiltonian formula

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

you'd get the integral equation

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

although ordinarily the H and E would be italicized.

Accordingly, Unicode includes the basic Latin and Greek alphabets in a number of specifically mathematical styles, such as normal, bold, italic, script, etc., as found in a comprehensive survey of the mathematical literature undertaken by the STIX project. Note that these alphabets encode only the semantic distinction, but not which normal, script, fraktur, double-struck, sans-serif, or monospace fonts are used, because this is beyond the scope of plain-text. The normal (that is upright serifed) letters have been unified with the existing characters in the Basic Latin and Greek blocks. There are 25 double-struck, 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 for the convenience of implementations. Only unaccented forms of the Latin letters are included, since general accents, e.g. the acute accent, would interfere with common mathematical diacritics, such as single or double dot above used commonly in physics to denote a derivative with respect to the time variable. Such accented mathematical symbols are always represented by combining character sequences.

When used with markup languages, for example with MathML <citation> the characters are expected to be used directly, instead of indirectly via entity references or by composing them from base letters and style markup. For consistency, all mathematical alphanumerics have compatibility decompositions to the base Latin and Greek letters - folding away such distinctions, however, is usually not desirable.

The alphabetic symbols encountered in mathematics are given in the following table:

Math style	Characters from basic set	Plane
normal (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.

To know if a character is a math alphanumeric character you can check for inclusion in the two ranges U+2100 – U+214F and U+1D400 – U+1D7FF. In the programming language C, you can see if the character ch is in these ranges using the if() statement

```
if(IN_RANGE(0x2100, ch, 0x214F) || IN_RANGE(0x1D400, ch, 0x1D7FF)) {}
```

where the macro IN_RANGE(n1, ch, n2) is defined by

```
#define IN_RANGE(n1, b, n2)      ((unsigned)((b) - (n1)) <= unsigned((n2) - (n1)))
```

This macro effectively has only one goto and is almost as fast as a single compare. With a single goto, the version IN_RANGE(0x2A00, ch | 0x200, 0x2AFF) matches all symbols in the Mathematical Operators and Supplemental Mathematical Operators blocks.

Some duplicated Greek letters are U+00B5 µ MICRO SIGN, U+2126 Ω OHM SIGN, and several characters among the APL functional symbols in the Miscellaneous Technical block.

1.3 Combining Marks

Mathematical characters are often enhanced via use of combining marks in the ranges U+0300 – U+036F and the mathematical combining marks in the range U+20D0 – U+20FF. These characters follow the base characters as in nonmathematical Unicode text. If a span of characters are 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 includes some combining marks that are designed to be used for pairs of characters, e.g., U+0360 through U+0362.

1.4 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, since they may well convey different meaning in a given context. Typically the choice of a vertical or forward-slanting stroke seems to be an aesthetic one, but both slants might appear in a given context and a back-slanted stroke always means something else. Accordingly Version 3.2 of The Unicode Standard does not unify many characters that might appear to be only aesthetic variants of one another.

On the other hand, 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.

Several mathematical operators derived from Greek characters have been given separate encodings to match usage in existing standards. 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.

Some typographical aspects of operators are discussed in Sec. 2.2. 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 preferrably 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.

1.5 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. These characters are not used in MathML and TeX. If they are used, it's 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.

1.6 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.

1.7 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). In particular, the Miscellaneous Technical block contains a set of brace pieces for building up large versions of (,), [,], {, }, ∑, and ∫ in a way that the displayed stem weights are compatible with the accompanying smaller characters. These brace pieces are not used in stored mathematical text, but are often used in creating technical display and print drivers.
[Add info from TUS]

1.8 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.

1.9 Variant Selector

Another way to represent math alphanumerics in plain text was considered (but abandoned) that uses math variant tags that follow the appropriate base characters. This approach is more general than outright encoding since the variant tags could follow any characters in the BMP. However only certain characters should be eligible for these math styles, so one would have to have tables defining which combinations are legal and which should be discarded or ignored. The approach was dropped because it was felt that it could be abused too easily for nonmath, rich-text purposes that would be better handled using markup.

Nevertheless, the variant selector VS1 was introduced to get well-defined variants of particular math symbols. The differences 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. The characters defined for use with the variant selector are given in the following table

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

1.10 Multiple Representations of the Same Character

There have been many discussions as to various normalization forms for Unicode characters; Unicode Technical Report 15 discusses the subject in detail. Math characters are no exception: there are multiple ways of expressing various math characters. 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. Accordingly it's worthwhile to give some guidelines.

The first idea is to use 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.

On the other hand, for alphabetic characters, use the fully decomposed sequence, e.g., use U+0061, U+0308 for ä, not U+00E4. Mathematics uses a multitude of combining marks that greatly exceeds the predefined composed characters in Unicode. It's 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 don't. The combining character sequences also typically have semantics as a group, so it's handy to be able to manipulate and search for them individually without having to have special tables to decompose characters for this purpose.

MathML uses markup in some situations to allow multicharacter combining marks, such as a tilde over two or more characters, or an extended radical bar covering a mul-

ticharacter expression. Accordingly MathML does not use bicharacter combining marks like U+0360 through U+0362.

1.11 Nonstandard Symbols

Mathematicians are by their natures inventive people and will continue to invent new symbols to express their theories. Until these symbols are used by a number of people, they shouldn't be standardized. Nevertheless, one needs a way to handle these symbols in their initial nonstandard usage.

The private use area (0xE000 – 0xF8FF) can be used for such nonstandard symbols. It's a tricky business, since the PUA is used for many purposes. For example, it's used on Microsoft operating systems to round-trip codes that aren't currently in Unicode, most notably many Chinese characters. The precise usage may well change since many such symbols may be assigned to plane 2 (Extension B) and hence are standardized.

When using the PUA, it's 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's maintained in a rich-text layer parallel to the plain text. Such layers are used by rich-text programs such as Microsoft Word and Internet Explorer.

2 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 overriden, 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, we describe these properties in greater detail.

In particular, many Unicode characters nearly always appear in mathematical expressions and are given the generic mathematics property. 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). The math property is useful in heuristics that seek to identify mathematical expressions in plain text. [TODO: mention the new Unicode 3.2 symbol groups]

2.1 Classification by Usage Frequency

2.1.1 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?}

2.1.2 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.

2.1.3 Other

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

2.2 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.

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

2.2.2 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

2.2.3 Large Operator

These include n-ary operators like summation and integration. These may expand in size to fit their associated expressions.

2.2.4 Digits

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

2.2.5 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 don't appear to be "logical", e.g., $]x,y[$.

2.2.6 Fences

Fences are similar to opening and closing delimiters, but aren't paired. In addition, they include "mid" delimiters, which aren't opening or closing ijn character.

2.2.7 Combining Marks

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 is a typesetting problem. When using combining marks, the composite characters have the same typesetting class as the base character.

2.3 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 precedences, 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.

3 Implementation Guidelines

3.1 Input of Mathematical and Other Unicode Characters

This leads to the important problem of input ease. The ASCII math symbols are easy to find, e.g., + - / * [ ] ( ) { }, but often need to be used as themselves. 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's easier to type ASCII letters than italic letters, but when used as mathematical variables, such

letters are traditionally italicized in print. 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.

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 example, 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 key pad 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 you realize that this approach rapidly approaches literally billions of combinations, i.e., several orders of magnitude more than Unicode can handle!

The autocorrect feature of Microsoft Word 97 (and later) offers another way of entering mathematical characters for people familiar with TeX. For example, you type \alpha and shazaam! It changes to α. This approach is noticably faster than using 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.

3.2 Use of Math Characters in Computer Programs

It can be very useful to have typical mathematical symbols available in computer programs. Java has made an important step in this direction by allowing Unicode variable names. The math alphanumerics allow this approach to go further with relatively little effort for compilers. 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 won't be able to make the debug windows use the math-oriented characters unless it can handle the underlying Unicode characters.

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 us figure out how to accomplish the ultimate goal of teaching computers to understand and use arbitrary mathematical expressions.

4 Unicode Plain Text Encoding of Mathematics

Unicode plus a few special symbols can encode most mathematical expressions in readable 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 new strong support for mathematical symbols. This plain-text approach is compared to the LaTeX dialect of TeX, "Unicode TeX", and MathML. The plain-text representation is substantially more compact and easy to read. Keyboard input methods are discussed. One use of the plain-text format is as a math input method, both for search text and for general editing. Most mathematical expressions up through calculus can be represented unambiguously in Unicode plain text. Export to (La)TeX, MathML, C++, and symbolic manipulation programs is outlined. Note that the discussion is only intended to show how mathematical plain text might be useful. It isn't intended to be a complete specification or to be used for general information interchange at this stage in its development.

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 we use 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 don't look very much like the expressions they represent. It's certainly not easy to make algebraic calculations using TeX's notation. With Unicode, we 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 we define a simple operand to consist of all consecutive non-operator characters. We call this sequence of one or more characters a span of non-operators. As such, a simple numerator or denominator is terminated by any operator, including, for example, arithmetic operators, the blank operator, 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 we depict 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 op-

erators, that set is not displayed in built-up form, since usually one doesn't 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. 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's 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>
```

Nature isn't so kind with subscripts and superscripts, but they're still quite readable. Specifically, we introduce a subscript by a subscript operator with its own special glyph that resembles a subscripted down arrow ↓. The subscript itself can be any operand as defined above. Another compound subscript is a subscripted subscript, which works using right-to-left associativity, e.g., a↓b↓c means a_{b_c}. Similarly a↑b↑c means a^{b^c}.

As an example of a slightly more complicated example, consider the expression $W_{\delta_1\rho_1\sigma_2}^{3\beta}$ has the plain-text format W↑3β↓δ₁ρ₁σ₂. In contrast, for TeX, you type

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

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↑3β↓δ₁ρ₁σ₂.

For the ratio

$$\frac{\alpha_2^3}{\beta_2^3 + \gamma_2^3} ,$$

the Unicode plain text reads α_2^3/(β_2^3 + γ_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↑β↓δ₁ρ₁σ₂ = U↑3β↓δ₁ρ₁ + 1/8π² ∫↓α₁↑α₂ dα₂' [(U↑2β↓δ₁ρ₁ - α₂'U↑1β↓ρ₁σ₂)/U↑0β↓ρ₁σ₂]
```

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 we use U+2032 for \prime and U+2044 for \over.

4.1 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 patently identify themselves as mathematical, obviating the need to declare them explicitly as such. One of TeX’s greatest limitations is its inability to detect expressions that are obviously mathematical, but that are not enclosed within $’s. To complicate matters, the popular TeX dialects use the $ as a toggle,

which is a poor choice as a myriad TeX users will loudly testify! It's quite frustrating to leave out a $ by mistake and thereby receiving 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 you basically have 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's not yet available in an object-oriented machine-readable form, into that form. A similar recognition problem exists for pen entry of equations.

It's 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 can be used to overrule these choices. Ultimately it 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.

As described above, a simple subscript operand consists of the string of all non-operators that follow the subscript operator. Compound subscripts include expressions within parentheses, square brackets, and curly braces. In addition it's 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.

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 are *not* italicized. These include trigonometric function names like sin and cos, as well as ln, cosh, etc. Words or abbreviations, often used as subscripts (see program in Sec. 4.3), 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. One needs a symbol that 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. Such "overhead" symbols clutter up the text and hopefully will be rarely needed in Unicode plain text. The method I've used up to now is to introduce a special override symbol to force the behavior desired. This does complicate the preparation of technical documents and although you can get very good at it, it's not the most user-friendly way of doing things. On the other hand, identifying the beginning and end of math expressions using $'s isn't user friendly either.

4.2 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 we define 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 is

FF CR \
([{
)]}|
Space ".,=-+ LF Tab
/*×·••⏐
■√
∫ Σ Π
↓ ↑

where 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 $\alpha + \beta/\gamma$ means

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

As in arithmetic, precedence can be overruled, so $(\alpha + \beta)/\gamma$ 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 $^{0-9+-()}$ 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 $_{0-9+-()}$ 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.

4.3 Export to Programming and Markup Languages

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. In the middle 1950's, the authors of FORTRAN named their computer language after FORmula TRANslation, but they only went half way. Arithmetic expressions in Fortran and other current high-level languages still don't look like mathematical formulas and considerable human coding effort is needed to translate formulas into their machine comprehensible counterparts. Whitehead once said that 90% of mathematics is notation and that a perfect notation would be a substitute for thought. From this point of view, modern computer languages are badly lacking.

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 we 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, we start with our Unicode plain-text format.

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. The expressions can contain 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 operations 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 us 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. Of course if we could convince our compiler friends to allow use to use Unicode for program-variable names, we'd really have it made! Compatibility with unenlightened ASCII-only compilers could be done via an ASCII representation of Unicode characters.

4.4 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 ;
}
```

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
        {
                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 ;
}
```

The ability to use the second and third versions of the program was built into the PS Technical Word Processor. With it we already come 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, you copy (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 are 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 P. Meystre and M. Sargent III (1991), *Elements of Quantum Optics*, Springer-Verlag].

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's 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 isn't 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 you want to see the comments, you widen the right pane accordingly. On the other hand, to view lines with many characters of code, the // comments needn't get in the way. Such a dual-pane facility would also be great for working with assembly-language programs.

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 names via table lookup and hashing, as for the program translations. Better yet, TeX should be recompiled to use Unicode.

4.5 Conclusions

We have 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.

5 References

[MathML] http://www.w3.org/mathml
[TeX] http://www.ams.org/tex/publications.html
[LaTeX]
[STIX] http://www.ams.org/STIX.

TODO: describe the following:

Double struck Greek and Italic in 2100 block
- special use in CAS

Squares
- call out the graduated sequence

Tilde/lazy S
- describe the unification

Terminal symbols
- describe the scan lines and blocks