Unicode Nearly Plain-Text Encoding of Mathematics Version 3

Murray Sargent III
Publisher Text Services, Microsoft Corporation
10-Mar-10

1. Introduction

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[1] can encode many mathematical expressions in readable nearly plain text. Technically this format is a "lightly marked up format"; hence the use of "nearly". The format is linear, but it can be displayed in built-up presentation form. To distinguish the two kinds of formats in this paper, we refer to the nearly plain-text format as the *linear format* and to the built-up presentation format as the *built-up format*. This linear format can be used with heuristics based on the Unicode math properties to recognize mathematical expressions without the aid of explicit math-on/off commands. The recognition is facilitated by Unicode's strong support for mathematical symbols.[2] Alternatively, the linear format can be used in "math zones" explicitly controlled by the user either with on-off characters as used in TeX or with a character format attribute in a rich-text environment. Use of math zones is desirable, since the recognition heuristics are not infallible.

The linear format is more compact and easy to read than [La]TeX,[3,4] or MathML.[5] However unlike those formats, it doesn't attempt to include all typographical embellishments. Instead we feel it's useful to handle some embellishments in the higher-level layer that handles rich text properties like text and background colors, font size, footnotes, comments, hyperlinks, etc. In principle one can extend the notation to include the properties of the higher-level layer, but at the cost of reduced readability. Hence embedded in a rich-text environment, the linear format can faithfully represent rich mathematical text, whereas embedded in a plain-text environment it lacks most rich-text properties and some mathematical typographical properties. The linear format is primarily concerned with presentation, but it has some semantic features that might seem to be only content oriented, e.g., *n*-

aryands and function-apply arguments (see Secs. 3.4 and 3.5). These have been included to aid in displaying built-up functions with proper typography, but they also help to interoperate with math-oriented programs.

Most mathematical expressions can be represented unambiguously in the linear format, from which they can be exported to [La]TeX, MathML, C++, and symbolic manipulation programs. The linear format borrows notation from TeX for mathematical objects that don't lend themselves well to a mathematical linear notation, e.g., for matrices.

A variety of syntax choices can be used for a linear format. The choices made in this paper favor a number of criteria: efficient input of mathematical formulae, sufficient generality to support high-quality mathematical typography, the ability to round trip elegant mathematical text at least in a rich-text environment, and a format that resembles a real mathematical notation. Obviously compromises between these goals had to be made.

The linear format is useful for 1) inputting mathematical expressions,[6] 2) displaying mathematics by text engines that cannot display a built-up format, and 3) computer programs. For more general storage and interchange of math expressions between math-aware programs, MathML and other higher-level languages are preferred.

Section 2 motivates and illustrates the linear format for math using the fraction, subscripts, and superscripts along with a discussion of how the ASCII space U+0020 is used to build up one construct at a time. Section 3 summarizes the usage of the other constructs along with their relative precedences, which are used to simplify the notation. Section 4 discusses input methods. Section 5 gives ways to recognize mathematical expressions embedded in ordinary text. Section 6 explains how Unicode plain text can be helpful in programming languages. Section 7 gives conclusions. The appendices present a simplified linear-format grammar and a partial list of operators.

2. *Encoding Simple Math Expressions*

Given Unicode's strong support for mathematics[2] relative to ASCII, how much better can a plain-text encoding of mathematical expressions look using Unicode? The most well-known ASCII encoding of such expressions is that of TeX, so we use it for comparison. MathML is more verbose than TeX and 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 representations of mathematical expressions reveals that they do not look very much like the expressions they represent. It's not easy to make algebraic calculations by hand directly using TeX's notation. With Unicode, one can represent mathematical expressions more readably, and the resulting nearly plain text can often be used with few or no modifications for such calculations. This capability is considerably enhanced by using the linear format in a system that can also display and edit the mathematics in built-up form.

The present section introduces the linear format with fractions, subscripts, and superscripts. It concludes with a subsection on how the ASCII space character U+0020 is used to build up one construct at a time. This is a key idea that makes the linear format ideal for inputting mathematical formulae. In general where syntax and semantic choices were made, input convenience was given high priority.

2.1 Fractions

One way to specify a fraction linearly is LaTeX's \frac{numerator}{denominator}. The { } are not printed when the fraction is built up. These simple rules immediately give a "plain text" that is unambiguous, but looks quite different from the corresponding mathematical notation, thereby making it harder to read.

Instead we define a simple operand to consist of all consecutive letters and decimal digits, i.e., a span of alphanumeric characters, those belonging to the L*x* and Nd General Categories (see *The Unicode Standard* 5.0,[1] Table 4-2. General Category). As such, a simple numerator or denominator is terminated by most nonalphanumeric characters, including, for example, arithmetic operators, the blank (U+0020), and Unicode characters in the ranges U+2200..U+23FF, U+2500..U+27FF, and U+2900..U+2AFF. The fraction operator is given by the usual solidus / (U+002F). So the simple built-up fraction

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

appears in linear format as *abc*/*d*. To force a display of a normal-size linear fraction, one can use \/ (backslash followed by slash).

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 parentheses are preceded and followed by operators, those parentheses are not displayed in built-up form, since usually one does not want to see such parentheses. So the plain text (*a* + *c*)/*d* displays as

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

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

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

A really neat feature of this notation is that the plain text is, in fact, often a legitimate mathematical notation in its own right, so it is relatively easy to read. Contrast this with the MathML version, which (with no parentheses) reads as

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

Three built-up fraction variations are available: the “fraction slash” U+2044 (which one might input by typing \sdiv) builds up to a skewed fraction, the “division slash” U+2215 (\ldiv) builds up to a potentially large linear fraction, and the circled slash ⊘ (U+2298, \ndiv) builds up a small numeric fraction (although characters other than digits can be used as well). The three kinds of built-up fractions are illustrated by

$$\frac{\frac{a}{b+c}}{\frac{d}{e}+f}, \qquad \frac{a}{b+c}\Big/\frac{d}{e}+f, \qquad \left(\frac{a}{b+c}\right)\Big/\left(\frac{d}{e}+f\right)$$

When building up the large linear fraction, the outermost parentheses should not be removed.

The same notational syntax is used for a “stack” which is like a fraction with no fraction bar. The stack is used to create binomial coefficients and the stack operator is ‘¦’ (\atop). For example, the binomial theorem

$$(a+b)^n = \sum_{k=0}^{n} \binom{n}{k} a^k b^{n-k}$$

in linear format reads as (see Sec. 3.4 for a discussion of the *n*-aryand “glue” operator ▒)

$$(a + b)\text{^}n = \sum\_(k{=}0)\text{^}n \text{▒} (n \text{¦} k)\ a\text{^}k\ b\text{^}(n\text{-}k),$$

where (*n* ¦ *k*) is the binomial coefficient for the combinations of *n* items grouped *k* at a time. The summation limits use the subscript/superscript notation discussed in the next subsection.

Since binomial coefficients are quite common, TeX has the \choose control word for them. In the linear format Version 3, this uses the \choose operator (c) instead of the \atop operator ¦. Accordingly the binomial coefficient in the binomial theorem above can be written as “n\choose k”, assuming that you type a space after the k. This shortcut is included primarily for compatibility with TeX, since (n¦k) is pretty easy to type.

When / is followed by an operator, it’s highly unlikely that a fraction is intended. This fact leads to a simple way to enter *negated* operators like ≠, namely, just

type /= to get ≠. A list of such negated operator combinations is given in Section 4.1. To enter ≠, you can also type TeX's name, \ne, but /= is slightly simpler. And the TeX names for the other negated operators in Section 4.1 are harder to remember. One other trick with fractions is that a period or comma in between two digits or in between the slash and a digit is considered to be part of a number, rather than being a terminator. For example 1/3.1416 builds up to $\frac{1}{3.1416}$, rather than $\frac{1}{3}$.1416.

2.2 Subscripts and Superscripts

Subscripts and superscripts are a bit trickier, but they're still quite readable. Specifically, we introduce a subscript by a subscript operator, which we display as the ASCII underscore _ as in TeX. 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 we display as the ASCII ^ as in TeX. So a^b means a^b. A nice enhancement for a text processing system with build-up capabilities is to display the _ as a small subscript down arrow and the ^ as a small superscript up arrow, in order to convey the semantics of these build-up operators in a math context.

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 an alphanumeric is treated as part of the subscript. This refinement obviates the need for many overriding parentheses, thereby yielding a more readable linear-format text (see Sec. 3.14 for more discussion of comma and period).

Another kind of compound subscript is a subscripted subscript, which works using right-to-left associativity, e.g., a_b_c stands for a_{b_c}. Similarly a^b^c stands for a^{b^c}.

Parentheses are needed for constructs such as a subscripted superscript like a^{b_c}, which is given by a^(b_c), since a^b_c displays as a_c^b (as does a_c^b). The build-up program is responsible for figuring out what the subscript or superscript base is. Typically the base is just a single math italic character like the *a* in these examples. But it could be a bracketed expression or the name of a mathematical function like sin as in sin^2 *x*, which renders as $\sin^2 x$ (see Sec. 3.5 for more discussion of this case). It can also be an operator, as in the examples $+_1$ and $=_2$. In Indic and other cluster-oriented scripts the base is by default the cluster preceding the subscript or superscript operator.

As an example of a slightly more complicated example, consider the expression $W^{3\beta}_{\delta_1\rho_1\sigma_2}$, which can be written with the linear format W^3β_δ₁ρ₁σ₂, where Unicode numeric subscripts are used. In TeX, one types

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

The TeX version looks simpler using Unicode for the symbols, namely $W^{3β}_{δ_1 ρ_σ_2}$ or $W^{3β}_{δ1ρ1σ2}$, since Unicode has a full set of decimal subscripts and superscripts. As a practical matter, numeric subscripts are typically entered using an underscore and the number followed by a space or an operator, so the major simplification is that fewer brackets are needed.

For the ratio

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

the linear-format text can read as $\alpha_2^3/(\beta_2^3 + \gamma_2^3)$, while the standard TeX version reads as

$$\alpha_2^3 \over \beta_2^3 + \gamma_2^3$$.

The linear-format text is a legitimate mathematical expression, while the TeX version bears no resemblance to a mathematical expression.

TeX becomes cumbersome for longer equations such as

$$W^{3\beta}_{\delta_1\rho_1\sigma_2} = U^{3\beta}_{\delta_1\rho_1} + \frac{1}{8\pi^2}\int_{\alpha_1}^{\alpha_2} d\alpha'_2 \left[\frac{U^{2\beta}_{\delta_1\rho_1} - \alpha'_2 U^{1\beta}_{\rho_1\sigma_2}}{U^{0\beta}_{\rho_1\sigma_2}}\right]$$

A linear-format version of this reads as

W_δ1ρ1σ2^3β=U_δ1ρ1^3β+1/8π^2 ∫ _α1^α2▒dα'2 [(U_δ1ρ1^2β-α'2 U_ρ1σ2^1β)/U_ρ1σ2^0β]

while the standard TeX version reads as

```
$$W_{\delta_1\rho_1\sigma_2}^{3\beta}
      = U_{\delta_1\rho_1}^{3\beta} + {1 \over 8\pi^2}
      \int_{\alpha_1}^{\alpha_2} d\alpha_2' \left[
      {U_{\delta_1\rho_1}^{2\beta} - \alpha_2'
      U_{\rho_1\sigma_2}^{1\beta} \over
      U_{\rho_1\sigma_2}^{0\beta}} \right] $$ .
```

2.3 Use of the Blank (Space) Character

The ASCII space character U+0020 is rarely needed for explicit spacing of built-up text since the spacing around operators should be provided automatically by the math display engine (Sec. 3.16 discusses this automatic spacing). However the space character is very useful for delimiting the operands of the linear-format notation. When the space plays this role, it is eliminated upon build up. So if you type \alpha

followed by a space to get α, the space is eliminated when the α replaces the \alpha. Similarly *a*_1 *b*_2 builds up as $a_1 b_2$ with no intervening space.

Another example is that a space following the denominator of a fraction is eliminated, since it causes the fraction to build up. If a space precedes the numerator of a fraction, the space is eliminated since it may be necessary to delimit the start of the numerator. Similarly if a space is used before a function-apply construct (see Sec. 3.5) or before above/below scripts (see Sec. 3.3), it is eliminated since it delimits the start of those constructs.

In a nested subscript/superscript expression, the space builds up one script at a time. For example, to build up *a*^*b*^*c* to a^{b^c}, two spaces are needed if spaces are used for build up. Some other operator like + builds up the whole expression, since the operands are unambiguously terminated by such operators.

In TeX, the space character is also used to delimit control words like \alpha and does not appear in built-up form. A difference between TeX's usage and the linear format's is that in TeX, blanks are invariably eliminated in built-up display, whereas in the linear format blanks that don't delimit operands or keywords do result in spacing. Additional spacing characters are discussed in Sec. 3.16.

One displayed use for spaces is in overriding the algorithm that decides that an ambiguous unary/binary operator like + or − is unary. If followed by a space, the operator is considered to be binary and the space isn't displayed. Spaces are also used to obtain the correct spacing around comma, period, and colon in various contexts (see Sec. 3.14).

3. Encoding Other Math Expressions

The previous section describes how we encode fractions, subscripts and superscripts in the linear format and gives a feel for that format. The current section describes how we encode other mathematical constructs using this approach and ends with a more formal discussion of the linear format.

3.1 Delimiters

Brackets [ ], braces { }, and parentheses ( ) represent themselves in the Unicode plain text, and a word processing system capable of displaying built-up formulas should be able to enlarge them to fit around what's inside them. In general we refer to such characters as *delimiters*. A delimited pair need not consist of the same kinds of delimiters. For example, it's fine to open with [and close with } and one sees this usage in some mathematical documents. The closing delimiter can have a subscript and/or a superscript. Delimiters are called *fences* in MathML.

These choices suffice for most cases of interest. But to allow for use of a delimiter without a matching delimiter and to overrule the open/close character of delimiters, the special keywords \open and \close can be used. These translate to the box-drawings characters ├ and ┤ , respectively. Box drawings characters are used for the

open/close delimiters because they aren't likely to be used as mathematical characters and they are readily available in fonts. If used before any character that isn't a delimiter of the opposite sense, the open/close delimiter acts as an invisible delimiter, defining the corresponding end of a delimited expression. A common use of this is the "cases" equation, such as

$$|x| = \begin{cases} x \text{ if } x \geq 0 \\ -x \text{ if } x < 0 \end{cases},$$

which has the linear format "|x| = {▉ (&x" if "x ≥ 0@−&x" if "x < 0)┤ " (see Sec. 3.19 for a discussion of the equation-array operator ▉).

Because the cases construct is fairly common, TeX has the \cases control word for it. This can be implemented in the linear format Version 3 with the \cases operator ©. With this the equation above can be written as "|x| = ©(&x" if "x ≥ 0@−&x" if "x < 0)", which is still a little strange, but you don't have to type the opening curly brace and \close.

The open/close delimiters can be used to overrule the normal open/close character of delimiters as in the admittedly strange, but nevertheless sometimes used, expression "]$a + b$[", which has the linear format " ├]a+b┤ [". Note that a blank following an open or close delimiter is "eaten". This is to allow an open delimiter to be followed by a normal delimiter without interpreting the pair as a single delimiter. See also Sec. 3.18 on how to make arbitrary groupings. If a ├ needs to be treated as an empty open delimiter when it appears before a delimiter like | or], follow the ├ by a space to force the open-delimiter interpretation.

To suppress automatic sizing and to choose specific sizes, ├ is followed by a digit '0' –'4' with the meanings in the following table

Digit	Meaning
0	Don't grow
1	TeX big
2	TeX Big
3	TeX bigg
4	TeX Bigg

It's rarely necessary to use explicit sizes if the display system can break equations within bracketed expressions.

The usage of open and close delimiters in the linear format is admittedly a compromise between the explicit nature of TeX and the desire for a legitimate math notation, but the flexibility can be worth the compromise especially when interoperating with ordinarily built-up text such as in a WYSIWYG math system. TeX uses \left and \right for this purpose instead of \open and \close. We use the latter since they apply to right-to-left mathematics used in many Arabic locales as well as to the usual left-to-right mathematics.

Absolute values are represented by the ASCII vertical bar | (U+007C). The evenness of its count at any given bracket nesting level typically determines whether

the vertical bar is a close |. Specifically, the first appearance is considered to be an open | (unless subscripted or superscripted), the next a close | (unless following an operator), the next an open |, and so forth.

Nested absolute values can be handled unambiguously by discarding the outermost parentheses within an absolute value. For example, the built-up expression $\big||x| - |y|\big|$ can have the linear format |(|*x*|−|*y*|)|. Some cases, such as this one, can be parsed without the clarifying parentheses by noting that a vertical bar | directly following an operator is an open |. But the example |*a*|*b*−*c*|*d*| needs the clarifying parentheses since it can be interpreted as either (|*a*|*b*)−(*c*|*d*|) or |*a*(|*b*−*c*|)*d*|. The usual algorithm gives the former, so if one wants the latter without the inner parentheses, one can type |(*a*|*b*−*c*|*d*)|.

Another case where we treat | as a close delimiter is if it is followed by a space (U+0020). This handles the important case of the bra vector in Dirac notation. For example, the quantum mechanical density operator ρ has the definition

$$\rho = \sum_{\psi} P_{\psi} |\psi\rangle\langle\psi|,$$

where the vertical bars can be input using the ASCII vertical bar.

If a | is followed by a subscript and/or a superscript and has no corresponding open |, it is treated as a script base character, i.e., *not* a delimiter. Its built-up size should be the height of the integral sign in the current display/inline mode.

The Unicode norm delimiter U+2016 (‖ or \norm) has the same open/close definitions as the absolute value character | except that it's always considered to be a delimiter.

Delimiters can also have separators within them. Version 2 of the linear format doesn't formalize the comma separators of function arguments (MathML does), but it supports the vertical bar separator \vbar, which is represented by the box drawings light vertical character │ (U+2502). We tried using the ASCII | (U+007C) for this purpose too, but the resulting ambiguities are insurmountable in general. One case using U+007C as a separator that can be deciphered is that of the form (*a*|*b*), where *a* and *b* are mathematical expressions. But (*a*|*b*|*c*) interprets the vertical bars as the absolute value. And one might want to interpret the | in (*a*|*b*) as an open delimiter with) as the corresponding close delimiter, while the (isn't yet matched. If so, precede the | by ⊢, i.e., (⊢|*b*). The vertical bar separator grows in size to match the size of the surrounding brackets. In Version 3, other operators can be treated as separators by preceding them with \middle (‖— U+2551).

Another common separator is the \mid character ∣ (U+2223), commonly used in expressions like $\{x \mid f(x) = 0\}$. This separator also grows in size to match the surrounding brackets and is spaced as a relational operator.

3.2 Literal Operators

Certain operators like brackets, braces, parentheses, superscript, subscript, integral, etc., have special meaning in the linear-format notation. In fact, even a charac-

ter like '+', which displays the same glyph in linear format as in built-up form (aside from a possible size reduction), plays a role in the linear format in that it terminates an operand. To remove the linear-format role of such an operator, we precede it by the "literal operator", for which the backslash \ is handy. So \[is displayed as an ordinary left square bracket, with no attempt by the build-up software to match a corresponding right square bracket. Such *quoted* operators are automatically included in the current operand.

Linear format operators always consist of a single Unicode character, although a control word like \open may be used to input the character. Using a single character has the advantage of being globalized, since the control word typically looks like English. Users can define other control words that look like words in other languages just so long as they map into the appropriate operator characters. A slight exception to the single-character operator rule occurs for accent operators that are applied to two or more characters (see Sec. 3.10). For these the accent combining mark may be preceded by a no-break space for the sake of readability. Another advantage of using operator characters rather than control words is that the build-up processing is simplified and therefore faster. And one should delight in the fact that the operator characters look like the operators they represent, while the control words do not.

3.3 Prescripts and Above/Below Scripts

A special parenthesized syntax is used to form prescripts, that is, subscripts and superscripts that precede their base. For this (_c^b)a creates the prescripted variable ${}_c^b a$. Variables can have both prescripts and postscripts (ordinary subscripts and superscripts).

In Version 3 of the linear format, you can use a prescript notation similar to TeX's. Just type a subscript and/or a superscript not preceded by a base and then follow it with a character that can be used as a base. For the ${}_c^b a$ example, you type _c^b a. Note that you need to terminate the superscript with a space. If a variable precedes the prescript, you also need to precede the prescript with a space. A common use of prescripts is for the confluent hypergeometric functions, such as ${}_1F_1$. In Version 3, this can be input as _1 F_1 or as (_1^)F_1.

Below scripts and above scripts are represented in general by the line drawing operators \below (⏉) and \above (⏊), respectively. Hence the expression $\lim_{n\to\infty} a_n$ can be represented by lim⏉ (n→∞) a_n. Since the operations det, gcd, inf, lim, lim inf, lim sup, max, min, Pr, and sup are common, their below scripts are also accessible by the usual subscript operator _. So in display mode, $\lim_{n\to\infty} a_n$ can also be represented by lim_(n→∞) a_n, which is a little easier to type than lim⏉(n→∞) a_n.

Although for illustration purposes, the belowscript examples are shown here in-line with the script below, ordinarily this choice is only for display-mode math. When inline, below- and abovescripts entered with _ and ^ are shown as subscripts

and superscripts, respectively, as are the limits for *n*-ary operators. When entered with ⊤ and ⊥, they remain below and above scripts in-line. If an above/below operator or a subscript/superscript operator is preceded by an operator, that operator becomes the base. See Sec. 3.8 for some examples.

3.4 *n*-ary Operators

n-ary operators like integral, summation and product are sub/superscripted or above/below operators that have a third argument: the "*n*-aryand". For the integral, the *n*-aryand is the integrand, and for the summation, it's the summand. For both typographical and semantic purposes, it's useful to identify these *n*-aryands. In the linear format, this is done by following the sub/superscripted *n*-ary operator by the naryand concatenation operator \naryand (▒) which is U+2592. The operand that follows this operator becomes the *n*-aryand. For example, the linear-format expression ∫_0^a▒xⅆx/(x^2+a^2) has the built up form

$$\int_0^a \frac{x\,dx}{x^2+a^2}$$

where xⅆx/(x^2+a^2) is the integrand and ⅆ is the Unicode differential character U+2146. Unlike with the fraction numerator and denominator, the outermost parentheses of a *n*-aryand are *not* removed on buildup, since parentheses are commonly used to delimit compound *n*-aryands. Notice that the ⅆ character automatically leads to a small space between the *x* and the *dx* and by default displays as a math-italic *d* when it appears in a math zone.

To delimit more complicated *n*-aryands without using parentheses or brackets of some kind, use the \begin \end (〖〗 see Sec. 3.18) delimiters, which disappear on build up.

Since \naryand isn't the most intuitive name, the alias \of can be used. This also works as an alias for \funcapply in math function contexts (see Sec. 3.5). This alias is motivated by sentences like "The integral from 0 to b of xdx is one-half b squared."

Sometimes one wants to control the positions of the limit expressions explicitly as in using TeX's \limits (upper limit above, lower below) and \nolimits (upper limit as superscript and lower as subscript) control words. To this end, if the *n*-ary operator is followed by the digit 1, the limit expressions are displayed above and below the *n*-ary operator and if followed by the digit 2, they are displayed as superscript and subscript. More completely, the number can be one of the first four of the following, OR'd with any of the next three (which were added in Version 3), along with neither or one of the last two

nLimitsDefault	0
nLimitsUnderOver	1
nLimitsSubSup	2

nUpperLimitAsSuperScript	3
nLimitsOpposite	4
nShowLowLimitPlaceHolder	8
nShowUpLimitPlaceHolder	16
fDontGrowWithContent	64
fGrowWithContent	128

3.5 Mathematical Functions

Mathematical functions such as trigonometric functions like "sin" should be recognized as such and not italicized. As such they are treated as ordinary text (see Sec. 3.16). In addition it's desirable to follow them with the Invisible Function Apply operator U+2061 (\funcapply). This is a special binary operator and the operand that follows it is the function argument. In converting to built-up form, this operator transforms its operands into a two-argument object that renders with the proper spacing for mathematical functions.

If the Function Apply operator is immediately followed by a subscript or superscript expression, that expression should be applied to the function name and the Function Apply operator moved passed the modified name to bind the operand that follows as the function argument. For example, the function $\sin^2 x$ falls into this category.

Unlike with the fraction numerator and denominator, the outermost parentheses of the second operand of the function-apply operator are not removed on buildup, since parentheses are commonly used to delimit function arguments. To delimit a more complicated arguments without using parentheses or brackets of some kind, use the ⟦ ⟧ delimiters (\begin \end) which disappear on build up. If brackets are used, they and their included content comprise the function's argument. For example, $\sin(x)\,b$ means $\sin(x) \times b$. To get $\sin(\omega - \omega_0)t$, where t is part of the argument, one can use sin\funcapply(\omega-\omega_0)t, or enclose the argument in ⟦ ⟧ delimiters.

Since \funcapply isn't the most intuitive name, \of can be used in function-apply contexts. \of autocorrects to ▒ (U+2592—\naryand, see Sec. 3.4), but context can give it this convenient second use. This alias is motivated by sentences like "The sine of 2x equals twice the sine of x times the cosine of x", i.e., $\sin 2x = 2 \sin x \cos x$.

If a function name has a space in it, e.g., "lim sup", the space is represented by a no-break space (U+00A0) as described in Sec. 3.16. If an ordinary ASCII space were used, it would imply build up of the "lim" function.

3.6 Square Roots and Radicals

Square, cube, and quartic roots can be represented by expressions started by the corresponding Unicode radical characters √ (U+221A, \sqrt), ∛ (U+221B, \cbrt), and ∜ (U+221C, \qdrt). These operators include the operand that follows. Examples

are √*abc*, √(*a*+*b*) and ∛(*c*+*d*), which display as $\sqrt{abc}$, $\sqrt{a+b}$, and $\sqrt[3]{c+d}$, respectively. In general, the *n*th root radical is represented by an expression like √(*n*&*a*), where *a* is the complete radicand. Anything following the closing parenthesis is not part of the radicand. For example, √(*n*&*a* + *b*) displays as $\sqrt[n]{a+b}$.

In Version 3 of the linear format, you can obtain $\sqrt[n]{a+b}$ using more TeX-like input \root n\of(a+b). In this format, the degree of the radical can be more than one character without enclosing it in parentheses. For example, $\sqrt[n+1]{b+c}$ can be input by \root n+1\of(b+c), which is similar to TeX's \root n+1\of{b+c}.

3.7 Enclosures

To enclose an expression in a rectangle one uses the rectangle operator ▭ (U+25AD, \rect) followed by the operand representing the expression. This syntax is similar to that for the square root. For example ▭(*E* = *mc*^2) displays as $\boxed{E = mc^2}$. The same approach is used to put an overbar above an expression, namely follow the overbar operator ¯ (U+00AF, \overbar) by the desired operand. For an underbar, use the operator ⎯ (U+2581, \underbar).

In general the rectangle function can represent any combination of borders, horizontal, vertical, and diagonal strikeouts, and enclosure forms defined by the MathML <menclose> element, except for roots, which are represented as discussed in the previous Section. The general syntax for enclosing an expression *x* is ▭(*n*&*x*), where *n* is a mask consisting of any combination of the following flags:

fBoxHideTop	1
fBoxHideBottom	2
fBoxHideLeft	4
fBoxHideRight	8
fBoxStrikeH	16
fBoxStrikeV	32
fBoxStrikeTLBR	64
fBoxStrikeBLTR	128

It is anticipated that the enclosure format number *n* is chosen via some kind of friendly user interface, but at least the choice can be preserved in the linear format. Note that the overbar function can also be given by ▭(2&*x*) and the underbar by ▭(8&*x*).

Other enclosures such as rounded box, circle, long division, actuarial, and ellipse can be encoded as for the rectangle operator but using appropriate Unicode characters (not yet chosen here).

An abstract box can be put around an expression *x* to change alignment, spacing category, size style, and other properties. This is defined by □(*n*&*x*), where □ is U+25A1 (\box) and *n* can be a combination of one Align option, one Space option, one Size option and any flags in the following table:

nAlignBaseline	0
nAlignCenter	1
nSpaceDefault	0
nSpaceUnary	4
nSpaceBinary	8
nSpaceRelational	12
nSpaceSkip	16
nSpaceOrd	20
nSpaceDifferential	24
nSizeDefault	0
nSizeText	32
nSizeScript	64
nSizeScriptScript	96
fBreakable	128
fXPositioning	256
fXSpacing	512

3.8 Stretchy Characters

In addition to overbars and underbars, stretchable brackets are used in mathematical text. For example, the “underbrace” and “overbrace” are as

$$\overbrace{x+\cdots+x}^{k \text{ times}}$$

$$\underbrace{x+y+z}_{>0}$$

The linear formats for these are ⏞(x+⋯+x)^(k "times") and ⏟(x+y+z)_(>0), respectively. Here the subscript and superscript operators are used for convenient keyboard entry (and compatibility with TeX); one can also use Sec. 3.3’s belowscript and abovescript operators, respectively. The horizontal stretchable brackets are given in the following table

U+23DC	⏜	\overparen
U+23DD	⏝	\underparen
U+23DE	⏞	\overbrace
U+23DF	⏟	\underbrace
U+23E0	⏠	\overshell
U+23E1	⏡	\undershell
U+23B4	⎴	\overbracket
U+23B5	⎵	\underbracket

There are many other characters that can stretch horizontally to fit text, such as various horizontal arrows. There are four configurations: a stretch character above or below a baseline text, and text above or below a baseline stretched character. Illustrating the linear format for these four cases with the stretchy character → and the text $a + b$, we have

$(a+b)\perp\rightarrow$	$\overrightarrow{a+b}$
$(a+b)\top\rightarrow$	$\underrightarrow{a+b}$
$\rightarrow \perp a+b$	$\overset{a+b}{\longrightarrow}$
$\rightarrow \top(a+b)$	$\underset{a+b}{\longrightarrow}$

3.9 Matrices

Matrices are represented by a notation very similar to TeX's, namely an expression of the form

■ (exp_1 [& exp_2]... @ ... $exp_{n\text{-}1}$ [& exp_n]...)

where ■ is the matrix character U+25A0 and @ is used to terminate rows, except for the last row which is terminated by the closing paren. This causes exp_1 to be aligned over $exp_{n\text{-}1}$, etc., to build up an $n \times m$ matrix array, where n is the maximum number of elements in a row and m is the number of rows. The matrix is constructed with enough columns to accommodate the row with the largest number of entries, with rows having fewer entries given sufficient null entries to keep the table $n \times m$. As an example, ■(a&b@c&d) displays as

$$\begin{matrix} a & b \\ c & d \end{matrix}$$

If you want parentheses around the matrix, include them as in (■(a&b@c&d)) Because parenthesized matrices are quite common, TeX has the \pmatrix control word that automatically includes parentheses. This is implemented in the linear format Version 3 with the \pmatrix operator ㎙. So ㎙(a&b@c&d) displays as

$$\begin{pmatrix} a & b \\ c & d \end{pmatrix}$$

3.10 Accent Operators

Mathematics often has accented characters. Simple primed characters like a' are represented by the character followed by the Unicode prime U+2032, which can be typed in using the ASCII apostrophe '. Double primed characters have two Unicode primes, etc. In addition, Unicode has multiple prime characters that render with somewhat different spacing than concatenations of U+2032. The primes are

special in that they need to be superscripted with appropriate use of heavier glyph variants (see Sec. 3.12).

The ASCII asterisk is raised in ordinary text, but in a math zone it gets translated into U+2217, which is placed on the math axis as the +. To make it a superscript or subscript, the user has to include it in a superscript or subscript expression. For example, a^{*2} has the linear format version a^*2 or a^(*2). Here for convenience, the asterisk is treated as an operand character if it follows a subscript or superscript operator.

Other kinds of accented characters can be represented by Unicode combining mark sequences. The combining marks are found in the Unicode ranges U+0300—U+036F and U+20D0 – U+20FF. The most common math accents are summarized in the following table

\hat	U+0302	$\hat{a}$
\check	U+030C	$\check{a}$
\tilde	U+0303	$\tilde{a}$
\acute	U+0301	$\acute{a}$
\grave	U+0300	$\grave{a}$
\dot	U+0307	$\dot{a}$
\ddot	U+0308	$\ddot{a}$
\dddot	U+20DB	$\dddot{a}$
\bar	U+0304	$\bar{a}$
\vec	U+20D7	$\vec{a}$

If a combining mark should be applied to more than one character or to an expression, that character or expression should be enclosed in parentheses and followed by the combining mark. Since this construct looks funny when rendered by plain-text programs, a no-break space (U+00A0) can appear in between the parentheses and the combining mark. For example, (*a* + *b*)^ renders as $\widehat{a+b}$ when built up. Special cases of this notation include overscoring (use U+0305) and underscoring (use U+0332) mathematical expressions.

The combining marks are treated by a mathematics renderer as operators that translate into special accent built-up functions with the proper spacing for mathematical variables.

3.11 Differential, Exponential, and Imaginary Symbols

Unicode contains a number of special double-struck math italic symbols that are useful for both typographical and semantic purposes. These are U+2145—U+2149 for double-struck *D*, *d*, *e*, *i*, and *j* ($\mathbb{D}$, $\mathbb{d}$, $\mathbb{e}$, $\mathbb{i}$, $\mathbb{j}$), respectively. They have the meanings of differential, differential, natural exponent, imaginary unit, and imaginary unit, respectively.

In US patent applications these characters should be rendered as $\mathbb{D}$, $\mathbb{d}$, $\mathbb{e}$, $\mathbb{i}$, $\mathbb{j}$ as defined, but in regular US technical publications, these quantities can be rendered as math italic. In European technical publications, they are sometimes rendered as upright characters. Furthermore the *D* and *d* start a differential expression and should have appropriate spacing for differentials. The linear format treats these symbols as operand characters, but the display routines should provide the appropriate glyphs and spacings. See Sec. 3.4 for an example of an integral using $\mathbb{d}$.

3.12 Unicode Subscripts and Superscripts

Unicode contains a small set of mostly numeric superscripts (U+00B2, U+00B3, U+00B9, U+2070—U+207F) and a similar set of subscripts (U+2080—U+208F) that should be rendered the same way that scripts of the corresponding script nesting level would be rendered. To perform this translation, these characters can be treated as high-precedence operators, spans of which combine into the corresponding superscripts or subscripts when built up. Since numeric subscripts and superscripts are very common in mathematics, it's worthwhile translating between standard built-up scripts in built-up format and the Unicode scripts in linear format.

The prime U+2032 and related multiple prime characters should also be treated as superscript operators. Display routines should use an appropriate glyph variant to render the superscripted prime. The ASCII apostrophe can be used to input the prime. When it follows a variable, e.g., a', it should be converted into a superscript function with *a* as the base and the prime as the superscript. It's also important to merge the prime into a superscript that follows, e.g., a'^c should display as a'^{c}, where both the prime and the *c* are in the same superscript argument.

3.13 Concatenation Operators

All remaining operators are "concatenation operators" so named because they are concatenated with their surrounding text in built-up form. In addition a concatenation operator has two effects: 1) it terminates whatever operand precedes it, and 2) it implies appropriate surrounding space as discussed in Sec. 3.16 along with the mathematical spacing tables of the font. Since the spacing around operators is well-defined in this way, the user rarely needs to add explicit space characters.

3.14 Comma, Period, and Colon

The comma, period, and colon have context sensitive spacing requirements that can be represented in the linear format.

> **Comma**: when surrounded by ASCII digits render with ordinary text spacing. Else treat as punctuation with or without an ASCII blank following it. In either punctuation case the comma is displayed with a small space following it. If two spaces follow, the comma is rendered as a clause separator (a relatively large space follows the comma).

Period: when surrounded by ASCII digits render with ordinary text spacing. Else treat as punctuation with or without an ASCII blank following it. In either punctuation case the period is displayed with a small space following it. No clause separator option exists for the period. An extended decimal-point heuristic useful in calculator scenarios allows one to omit a leading 0, e.g., use numbers like .5. For this if the period is followed by an ASCII digit and 1) is at the start of a math zone, 2) follows a built-up math object start character or end-of-argument character, or 3) follows any operator except for closers and punctuation, then the period should be classified as a decimal point. With this algorithm, a/.3 displays as

$$\frac{a}{.3}$$

Colon: <space> ':' is displayed as Unicode RATIO U+2236 with relational spacing. ':' without a leading space is displayed as itself with punctuation spacing.

3.15 Ordinary Text Inside Math Zones

Sometimes one wants ordinary text inside a function argument or in a math zone as in the formula

$$\text{rate} = \frac{\text{distance}}{\text{time}}.$$

For such cases, the alphabetic characters should not be converted to math alphabetic characters and the typography should be that of ordinary text, not math text. To embed such text inside functions or in general in a math zone, the text can be enclosed inside ASCII double quotes. So the formula above would read in linear format as

"rate"="distance"/"time".

If you want to include a double quote inside such text, insert \". Another example is $\sin\theta = ½e^{i\theta}$ + c.c. To get the "c.c." as ordinary text, enclose it with ASCII double quotes. Otherwise the c's will be italicized and the periods will have some space after them.

Alternatively ordinary text inside a math zone can be specified using a character-format property. This property is exported to plain text started and ended with the ASCII double quote. Note that no math object or math text can be nested inside an ordinary text region. Instead if you paste a math object or text into an ordinary text region, you split the region into two such regions with the math object and/or text in between.

3.16 Space Characters

Unicode contains numerous space characters with various widths and properties. These characters can be useful in tweaking the spacing in mathematical expres-

sions. Unlike the ASCII space, which is removed when causing build up as discussed in Sec. 2.3, the other spaces are not removed on build up. Spaces of interest include the no-break space (U+00A0) and the spaces U+2000—U+200B, 202F, 205F.

In mathematical typography, the widths of spaces are usually given in integral multiples of an eighteenth of an em. The em space is given by U+2003. Various space widths are defined in the following table, which includes the corresponding MathML names having these widths by default

Space	Unicode	MathML name	Autocorrect
0 em	U+200B	zero-width space	\zwsp
1/18 em	U+200A	veryverythinmathspace	\hairsp
2/18 em	U+200A U+200A	verythinmathspace	
3/18 em	U+2009	thinmathspace	\thinsp
4/18 em	U+205F	mediummathspace	\medsp
5/18 em	U+2005	thickmathspace	\thicksp
6/18 em	U+2004	verythickmathspace	\vthicksp
7/18 em	U+2004 U+200A	veryverythickmathspace	
9/18 em	U+2002	ensp	\ensp
18/18 em	U+2003	emsp	\emsp
digit width	U+2007	numsp	\numsp
space width	U+00A0	no-break space	\nbsp

In general, spaces act as concatenation operators and cause build up of higher-precedence operators that precede them. But it's useful for the zero-width space (U+200B) to be treated as an operand character and not to cause build up of the preceding operator. The no-break space (U+00A0) is used when two words need to be separated by a blank, but remain on the same line together. The no-break space is also treated as an operand character so that linear format combinations like "lim sup" and "lim inf" can be recognized as single operands. If an ASCII space (U+0020) were used after the "lim", it would imply build up of the "lim" function, rather than being part of the "lim sup" or "lim inf" function.

In math zones, most spacing is automatically implied by the properties of the characters. The following table shows examples of how many 1/18^{ths} of an em size are automatically inserted between a character with the row property followed by a character with the column property for text-level expressions (see also p. 170 of *The TeXbook* and Appendix F of the MathML 2.0 specification)

	ord	unary	binary	rel	open	close	punct
ord	0	0	4	5	0	0	0
unary	0	0	4	0	0	0	0
binary	4	4	0	0	4	0	0
rel	5	5	0	0	5	0	0

open	0	0	0	0	0	0	0
close	0	0	4	5	0	0	0
punct	3	3	0	3	3	3	3

For the combinations described by this simple table, all script-level spacings are 0, but a more complete table would have some nonzero values. For example, in the expression $a + b$, the letters a and b have the ord (ordinary) property, while the + has the binary property in this context. Accordingly for the text level there is $4/18^{th}$ em between the a and the + and between the + and the b. Similarly there is $5/18^{th}$ em between the = and the surrounding letters in the equation $a = b$. A more complete table could include properties like math functions (trigonometric functions, etc.), *n*-ary operators, tall delimiters, differentials, subformulas (e.g., expression with an over brace), binary with no spacing (e.g., /), clause separators, ellipsis, factorial, and invisible function apply.

The zero-width space (U+200B, \zwsp) is handy for use as a null argument. For example, the expression $\mathcal{V}_{ab}$ shows the subscript ab automatically kerned in under the overhang of the $\mathcal{V}$. To prevent this kerning, one can insert a \zwsp before the subscript, which then displays unkerned as $\mathcal{V}_{ab}$.

3.17 Phantoms and Smashes

Sometimes one wants to obtain horizontal and/or vertical spacings that differ from the normal values. In [La]TeX this can be accomplished using phantoms to introduce extra space or smashes to zero out space. In the linear format, seven special cases are defined as in the following table

Autocorrect	LF op	Op name	width	ascent	descent	ink
\phantom	✧ U+27E1	white concave-sided diamond	*w*	*a*	*d*	no
\hphantom	⇔ U+2B04	white left-right arrow	*w*	0	0	no
\vphantom	⇳ U+21F3	white up-down arrow	0	*a*	*d*	no
\smash	⬍ U+2B0D	black up-down arrow	*w*	0	0	yes
\asmash	⬆ U+2B06	black up arrow	*w*	0	*d*	yes
\dsmash	⬇ U+2B07	black down arrow	*w*	*a*	0	yes
\hsmash	⬌ U+2B0C	black left-right arrow	0	*a*	*d*	yes

The general case is given by \phantom(*n*&<operand>), where *n* is any combination of the following flags:

fPhantomShow	1
fPhantomZeroWidth	2
fPhantomZeroAscent	4
fPhantomZeroDescent	8
fPhantomTransparent	16

For example, in the following equation the π in the upper limit is inside an \hsmash phantom, so that it has no width and thereby pulls the integrand in toward the integral

$$\frac{1}{2\pi}\int_0^{2\pi}\frac{d\theta}{a+b\sin\theta}=\frac{1}{\sqrt{a^2-b^2}}$$

3.18 Arbitrary Groupings

The left/right white lenticular brackets 〖 and 〗 (U+3016 and U+3017) can be used to delimit an arbitrary expression without displaying these brackets on build up. The elimination of outermost parentheses for arguments of fractions, subscripts, and superscripts solves such grouping problems nicely in most cases, but the white lenticular brackets can handle any remaining cases. Note that in math zones, these brackets should be displayed using a math font rather than an East Asian font.

3.19 Equation Arrays

To align one equation relative to another vertically, one can use an equation array, such as

$$\begin{aligned} 10x+\ \ 3y&=2\\ 3x+13y&=4 \end{aligned}$$

which has the linear format ▉(10&*x*+&3&*y*=2@3&*x*+&13&*y*=4), where ▉ is U+2588. Here the meaning of the ampersands alternate between *align* and *spacer*, with an implied *spacer* at the start of the line. So every odd & is an alignment point and every even & is a place where space may be added to align the equations. This convention is used in AmSTeX.

3.20 Math Zones

Section 5 discusses heuristic methods to identify the start and end of math zones in plain text. While the approaches given are surprisingly successful, they are not infallible. Hence if one knows the start and end of math zones, it's desirable to preserve this information in the linear format.

In plain text, the linear format uses ⁅ (U+2045) to start a math zone and ⁆ (U+2046) to end it. These are not ordinarily be used in technical documents, so these characters would rarely need to be quoted (preceded by a backslash). They are analogous to TeX's $.

When importing plain text, the user can execute a command to build up math zones defined by these math-zone delimiters. Note that although there's no way to specify display versus inline modes (TeX's $ versus $$), a useful convention for systems that mark math zones is that a (hard or soft) paragraph consisting of a math zone is in display mode. If any part of the paragraph isn't in a math zone including a possible terminating period, then inline rendering is used.

3.21 Equation Numbers

Equation numbers are often used with equations presented in display mode. To represent an equation number flushed right of the equation in the linear format, enter the equation followed by a # (U+0023) followed by the desired equation number text. For example ▉(E=mc^2#(30)) or more simply just E=mc^2#(30) renders as

$$E = mc^2 \tag{30}$$

3.22 Linear Format Characters and Operands

The linear format divides the roughly 100,000 assigned Unicode characters into three categories: 1) operand characters such as alphanumerics, 2) the bracket characters described in Sec. 3.1, and 3) other operator characters such as those described in Secs. 2.1—2.2 and 3.2—3.19. Operand characters include some nonalphanumeric characters, such as infinity (∞), exclamation point (!) if preceded by an operand, Unicode minus (U+2212) or plus if either starts a sub/superscript operand, and period and comma if they're surrounded by ASCII (or full-width ASCII) digits (Sec. 3.14 gives a generalization of this last case). In other contexts, period and comma are treated as operators with the same precedence as plus. To reveal which characters are operators, operator-aware editors could be instructed to display operators with a different color or some other attribute.

In addition, operands include bracketed expressions and mixtures of such expressions and other operand characters. Hence $f(x)$ can be an operand. More specific definitions of operands are given in the linear-format syntax of Appendix A.

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 (U+0020) is an important operator in the plain-text encoding of mathematics since it can be used to terminate operands as discussed in Sec. 2-3. A small but common list of operators is given in Table 3.1

Table 3.1 List of the most common operators ordered by increasing precedence

CR
([{ ⌊ ⊢ ⟦
)] } ⌋ ⊣ ⟧
& ⎪
Space " . , = − + * × · • ● ▒
/ ¦
∫ ∑ ∏

_ ^
□ ▭ ■ √ ∛ ∜ ‾ ⁻
Combining marks

where CR = U+000D. Note that the ASCII vertical bar | (U+007C) shows up both as an opening bracket and as a closing bracket. The choice is disambiguated by the evenness of its count at any given bracket nesting level or other considerations (see Sec. 3.1). So typically the first appearance is considered to be an open |, the next a close |, the next an open |, and so forth. The vertical bar appearing on the same level as & is considered to be a vertical bar separator and is given by the box drawings light vertical character (U+2502). We tried using the ASCII U+007C for this too, but the resulting ambiguities were insurmountable except in simple cases like (a|b) (see Section 3.1).

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

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

Precedence can be overruled using parentheses, so (α + β)/γ gives the latter.

The following gives a list of the syntax for a variety of mathematical constructs (see Appendix A for a more complete grammar).

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).
exp$_1$¦*exp*$_2$	Similar to fraction, but no fraction bar is displayed. Sometimes called a stack.
base^*exp*$_1$	Superscript expression *exp*$_1$ to the base *base*. The superscripts $^{0-9+-()}$ exist as Unicode symbols. Sub/superscript expressions are terminated, for example, by /*]) and blank. Sub/superscript operators associate right to left.
*base*_*exp*$_1$	Subscript expression *exp*$_1$ to the base *base*. The subscripts $_{0-9+-()}$ exist as Unicode symbols.
*base*_*exp*$_1$^*exp*$_2$	Subscript expression *exp*$_1$ and superscript expression *exp*$_2$ to the base *base*. The subscripts $_{0-9+-()}$ exist as Unicode symbols.
(_*exp*$_1$^*exp*$_2$)*base*	Prescript the subscript *exp*$_1$ and superscript *exp*$_2$ to the base *base*.

base┴*exp*$_1$	Display expression *exp*$_1$ centered above the base *base*. Above/below script operators associate right to left.
base┬*exp*$_1$	Display expression *exp*$_1$ centered below the base *base*.
[*exp*$_1$]	Surround *exp*$_1$ with built-up brackets. Similarly for { } and ( ). Similarly for { }, ( ), | |. See Sec. 3.1 for generalizations.
[*exp*$_1$]^*exp*$_2$	Surround *exp*$_1$ with built-up brackets followed by super-scripted *exp*$_2$ (moved up high enough).
□*exp*$_1$	Abstract box around *exp*$_1$.
▭*exp*$_1$	Rectangle around *exp*$_1$.
▁*exp*$_1$	Underbar under *exp*$_1$ (underbar operator is U+2581, not the ASCII underline character U+005F).
‾*exp*$_1$	Overbar above *exp*$_1$.
√‾*exp*$_1$	Square root of *exp*$_1$.
∛*exp*$_1$	Cube root of *exp*$_1$.
∜*exp*$_1$	Fourth root of *exp*$_1$.
√‾(*exp*$_1$&*exp*$_2$)	*exp*$_1^{\text{th}}$ root of *exp*$_2$.
Σ_*exp*$_1$^*exp*$_2$▒*exp*$_3$	Summation from *exp*$_1$ to *exp*$_2$ with summand *exp*$_3$. _*exp*$_1$ and ^*exp*$_2$ are optional.
∏_*exp*$_1$^*exp*$_2$▒*exp*$_3$	Product from *exp*$_1$ to *exp*$_2$ with multiplicand *exp*$_3$. _*exp*$_1$ and ^*exp*$_2$ are optional.
∫ _*exp*$_1$^*exp*$_2$▒*exp*$_3$	Integral from *exp*$_1$ to *exp*$_2$ with integrand *exp*$_3$. _*exp*$_1$ and ^*exp*$_2$ are optional.
(*exp*$_1$ [& *exp*$_2$]… [@ … *exp*$_{n-1}$ [& *exp*$_n$]…])	Align *exp*$_1$ over *exp* $_{n-1}$, etc., to build up an array (see Appendix A for complete syntax).

Note that Unicode's plethora of mathematical operators[2] fill out the capabilities of the approach in representing mathematical expressions in the linear format.

Precedence simplifies the text representing formulas, but may need to be overruled. To terminate an operand (shown above as, for example, *exp*$_1$) that would otherwise combine with the following operand, insert a blank (U+0020). This blank does not show up when the expression is built up. Blanks that don't terminate operands may be used to space formulas in addition to the built-in spacing provided by a math display engine. Blanks are discussed in greater detail in Sec. 2-3.

To form a compound operand, parentheses can be used as described for the fraction above. For such operands, the outermost parentheses are removed. These

operands occur for fraction numerators and denominators, subscript and superscript expressions, and arguments of functions like square root. Parentheses appearing in other contexts are always displayed in built-up format.

A curious aspect of the notation is that implied multiplication by juxtaposing two variable letters has very high precedence (just below that of diacritics), while explicit multiplication by asterisk and raised dot has a precedence equal to that of plus. So even though the analysis is similar to that for arithmetic expressions, it differs occasionally from the latter.

3.23 Equation Breaking and Alignment

Version 3 of the linear format has two features aiding equation breaking and alignment in display math zones. A soft (optional) line break is created by the invisible times (U+2062), which is a binary operator and you can break on it and align to it. It shouldn't display a glyph, except for a thin space if at the end of a math zone. With it you can effectively break an equation before any character, not just on binary, relational and some other operators. Generally it's nice to display a multiplication times symbol × if it ends up being the best point for an automatic break. This is analogous to the way the soft hyphen (U+00AD) is used in ordinary text. The zero-width space is also a binary operator and as such can be used to break equations automatically.

Interequation alignment can be accomplished by inserting &'s in front of the operators, one per equation and not inside math objects, to be aligned at the same horizontal position. For example, the lines

```
a&=b+c
x+y&=3
```

build up as

$$\begin{aligned} a &= b + c \\ x + y &= 3 \end{aligned}$$

See also Sec. 3.19 on the equation array for similar functionality.

3.24 Size Overrides

Version 3 of the linear format has a command to override the default character sizing. The inverted F character Ⅎ (U+2132) followed by various ASCII characters changes the "font" of the text. For example, a_ⅎA2 builds up as a_{2} in contrast to a_2, which builds up as a_2. The subscript 2 is larger than normal in the former. Some of the Ⅎ codes are defined in the table

ⅎA	One size larger
ⅎB	Two sizes larger
ⅎC	One size smaller
ⅎD	Two sizes smaller

These values are handy for roundtripping increase/decrease argument size context-menu options.

4. Input Methods

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. To handle these cases and to provide convenient entry of many other symbols, one can use an escape character, the backslash (\), followed by the desired operator or its autocorrect name. Note that a particularly valuable use of the nearly plain-text math format in general is for inputting formulas into technical documents or programs. In contrast, the direct input of tagged formats like MathML is very cumbersome.

4.1 Character Translations

From syntax and typographical points of view, the Unicode minus sign (U+2212) is displayed instead of the ASCII hyphen-minus (U+002D) and the prime (U+2032) is used instead of the ASCII apostrophe (U+0027), but in math zones the minus sign and prime can be entered using these ASCII counterparts. Note that for proper typography, the prime should have a large glyph variant that when superscripted looks correct. The primes in most fonts are chosen to look approximately like a superscript, but they don't provide the desired size and placement to merge well with other superscripts.

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. A more elegant approach in math zones is to translate letters deemed to be standalone to the appropriate math alphabetic characters (in the range U+1D400–U+1D7FF or in the Letterlike Block U+2100—U+213F). Letter combinations corresponding to standard function names like "sin" and "tan" should be represented by ASCII alphabetics. As such they are not italicized and are rendered with normal typography, i.e., not mathematical typography. Other post-entry enhancements include mappings like

!!	‼	U+203C
+-	±	U+00B1
-+	∓	U+2213
::	∷	U+2237
:=	≔	U+2254
<=	≤	U+2264
>=	≥	U+2265
<<	≪	U+226A
>>	≫	U+226B

~=	≅	U+2245
->	→	U+2192

The pair <- shouldn't map into ←, since expressions like $x < -b$ are common. Also it's not a good idea to map != into ≠, since ! is often used in mathematics to mean factorial.

In Version 3, negated counterparts to common mathematical operators can be entered by typing a / in front of the operator by. Operators with this behavior include those in the following table

Operator	Negated op	Input
<	≮	/<
=	≠	/=
>	≯	/>
∃	∄	/\exists
∈	∉	/\in
∋	∌	/\ni
~	≁	/\sim
≃	≄	/\simeq
≅	≇	/\cong
≈	≉	/\approx
≍	≭	/\asymp
≡	≢	/\equiv
≤	≰	/\le
≥	≱	/\ge
≶	≸	/\lessgtr
≷	≹	/\gtrless
≽	⋡	/\succeq
≺	⊀	/\prec
≻	⊁	/\succ
≼	⋠	/\preceq
⊂	⊄	/\subset
⊃	⊅	/\supset
⊆	⊈	/\subseteq
⊇	⊉	/\supseteq
⊑	⋢	/\sqsubseteq
⊒	⋣	/\sqsupseteq

All of these characters are in the U+22xx Unicode block (Mathematical Operators) except for the ASCII characters <, =, and >.

If you don't like an automatic translation when entering math, you can undo the translation by typing, for example, Ctrl+z. Suffice it to say that intelligent input algorithms can dramatically simplify the entry of mathematical symbols and expressions.

4.2 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 example, the left Alt key could 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, that is, several orders of magnitude more than Unicode can handle!

4.3 Hexadecimal Input

A handy hex-to-Unicode entry method can be used 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 is 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. Toggling back to the hex code is very useful for figuring out what a character is if the glyph itself doesn't make it clear or for looking up the character properties in the Unicode Standard. If the hex code is preceded by one or more hexadecimal digits, select the desired 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.

4.4 Pull-Down Menus, Toolbars, Context Menus

Pull-down menus and toolbars are popular methods for handling large character sets, but they tend to be slower than keyboard approaches if you know the right keys to type. A related 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. Multiple tabs can organize the symbol selections according to subject matter. On-screen keyboards and symbol boxes are valuable for entry of mathematical expressions and of Unicode text in general. Context menus (right-mouse menus) are quite useful since they provide easy access to context-sensitive options, such as converting a stacked fraction into a linear fraction.

4.5 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 and is particularly attractive to those with some familiarity with TeX.

4.6 Linear Format Math Autocorrect List

The linear format math autocorrect list includes most of those defined in Appendix F of *The TeXbook*, like \alpha for α, plus a number of others useful for inputting the linear format as shown in the following table

Control word	Character	Control word	Character
\int	∫ (U+222B)	\oint	∮ (U+222E)
\sum	∑ (U+2211)	\prod	∏ (U+220F)
\funcapply	(U+2061)	\naryand, \of	▒ (U+2592)
\rect	▭ (U+25AD)	\sqrt	√ (U+221A)
\open	├ (U+251C)	\close	┤ (U+2524)
\above	┴ (U+2534)	\below	┬ (U+252C)
\underbar	⎯ (U+2581)	\overbar	¯ (U+00AF)
\underbrace	⏟(U+23DF)	\overbrace	⏞(U+23DE)
\begin	〖 (U+3016)	\end	〗 (U+3017)
\phantom	✧(U+27E1)	\box	□ (U+25A1)
\hphantom	⇔(U+2B04)	\vphantom	⇳(U+21F3)
\asmash	⬆(U+2B06)	\dsmash	⬇(U+2B07)
\hsmash	↔(U+2B0C)	\smash	↕(U+2B0D)
\matrix	■ (U+25A0)	\eqarray	▉ (U+2588)

Appendix B contains a default set of keywords containing both *The TeXbook* keywords and the linear-format keywords

Users can define their own control words for convenience or preference, such as \a for *α*, which requires less typing than the official TeX control word \alpha. This also allows localization of the control word list.

4.7 Handwritten Input

Particularly for PDAs and Tablet PCs, handwritten input is attractive provided the handwriting recognizer is able to decipher the user's handwriting. For this approach, it's desirable to bypass the linear format altogether and recognize built-up mathematical expressions directly.

5. Recognizing Mathematical Expressions

Plain-text linearly formatted 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.

Note that if explicit math-zone-on and math-zone-off characters are desired, Sec. 3.20 specifies that ⁅ (U+2045) starts a math zone and ⁆ (U+2046) ends it. These are not ordinarily be used in technical documents. If they do need to be included in a math zone, they can be preceded by the "quote" character \ as described in Sec. 3.2.

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 many error messages because TeX interprets subsequent text in the wrong mode.

An advantage of recognizing mathematical expressions without math-on and 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 could 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 whose state is revealed to the user by a toolbar button. 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), the math alphanumerics (U+1D400 – U+1D7FF), and in general, Unicode characters with the mathematics property, identify the characters immediately surrounding them as parts of math expressions.

If Latin 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. 6), 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 operator that follows it to be treated as an nonbuildup operator. 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 or by an ordinary text attribute if they need to be embedded in the math zone, e.g., in the numerator of a fraction.

6. Using the Linear Format 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[7] "a good notation has a subtlety and suggestiveness which at times make it seem almost like a live teacher…and a perfect notation would be a substitute for thought." From this point of view, popular modern computer languages are badly lacking. At least Java allows many Unicode characters as variable names and Fortress goes further, resembling mathematics more closely.

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's worth reexamining this premise. Whereas true mathematical notation clearly used to be beyond the capabilities of machine recognition, we're getting a lot closer now.

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 linear format.

6.1 Advantages of Linear Format in Programs

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 Maple, Mathematica, and Macsyma. The translation between Unicode symbols and the ASCII names needed by ASCII-based compilers and symbolic manipulation programs can be 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 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 plain-text linear format 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 hand written input into machine usable form.

The concept is portable to any environment that supports 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 ith 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 un-

enlightened ASCII-only compilers can be done via an ASCII representation of Unicode characters. The Fortress language adopts Unicode much more seriously, taking considerable advantage of Unicode's large mathematical operator repertoire.

6.2 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;
        α_inc = α₀ • (1 − (γ • γ • I₂/γ')/(γ' + υ));
        if (! γ₁ || fabs(Δ • T₁) < 0.01)
                α_coh = −.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;
                α_coh  = .5 • γ • α₀ • (I₂ℱ(γ + υ)/(γ' • γ' − β²))
                        × ((1 + γ/β) • (β − υ)/(β + υ) − (1 + γ/γ') • (γ' − υ)/(γ' + υ));
        }
        α₁ = α_inc  + α_coh;
}
```

The above function runs fine with 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;
        α_inc = α₀ • (1 − (γ • γ • I₂/γ')/(γ' + υ));
        if (! γ₁|| fabs(Δ • T₁) < 0.01)
                α_coh = −.5 • α₀ • I₂ • (γ/γ')³;
        else
        {
                Γ = 1/T₁ + γ₁;
                I₂ℱ = I₂/T₁/(Γ + i • Δ);
                β² = υ • (υ + γ • I₂ℱ);
                β = √β²;
                α_coh = .5 • γ • α₀ • I₂ℱ(γ + υ)/(γ' • γ' − β²) • ((1 + γ/β) • (β − υ)/(β + υ) − (1 + γ/γ') • (γ' − υ)/(γ' + υ));
        }
```

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

The ability to use the second and third versions of the function was built into the PS Technical Word Processor[8] 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 could be previewed in built-up format, complete with fraction bars, square roots, and large parentheses. To code a formula, one copies it from a technical document, pastes it into a program file, inserts appropriate raised dots for multiplication and compiles. 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[9]].

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 limited (although usually 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.

6.3 Export to TeX

Export to TeX is similar to export 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 linear format, 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 needn't get in the way.

With TeX, the text surrounding the mathematics is part and parcel of the technical document, and TeX needs $'s to distinguish the two. These can be included in

the plain text, but it is somewhat ugly. The heuristics described in Sec. 5 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 as for the program translations. Alternatively one can use LaTeX's \[...\] open/close approach.

Export to MathML also requires knowing the start and end of a math zone. The built-up functions can all be represented using MathML elements or combinations of elements. The most glaring omission in Presentation MathML is that there's no "*n*-ary" element: one needs to use one of a variety of other elements like <msub> along with the desired *n*-ary operator inside an <mo>. In addition one needs to tag numbers, operators, and identifiers.

7. Conclusions

We have shown how with a few additions to Unicode, mathematical expressions can usually be represented with a readable Unicode nearly plain-text format, which we call the linear format. The text consists of combinations of operators and operands. A simple operand consists of a span of non-operators, a definition that substantially reduces the number of parenthesis-override pairs and thereby increases the readability of the plain text. 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 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 particularly for input purposes, symbolic manipulation, and numerical computation.

A variety of syntax choices could be used for a linear format. The choices made in this paper favor efficient input of mathematical formulae, sufficient generality to support high-quality mathematical typography, the ability to round trip elegant mathematical text at least in a rich-text environment, and a format that resembles a real mathematical notation. Obviously compromises between these goals had to be made.

The heuristics given for recognizing mathematical expressions work well, but they are not infallible. An effective use of the heuristics would be by an autoformatting wizard that delimits what it thinks are math zones with on/off codes or a character-format attribute. The user could then overrule any incorrect choices. Once the math zones are identified unequivocally, export to MathML, compilers, and other consumers of mathematical expressions is straightforward.

Acknowledgements

This work has benefitted from discussions with many people, notably PS Technical Word Processor users, Asmus Freytag, Barbara Beeton, Ken Whistler, Donald

Knuth, Jennifer Michelstein, Ethan Bernstein, Said Abou-Hallawa, Jason Rajtar, Yi Zhang, Geraldine Wade, Ross Mills, John Hudson, Ron Whitney, Richard Lawrence, Sergey Malkin, Alex Gil, Mikhail Baranovsky, Hon-Wah Chan, José Oglesby, Isao Yamauchi, Yuriko Rosnow, Robert Miller, Joe Roni, Jinsong Yu, Sergey Genkin, Victor Kozyrev, Andrei Burago, and Eliyezer Kohen. Earlier related work is listed in Ref. 10.

Appendix A. Linear Format Grammar

This grammar is simplified compared to the model in the text.

char	←	Unicode character
space	←	ASCII space (U+0020)
αASCII	←	ASCII A-Z a-z
nASCII	←	ASCII 0-9
αnMath	←	Unicode math alphanumeric (U+1D400 – U+1D7FF with some Letterlike symbols U+2102 – U+2134)
αnOther	←	Unicode alphanumeric not including *αnMath* nor *nASCII*
αn	←	*αnMath* | *αnOther*
diacritic	←	Unicode combining mark
opArray	←	'&' | VT | '■'
opClose	←	')' | ']' | '}' | '⟩'
opCloser	←	*opClose* | "\close"
opDecimal	←	'.' | ','
opHbracket	←	Unicode math horizontal bracket
opNary	←	Unicode integrals, summation, product, and other nary ops
opOpen	←	'(' | '[' | '{' | '⟨'
opOpener	←	*opOpen* | "\open"
opOver	←	'/' | "\atop"
opBuildup	←	'_' | '^' | '√' | '∛' | '∜' | '□' | '/' | '|' | *opArray* | *opOpen* | *opClose* | *opNary* | *opOver* | *opHbracket* | *opDecimal*
other	←	*char* – {*αn* + *nASCII* + *diacritic* + *opBuildup* + CR}
diacriticbase	←	*αn* | *nASCII* | '(' *exp* ')'
diacritics	←	*diacritic* | *diacritics diacritic*
atom	←	*αn* | *diacriticbase diacritics*
atoms	←	*atom* | *atoms atom*
digits	←	*nASCII* | *digits nASCII*
number	←	*digits* | *digits opDecimal digits*
expBracket	←	*opOpener exp opCloser*
	←	'||' *exp* '||'
	←	'|' *exp* '|'

word ← *αASCII* | *word αASCII*
scriptbase ← *word* | *word nASCII* | *αnMath* | *number* | *other* | *expBracket* | *opNary*
soperand ← *operand* | '∞' | '-' *operand* | "-∞"
expSubsup ← *scriptbase* '_' *soperand* '^' *soperand* | *scriptbase* '^' *soperand* '_' *soperand*
expSubscript ← *scriptbase* '_' *soperand*
expSuperscript ← *scriptbase* '^' *soperand*
expScript ← *expSubsup* | *expSubscript* | *expSuperscript*

entity ← *atoms* | *expBracket* | *number*
factor ← *entity* | *entity* '!' | *entity* "!!" | *function* | *expScript*
operand ← *factor* | *operand factor*
box ← '□' *operand*
hbrack ← *opHbracket operand*
sqrt ← '√' *operand*
cubert ← '∛' *operand*
fourthrt ← '∜' *operand*
nthrt ← "√(" *operand* '&' *operand* ')'
function ← *sqrt* | *cubert* | *fourthrt* | *nthrt* | *box* | *hbrack*
numerator ← *operand* | *fraction*
fraction ← *numerator opOver operand*

row ← *exp* | *row* '&' *exp*
rows ← *row* | *rows* '@' *row*
array ← "\array(" *rows* ')'

element ← *fraction* | *operand* | *array*
exp ← *element* | *exp other element*

Appendix B. Character Keywords and Properties

The following table gives the default math keywords, their target characters and codes along with spacing and linear-format build-up properties. A full keyword consists of a backslash followed by a keyword in the table.

Keyword	Glyph	Code	Spacing	LF Property
\above	⊥	U+2534	ordinary	subsup upper
\acute	´	U+0301	ordinary	accent
\aleph	ℵ	U+2135	ordinary	operand
\alpha	α	U+03B1	ordinary	operand

\amalg	⨿	U+2210	ordinary	nary
\angle	∠	U+2220	relational	normal
\aoint	∳	U+2233	ordinary	nary
\approx	≈	U+2248	relational	normal
\asmash	⬆	U+2B06	ordinary	encl phantom
\ast	∗	U+2217	binary	normal
\asymp	≍	U+224D	relational	normal
\atop	¦	U+00A6	ordinary	divide
\Bar	‾	U+033F	ordinary	accent
\bar	¯	U+0305	ordinary	accent
\because	∵	U+2235	relational	normal
\begin	⟦	U+3016	open	open
\below	⊤	U+252C	ordinary	subsup lower
\beta	β	U+03B2	ordinary	operand
\beth	ב	U+2136	ordinary	operand
\bot	⊥	U+22A5	relational	normal
\bigcap	⋂	U+22C2	ordinary	nary
\bigcup	⋃	U+22C2	ordinary	nary
\bigodot	⊙	U+2A00	ordinary	nary
\bigoplus	⊕	U+2A01	ordinary	nary
\bigotimes	⊗	U+2A02	ordinary	nary
\bigsqcup	⊔	U+2A06	ordinary	nary
\biguplus	⊎	U+2A04	ordinary	nary
\bigvee	⋁	U+22C1	ordinary	nary
\bigwedge	⋀	U+22C0	ordinary	nary
\bowtie	⋈	U+22C8	relational	normal
\bot	⊥	U+22A5	relational	normal
\box	□	U+25A1	ordinary	encl box
\bra	⟨	U+27E8	open	open
\breve	˘	U+0306	ordinary	accent
\bullet	∙	U+2219	binary	normal
\cap	∩	U+2229	binary	normal
\cbrt	∛	U+221B	open	encl root
\cdot	⋅	U+22C5	binary	normal
\cdots	⋯	U+22EF	ordinary	normal

\check	ˇ	U+030C	ordinary	accent
\chi	χ	U+03C7	ordinary	operand
\circ	∘	U+2218	binary	normal
\close	┤	U+2524	ordinary	close
\clubsuit	♣	U+2663	ordinary	normal
\coint	∲	U+2232	ordinary	nary
\cong	≅	U+2245	relational	normal
\cup	∪	U+222A	binary	normal
\daleth	ℸ	U+2138	ordinary	operand
\dashv	⊣	U+22A3	relational	stretch horz
\Dd	ⅅ	U+2145	differential	operand
\dd	ⅆ	U+2146	differential	operand
\ddddot	⃜	U+20DC	ordinary	accent
\dddot	⃛	U+20DB	ordinary	accent
\ddot	¨	U+0308	ordinary	accent
\ddots	⋱	U+22F1	relational	normal
\degree	°	U+00B0	ordinary	operand
\Delta	Δ	U+0394	ordinary	operand
\delta	δ	U+03B4	ordinary	operand
\diamond	⋄	U+22C4	binary	normal
\diamondsuit	♢	U+2662	ordinary	normal
\div	÷	U+00F7	binary	normal
\dot	˙	U+0307	ordinary	accent
\doteq	≐	U+2250	relational	normal
\dots	…	U+2026	ordinary	normal
\Downarrow	⇓	U+21D3	relational	normal
\downarrow	↓	U+2193	relational	normal
\dsmash	⬇	U+2B07	ordinary	encl phantom
\ee	ⅇ	U+2147	ordinary	operand
\ell	ℓ	U+2113	ordinary	operand
\emptyset	∅	U+2205	unary	operand
\emsp		U+2003	skip	normal
\end	〗	U+3017	close	close
\ensp		U+2002	skip	normal
\epsilon	ϵ	U+03F5	ordinary	operand

\eqarray	▉	U+2588	ordinary	encl eqarray
\eqno	#	U+0023	ordinary	marker
\equiv	≡	U+2261	relational	normal
\eta	η	U+03B7	ordinary	operand
\exists	∃	U+2203	unary	normal
\forall	∀	U+2200	unary	normal
\funcapply	⁡	U+2061	binary	subsupFA
\Gamma	Γ	U+0393	ordinary	operand
\gamma	γ	U+03B3	ordinary	operand
\ge	≥	U+2265	relational	normal
\geq	≥	U+2265	relational	normal
\gets	←	U+2190	ordinary	stretch horiz
\gg	≫	U+226B	relational	normal
\gimel	ℷ	U+2137	ordinary	operand
\grave	`	U+0300	ordinary	accent
\hairsp		U+200A	skip	normal
\hat	ˆ	U+0302	ordinary	accent
\hbar	ℏ	U+210F	ordinary	operand
\heartsuit	♡	U+2661	ordinary	normal
\hookleftarrow	↩	U+21A9	relational	stretch horiz
\hookrightarrow	↪	U+21AA	relational	stretch horiz
\hphantom	⇔	U+2B04	ordinary	encl phantom
\hsmash	⬌	U+2B0C	ordinary	encl phantom
\hvec	⃗	U+20D1	ordinary	accent
\ii	ⅈ	U+2148	ordinary	operand
\iiiint	⨌	U+2A0C	ordinary	nary
\iiint	∭	U+222D	ordinary	nary
\iint	∬	U+222C	ordinary	nary
\Im	ℑ	U+2111	ordinary	operand
\imath	ı	U+0131	ordinary	operand
\in	∈	U+2208	relational	normal
\inc	Δ	U+2206	unary	operand
\infty	∞	U+221E	ordinary	operand
\int	∫	U+222B	ordinary	nary
\iota	ι	U+03B9	ordinary	operand

\jj	ⅉ	U+2149	ordinary	operand
\jmath	ȷ	U+0237	ordinary	operand
\kappa	κ	U+03BA	ordinary	operand
\ket	⟩	U+27E9	close	close
\Lambda	Λ	U+039B	ordinary	operand
\lambda	λ	U+03BB	ordinary	operand
\langle	⟨	U+27E8	open	open
\lbrace	{	U+007B	open	open
\lbrack	[	U+005B	open	open
\lceil	⌈	U+2308	open	open
\ldiv	∕	U+2215	binary	divide
\ldots	…	U+2026	ordinary	normal
\le	≤	U+2264	relational	normal
\Leftarrow	⇐	U+21D0	relational	stretch horiz
\leftarrow	←	U+2190	relational	stretch horiz
\leftharpoondown	↽	U+21BD	relational	stretch horiz
\leftharpoonup	↼	U+21BC	relational	stretch horiz
\Leftrightarrow	⇔	U+21D4	relational	stretch horiz
\leftrightarrow	↔	U+2194	relational	stretch horiz
\leq	≤	U+2264	relational	normal
\lfloor	⌊	U+230A	open	open
\ll	≪	U+226A	relational	normal
\Longleftarrow	⟸	U+27F8	relational	normal
\longleftarrow	⟵	U+27F5	relational	normal
\Longleftrightarrow	⟺	U+27FA	relational	normal
\longleftrightarrow	⟷	U+27F7	relational	normal
\Longrightarrow	⟹	U+27F9	relational	normal
\longrightarrow	⟶	U+27F6	relational	normal
\mapsto	↦	U+21A6	relational	stretch horiz
\matrix	■	U+25A0	ordinary	encl matrix
\medsp		U+205F	Ordinary	normal
\mid	∣	U+2223	relational	list delims
\models	⊨	U+22A8	relational	stretch horz
\mp	∓	U+2213	unary/binary	unary/binary
\mu	μ	U+03BC	ordinary	operand

\nabla	∇	U+2207	unary	operand
\naryand	▒	U+2592	ordinary	normal
\nbsp		U+00A0	skip	normal
\ndiv	⊘	U+2298	binary	divide
\ne	≠	U+2260	relational	normal
\nearrow	↗	U+2197	relational	normal
\neg	¬	U+00AC	unary	normal
\neq	≠	U+2260	relational	normal
\ni	∋	U+220B	relational	normal
\norm	‖	U+2016	ordinary	open/close
\nu	ν	U+03BD	ordinary	operand
\nwarrow	↖	U+2196	relational	normal
\odot	⊙	U+2299	binary	normal
\of	▒	U+2592	ordinary	normal
\oiiint	∰	U+2230	ordinary	nary
\oiint	∯	U+222F	ordinary	nary
\oint	∮	U+222E	ordinary	nary
\Omega	Ω	U+03A9	ordinary	operand
\omega	ω	U+03C9	ordinary	operand
\ominus	⊖	U+2296	binary	normal
\open	├	U+251C	ordinary	open
\oplus	⊕	U+2295	binary	normal
\oslash	⊘	U+2298	binary	normal
\otimes	⊗	U+2297	binary	normal
\over	/	U+002F	binarynsp	divide
\overbar	¯	U+00AF	ordinary	encl overbar
\overbrace	⏞	U+23DE	ordinary	stretch over
\overparen	⏜	U+23DC	ordinary	stretch over
\parallel	∥	U+2225	relational	normal
\partial	∂	U+2202	unary	operand
\phantom	✧	U+27E1	ordinary	encl phantom
\Phi	Φ	U+03A6	ordinary	operand
\phi	ϕ	U+03D5	ordinary	operand
\Pi	Π	U+03A0	ordinary	operand
\pi	π	U+03C0	ordinary	operand

\pm	±	U+00B1	unary/binary	unary/binary
\pppprime	⁗	U+2057	ordinary	Unisubsup
\ppprime	‴	U+2034	ordinary	Unisubsup
\pprime	″	U+2033	ordinary	Unisubsup
\prec	≺	U+227A	relational	normal
\preceq	≼	U+227C	relational	normal
\prime	′	U+2032	ordinary	Unisubsup
\prod	∏	U+220F	ordinary	nary
\propto	∝	U+221D	relational	normal
\Psi	Ψ	U+03A8	ordinary	operand
\psi	ψ	U+03C8	ordinary	operand
\qdrt	∜	U+221C	open	encl root
\rangle	⟩	U+27E9	close	close
\ratio	∶	U+2236	relational	normal
\rbrace	}	U+007D	close	close
\rbrack	]	U+005D	close	close
\rceil	⌉	U+2309	close	close
\rddots	⋰	U+22F0	relational	normal
\Re	ℜ	U+211C	ordinary	operand
\rect	▭	U+25AD	ordinary	encl rect
\rfloor	⌋	U+230B	close	close
\rho	ρ	U+03C1	ordinary	operand
\Rightarrow	⇒	U+21D2	relational	stretch horiz
\rightarrow	→	U+2192	relational	stretch horiz
\rightharpoondown	⇁	U+21C1	relational	stretch horiz
\rightharpoonup	⇀	U+21C0	relational	stretch horiz
\sdiv	⁄	U+2044	binarynsp	divide
\searrow	↘	U+2198	relational	normal
\setminus	\	U+2216	binary	normal
\Sigma	Σ	U+03A3	ordinary	operand
\sigma	σ	U+03C3	ordinary	operand
\sim	∼	U+223C	relational	normal
\simeq	≃	U+2243	relational	normal
\smash	⬍	U+2B0D	ordinary	encl phantom
\spadesuit	♠	U+2660	ordinary	normal

\sqcap	⊓	U+2293	binary	normal
\sqcup	⊔	U+2294	binary	normal
\sqrt	√	U+221A	open	encl root
\sqsubseteq	⊑	U+2291	relational	normal
\sqsupseteq	⊒	U+2292	relational	normal
\star	⋆	U+22C6	binary	normal
\subset	⊂	U+2282	relational	normal
\subseteq	⊆	U+2286	relational	normal
\succ	≻	U+227B	relational	normal
\succeq	≽	U+227D	relational	normal
\sum	Σ	U+2211	ordinary	nary
\superset	⊃	U+2283	relational	normal
\superseteq	⊇	U+2287	relational	normal
\swarrow	↘	U+2199	relational	normal
\tau	τ	U+03C4	ordinary	operand
\therefore	∴	U+2234	relational	normal
\Theta	Θ	U+0398	ordinary	operand
\theta	θ	U+03B8	ordinary	operand
\thicksp		U+2005	skip	normal
\thinsp		U+2006	skip	normal
\tilde	˜	U+0303	ordinary	accent
\times	×	U+00D7	binarynsp	normal
\to	→	U+2192	relational	stretch horiz
\top	⊤	U+22A4	relational	normal
\tvec	↔	U+20E1	ordinary	accent
\underbar	⎯	U+2581	ordinary	encl un-
\underbrace	⏟	U+23DF	ordinary	stretch under
\underparen	⏝	U+23DD	ordinary	stretch under
\Uparrow	⇑	U+21D1	relational	normal
\uparrow	↑	U+2191	relational	normal
\Updownarrow	⇕	U+21D5	relational	normal
\updownarrow	↕	U+2195	relational	normal
\uplus	⊎	U+228E	binary	normal
\Upsilon	ϒ	U+03A5	ordinary	operand
\upsilon	υ	U+03C5	ordinary	operand

\varepsilon	ε	U+03B5	ordinary	operand
\varphi	φ	U+03C6	ordinary	operand
\varpi	ϖ	U+03D6	ordinary	operand
\varrho	ϱ	U+03F1	ordinary	operand
\varsigma	ς	U+03C2	ordinary	operand
\vartheta	ϑ	U+03D1	ordinary	operand
\vbar	|	U+2502	ordinary	list delims
\vdash	⊢	U+22A2	relational	stretch horz
\vdots	⋮	U+22EE	relational	normal
\vec	⃗	U+20D7	ordinary	accent
\vee	∨	U+2228	binary	normal
\Vert	‖	U+2016	ordinary	open/close
\vert	|	U+007C	ordinary	open/close
\vphantom	⇳	U+21F3	relational	encl phantom
\vthicksp		U+2004	skip	normal
\wedge	∧	U+2227	binary	normal
\wp	℘	U+2118	ordinary	operand
\wr	≀	U+2240	binary	normal
\Xi	Ξ	U+039E	ordinary	operand
\xi	ξ	U+03BE	ordinary	operand
\zeta	ζ	U+03B6	ordinary	operand
\zwnj		U+200C	ordinary	normal
\zwsp		U+200B	ordinary	normal

Version Differences

The differences between Version 1 and 2 of this paper are largely cosmetic, but there were enough changes in Version 2 to merit a new number. Version 2 is mostly implemented in Microsoft Word 2007, where it is referred to as the "linear format". Typing the linear format in Word 2007 results in "formula autobuildup", that is, automatic conversion to the built-up format of expressions as their syntax becomes unambiguous.

In this document features added in Version 3 are identified as such. These features are mostly implemented in the Microsoft Office 2010 applications Word, PowerPoint, Excel, and OneNote. Typically the additions offer convenience over ways needed in Version 2, but no addition is necessary and the Version 2 syntax remains valid in Version 3. The additions were often inspired by [La]TeX. Examples of simplified input are \choose for binomial coefficients, \cases for alternative definitions, \pmatrix for parenthesized matrices, \middle to define a character as a bracket separator, a simpler prescript notation, \root *n*\of *x* notation for n^{th} roots, equation alignment (see Sec. 3.23), size overrides (see Sec. 3.24), and simple negated operator input (see Sec. 4.1). There are also numerous cosmetic changes.

References

1. *The Unicode Standard*, Version 5.0, (Reading, MA, Addison-Wesley, 2006. ISBN 0-321-18578-1) or online as http://www.unicode.org/versions/Unicode5.0.0/
2. Barbara Beeton, Asmus Freytag, Murray Sargent III, Unicode Technical Report #25 "Unicode Support for Mathematics", http://www.unicode.org/reports/tr25
3. Leslie Lamport, *LaTeX: A Document Preparation System, User's Guide & Reference Manual*, 2nd edition (Addison-Wesley, 1994; ISBN 1-201-52983-1)
4. Donald E. Knuth, *The TeXbook*, (Reading, Massachusetts: Addison-Wesley 1984)
5. *Mathematical Markup Language (MathML) Version 2.0* (Second Edition) http://www.w3.org/TR/2003/REC-MathML2-20031021/ .
6. For example, the linear format is used for keyboard entry of mathematical expressions in Microsoft Word 2007 and the Microsoft Math Calculator.
7. Bertrand Russell, in his Introduction to *Tractatus Logico-Philosophicus* by Lugwig Wittgenstein, Routledge and Kegan Paul, London 1922 (also currently available at http://www.kfs.org/~jonathan/witt/tlph.html).
8. PS Technical Word Processor, Scroll Systems, Inc. (1989). This WP used a non-Unicode version of the plain-text math notation.
9. P. Meystre and M. Sargent III (1991), *Elements of Quantum Optics*, Springer-Verlag

10. Some of these ideas were discussed in the following presentations: M. Sargent III, *Unicode, Rich Text, and Mathematics*, 7th International Unicode Conference, San Jose, California, Sept (1995); Murray Sargent III and Angel L. Diaz, *MathML and Unicode,* 15th International Unicode Conference, San Jose, California, Sept (1999); Murray Sargent III, *Unicode Plain Text Encoding of Mathematics,* 16th International Unicode Conference, Amsterdam, Holland, March (2000); Murray Sargent III, *Unicode Support for Mathematics*, 17th International Unicode Conference, San Jose, California, Sept (2000); Murray Sargent III, *Unicode Support for Mathematics*, 22nd International Unicode Conference, San Jose, California, Sept (2002); Murray Sargent III, *Unicode Nearly Plain-Text Encoding of Mathematics*, 26th Internationalization and Unicode Conference, San Jose, California, Sept (2004). Murray Sargent III, *Editing and Display of Mathematics using Unicode*, 29th Internationalization and Unicode Conference, San Francisco, California, March (2006). Murray Sargent III, *Mathematical Input Methods*, 31st Internationalization and Unicode Conference, San Jose, California, Oct (2007). Murray Sargent III, *Math Editing and Display in Microsoft Office*, 33rd Internationalization and Unicode Conference, San José, California, Sept (2009).

This document was prepared using Microsoft Word 2010 with Cambria and Cambria Math fonts.