Technical Aspects of the Digital Library of Mathematical Functions †

Bruce R. Miller
National Institute of Standards and Technology; Gaithersburg, MD 20899, USA; bruce.miller@nist.gov

Abdou Youssef
Dept. of Computer Science; The George Washington University; Washington, DC 20052; youssef@seas.gwu.edu

Abstract. The NIST Digital Library of Mathematical Functions (DLMF) Project, begun in 1997, is preparing a handbook and Web site intended for wide communities of users. The contents are primarily mathematical formulas, graphs, methods of computation, references, and links to software. The task of developing a Web handbook of this nature presents several technical challenges. We describe the goals of the Digital Library of Mathematical Functions Project and the realities that constrain those goals. We propose practical initial solutions, in order to ease the authoring of adaptable content: a LaTeX class which encourages a modestly semantic markup style; and a mathematical search engine that adapts a text search engine to the task.

Keywords: Digital Library, Mathematical markup, Mathematical notation, Mathematical Search

AMS(MOS) Codes: 33-00, 68T30

1. Introduction

The NIST Digital Library of Mathematical Functions (DLMF) Project[1], begun in 1997, is preparing a handbook and Web site intended for wide communities of users. The contents are primarily mathematical formulas, graphs, methods of computation, references, and links to software. This project revises and extends Abramowitz and Stegun's Handbook of Mathematical Functions [1]. An overview of the project can be found in [11]. This paper addresses some of the technical aspects of the project. The route that we will take is to first describe the goals of the DLMF. That some of these goals conflict with reality leads to discussion of how we have adapted to those conflicts.

There are two underlying themes to this paper. The first theme is finding the most practical way of obtaining a variety of information

† Official contribution of the National Institute of Standards and Technology; not subject to copyright in the United States. Research was supported in part by NSF Grant 9980036 and in part by the SIMA, SRD and ATP Programs at NIST.

[1] `http://dlmf.nist.gov/`

from the project's authors, much of which may seem unusual to them. The second theme is how to make the information available to users through different media and via appropriate search capabilities. We must extract from the author's manuscripts a close approximation to the semantic content, especially the mathematical content, in a form that will allow the broadest, most long-term usage. It must be noted that this project is a work-in-progress; many of the most interesting problems have not yet been completely solved; and even the notion of 'best solution' will evolve with the evolution of the web, itself.

2. Goals and Challenges

2.1. Goals

That the material must be of the highest quality — *authoritative and validated* — should go without saying. We aim to produce a book, a web site with extensive search capabilities and a CD-ROM version. The material must therefore adapt to *various media*.

Clearly, the electronic formats must provide *search capabilities*. And since the DLMF is primarily mathematics with little text, providing a traditional search for the textual components will not be enough. We must provide the capability to search for formulas that match a user's criteria as well; searching for formulas according to keywords or properties of the formula: e.g. 'addition theorem for elliptic functions'. More intriguing would be searches using mathematical patterns. These should be flexible regarding syntax, not restricted to TEX[8], say. And the matching process must understand the basic properties of commutativity, associativity and so forth.

Along with search capabilities, it would be extremely useful to be able to extract *virtual documents*. For example, one might wish to create a page of addition theorems, or a short booklet of differential equations of certain classes. Thus one must not only find pages, but collections of document fragments and formulas, and synthesize a document from them. These, along with existing sections or subsections should be viewable in a screen friendly format, or printable.

The DLMF should provide *layers of detail*. Beyond the dry, telegraphic, front facade of the identities, there should be associated with most elements additional metadata. This data would point to references, original sources, or short tracts going into extra detail or more esoterica.

Often one finds a formula not quite in the terms one needs and one would like to *transform* the expressions. It would be exciting,

from within the online book, to substitute variables, or rearrange an expression, to acquire exactly the identity required. One may need to re-expand, apply transformations, other identities, ... Transformed or not, users will likely want to copy the formula into their own documents, graphics programs or computer algebra system, without loss of meaning.

For some people, a differential equation tells them all they need to know about a special function. Other, more visually oriented people, will need *interactive graphics* to explore the function in different regions, on different scales. Graphics in 2D and 3D (or more) will be needed. Above all, these graphics must be 'Honest' in the sense of Fateman [5]: they must not display artificial flat planes where clipped, nor smooth over narrow features, nor succumb to other sampling errors.

2.2. Some Successes

Some of these goals are well on their way to success. Dan Lozier describes in [11] the project and the strategy for managing it. Given the project's Editorial Board and the authors they have contracted, the quality of the mathematical content is well in hand.

Saunders and Wang [22, 18] have been making significant progress on 3D graphics, with some 'honest' features (see also [11]). We have been exploring 2D interactive honest graphics, as well.

2.3. Challenges

Other elements remain difficult, however.

What might be called 'live mathematics', direct manipulation of the formula within the browser, has been explored in teaching materials, where it is quite appropriate. But in our case, the capability to transform a given identity into an arbitrary form of the user's choosing is an open-ended task. Any non-trivial usage would require access to the full power of a computer algebra system. Although presenting limited alternative views of formulas is reasonable, providing completely live on-line manipulations of them is quite beyond the scope of the planned DLMF.

On the other hand, providing mathematical *reference data* is clearly within our role at NIST. The right representation of the mathematics will allow inclusion in user documents, or insertion into a user's computer algebra system. The problem is thus transformed into the problem of extracting a sufficiently generic, semantic representation of the mathematics, say in OpenMath [21] or Content MathML [23], from what the authors have written.

While a mathematical structure editing application, say with buttons for every conceivable mathematical concept, might be a good solution for the future, it is not an answer for us at present. The chapter authors were chosen for their mathematical expertise. They are experienced mathematicians who already have their preferred authoring tools, their favorite operating system. For the most part, they do not want to be retrained for new tools, nor do we want to retrain them — we want them to focus on the mathematical content.

In fact, most (but not all!) of these authors are comfortable with TEX or LATEX[10]. While LATEX markup is significantly more content oriented than many systems, it still allows for a great many ambiguities. It has been suggested that one should develop a LATEX package providing completely unambiguous mathematics markup. Again, although this may be a good goal for future work — and future authors — it would not be workable here for the same reasons given above; it would be too unfamiliar and too verbose, and would not get used.

Search, virtual documents and layered information require the support of a variety of metadata. Annotations are needed to provide layered information. An abundance of references and indexing keywords are called for; some would appear in the regular bibliography or index, but others used solely for layered information or for the search engine. Constructing sensible virtual documents would need some knowledge of the 'role' of sections and formulas, e.g. does it represent a definition, or notation. Unless one is willing to impose a rigid organization on all chapters, one must also include metadata to indicate this role.

These items are fairly unusual requests to make of an author, and yet the authors are the best sources. So, while we accept that we will have to augment and rationalize the metadata that they do provide, we must make it as easy as possible for them to include that information. Furthermore, since metadata is generally hidden from view, we must provide feedback to the authors, and also to the editors, for proofreading purposes of what data is present.

Even with extensive indexing metadata, users must be able to search in the mathematical content itself. Search and retrieval technology has been developing for several decades, and has reached a high level of maturity for *text* search [9, 17]. The formulas, equations, and other mathematical constructs present in the DLMF, however, are symbolic and highly structured. Current search systems and technology do not provide the means for formulating math queries and for searching for equations and other mathematical constructs. Many equivalences, such as commutativity and associativity, are simply not recognized.

Finally, there is the issue of targeting the material to different media. Even were it not for our emphasis on semantic markup, we wish to

discourage presentation markup. Not only does it obfuscate the content and waste the author's time, it tends to be specific to a certain medium, layout or dimensions.

2.4. A path to solution

Our general strategy is an evolutionary one. We have developed a DLMF LaTeX class that strays as little as possible from standard LaTeX markup, while encouraging content over presentation. This 'modestly' semantic markup is described further in Section 3.

Our initial work on the mathematical search problem adapts and augments a conventional text search engine to meet the needs of the DLMF. This involves predigesting the mathematics into a more textual form before indexing, defining and implementing a math query language to express formula queries naturally and transforming the search queries into special forms. While perhaps not ideal, the method has had remarkable success so far. The approach is described in more detail in Section 4.

These approaches will allow us to reach many of our goals in the short term, while we continue research and development on better solutions. Focusing the authors on content over presentation and implementation specific details, will allow us to evolve the handling of the material without restarting the project from scratch.

3. Modestly Semantic Markup

Given these constraints, the first technical objective is to modify LaTeX markup just enough to:

- allow the authors to focus on the mathematical content;
- minimize presentation markup, while supporting different formats and media;
- minimize mathematical ambiguities;
- support proofreading with optionally printed representations of all hidden data.

3.1. Metadata

The indexing, labeling and citation mechanisms of standard LaTeX provide a starting point for metadata. The addition of a few simple macros

for annotations and original references forms a good basis. However, as feedback to the authors and editors, as well as to other authors who wish to cross-reference, we also provide means to print this extra data without destroying the flow of the main text. An annotated format has been developed which prints the chapter in a single narrow column, with all metadata printed in the second column, aligned to the material it relates to. Alternatively, the data can be printed in an appendix.

3.2. Optional Material

For practical reasons, a printed version of the DLMF must be constrained in size to not much more than 1000 pages. Electronic versions are much less restricted, and thus may contain additional material. Also, material that is most likely to change quickly (e.g. lists of available software) should probably be relegated to the more dynamic electronic media. Thus we provide LaTeX environments that authors use to mark sections of material as being appropriate only for certain media. Again, feedback is needed for proofreading, so such marked material is indicated by bars in the margin with an appropriate label.

3.3. Multiple Media

Although the book will be printed in a two column format, other media will generally be in a single column — but without a known, fixed width. We therefore strongly prefer that authors *not* break their formula by hand.

We have adopted Michael J. Downes' equation breaking package, `breqn`[3]. It automatically breaks formulas across multiple lines depending on the current column width. Although this package is experimental, and the formulas we encounter stress the package, it is handling the task remarkably well. By hiding the invocation of `breqn` inside the standard `equation` environment, the author ideally is not even aware of its presence.

Nevertheless, there are formulas, as well as tables or other material, that the author knows will be too wide to be successfully stuffed into a narrow column. For this, a `onecolumn` environment is defined. In order to support this, and balanced switching between one and two column modes, we have integrated Frank Mittelbach's `multicol` package[6, Ch. 3]. Again, other than the `onecolumn` environment, the author need not be aware of its presence.

3.4. Mathematical Markup

The range of mathematics represented in the DLMF, primarily algebra and calculus, is considerably narrower than the field of mathematics as a whole. Thus the range of notations — and ambiguities — we must contend with is considerably smaller. Nevertheless, there are still problems.

The intention is to adapt an infix parser to the task of transforming the LaTeX markup into semantic form. Since the goal is to only minimally extend the markup, the parser must accept implicit operations; in our application this will generally be implicit multiplication, but we anticipate the need for a type inference engine to resolve ambiguities. To further reduce the amount of guesswork such a parser must carry out, we have developed several sets of markup described in the following sections.

3.4.1. *Special Functions*

Two of the most widespread ambiguities in the field of special functions, although they are not unique to that field, are the roles of parentheses and superscripts. A parenthesis may indicate either grouping or function application. Superscripts may indicate a power, a function parameter or a derivative. Both of these are partly resolved by declaring all mathematical functions (as LaTeX macros) and incorporating the various sub/superscript parameters into the macro definition. Thus, any remaining superscript can only be a power or derivative. A parenthesis following a function macro probably contains the arguments (and since we know which special function is involved, we know how many arguments to expect), otherwise the parenthesis is for grouping.

This distinction between the function 'parameters' (e.g. the sub- and super-scripts) and 'arguments' (e.g. the parenthesized, or fenced, arguments) is consistent with LaTeX conventions. While the macro `\sin` 'names' the sine function (without arguments), one typically 'names' the Bessel function by writing J_ν (Strictly speaking, the function J of two arguments, ν and z, is curried to a function of z alone).

Consequently, the LaTeX macro we have defined for the Bessel function takes the parameters as arguments. Thus, one refers to the Bessel function itself by

$$\texttt{\textbackslash BesselJ\{\textbackslash nu\}} \to J_\nu$$

but the application of the function, say in an expression, would be

$$\texttt{\textbackslash BesselJ\{\textbackslash nu\}(z)} \to J_\nu(z)$$

Since the formatting of sub- and super- and pre-sub-scripts, tends to yield somewhat messy markup, eliminating that markup has the benefit

of easing both the author and parser's tasks, as well as standardizing the presentation of these elements.

And of course a non-trivial benefit is that we know *which* function J is being referred to, and indeed that it is a function at all.

3.4.2. *Fractions and Derivatives*
We further discourage various other typical LaTeX idioms that are likely to be ambiguous. Rather than using a / for inline fractions, we prefer to complete the set `\frac` and (amstex's) `\tfrac` with `\ifrac` (which, admittedly, are providing presentation hints, by indicating normal, text or inline fraction).

Likewise, rather than relying on a parser to wonder at the significance of fractions that have non-canceled 'd' in the numerator and denominator in `\frac{df}{dx}`, derivative macros `\deriv`, `\pderiv` (along with text and inline variants, and optional 'n'), macros for differentials, etc. have been defined.

3.4.3. *Continuing work*
The macros described above are currently being used by the authors with apparent success. Yet, the macros do not quite reduce ambiguities to the desired extent, particularly regarding powers and derivatives using superscripts, and the status of parentheses is somewhat uncomfortable. The main extension being developed is that an optional `@` sign will be taken as introducing the argument list (with the mnemonic 'function evaluated at a point'). Thus,

$$\texttt{\textbackslash BesselJ\{\textbackslash nu\}@\{z\}} \rightarrow J_\nu(z).$$

Further, authors often wish to incorporate operations, particularly powers and derivatives between the function and its arguments. We must recognize the semantic import, while still accepting the presentational hint. We extend the syntax by allowing a primes (or a backprime taking a macro argument), which applied to a mathematical function, is taken as signifying single (or multiple) derivative; the caret is then reserved solely for powers. Inclusion of other declared operators needs also be considered (eg. transpose, conjugation, etc.).

Thus the following markup could be employed

$$\texttt{\textbackslash BesselJ\{\textbackslash nu\}''@\{z\}} \rightarrow J''_\nu(z)$$

or

$$\texttt{\textbackslash HypergeoF\^{}\{2\}@\{a,b\}\{c\}\{z\}} \rightarrow F^2\left(\begin{matrix}a,b\\c\end{matrix};z\right).$$

This additional markup seems only slightly 'odd' (from a LaTeX point of view), while significantly clarifying the author's intent.

At the same time, it still supports good presentation. It preserves the author's flexibility in choosing whether to write derivatives with embedded primes or as a prefix operation; similarly with powers. Standardized presentation of function arguments is also provided.

The special function macros also serve as a starting point for type inference, but further declarations will be needed to handle variables and operators, at least.

4. Mathematical Search

In traditional books and handbooks, and even in well-structured Web sites, an index and a table of contents are often adequate for finding information. In the DLMF Handbook and Web site, however, the contents contain so many formulas and other mathematical constructs that mere indexes and table of contents fall extremely short. Rather, a special search system has to be provided in order for users to quickly locate what they are looking for.

Furthermore, although text search and retrieval is a mature technology and many text search systems are available, math search presents new demands and issues that the text search community never had to face. To identify the major math search issues, it helps to consider the reasons why conventional search systems are inadequate for math search. We recognize three major reasons.

The first is that mathematical contents often involve non-alphabetical symbols that are not understood by current search systems, or at least not rightly interpreted. Terms like `Gamma(1/2)`, `P_n(x)`, `x**5`, or `d^2y/dx^2-x=0` are either meaningless or improperly read and processed by current systems.

The second and more challenging reason is that formulas and equations, as well as other mathematical constructs, have rich structures that convey much meaning. Current search engines are not "aware" of those structures, do not capture or index them, and are thus unable to search for information that involve structural relationships and patterns. A query like `sin(x + log x)` is no different to a current search system than `sin x + log x`. Similarly, `x (y + z)` is misinterpreted as `x y + z`, if interpreted at all.

The third and most challenging reason is that the many equivalent ways in which mathematical terms can be expressed, which correspond to synonyms in text search, are often much more complex than textual synonyms, and thus cannot be fully captured in a thesaurus. A summation or a product of two or more terms can be expressed in many equivalent ways due to commutativity and associativity laws. Numbers can be

represented in multiple forms (e.g., `1/2` vs. `0.5` vs `2^{-1}`). Polynomials can be expressed in many factored and unfactored forms. Trigonometric terms can be easily substituted by other equivalent trigonometric terms. Indeed, it can be argued that a large part of Mathematics is about the different and equivalent ways of expressing a concept or a quantity. Obviously, current search systems are not equipped to recognize those equivalences and take them into account when searching — indeed, the problem is not solvable in general.

Therefore, the major math search issues can be summarized as follows:

- Recognition and proper processing of mathematical symbols in mathematical content and queries.
- Capturing and indexing mathematical structures.
- Providing a math-appropriate query language and user interface that enable users to express their information needs, which often involve math symbols and structures.
- Developing and integrating techniques for taking into account mathematical synonyms and equivalences — at least some of the more common ones such as commutativity and associativity based equivalences.

The next subsection will discuss some of the approaches that are being taken to address those issues.

4.1. Approaches to Math Search

Broadly speaking, two general approaches suggest themselves. The first approach is an evolutionary one, which augments existing text search systems with math-appropriate searching capabilities. The second approach is to create a math search system wholly from scratch to fully capture and index mathematical content.

The augmentation approach leverages the power and maturity of text search engines, and involves less work. Although it has inherent limitations as to how much mathematical structure it can capture, it has taken us a long way toward effective equation-based searching. It is the approach currently adopted by the DLMF, and will be discussed in more detail in the next subsection.

The second approach is much more demanding in time and effort. It will require not only the careful use and integration of various computer algebra and symbolic processing techniques, but also the development of novel indexing and search techniques for which research has barely

begun. Certainly, parsing techniques for mathematical expressions have been developed for compilers and computer algebra systems [2, 13]. Those techniques can and should be adopted by this approach as well as by the augmentation approach.

The indexing of mathematical structures is a largely new subject. Some work has been done on searching for integrals [4]. However, to the authors' knowledge, no work has been published on indexing full mathematical structures in such a way that users can search for structures of parts thereof. For example, a user can search for a whole matrix or a submatrix, a fraction or a denominator, a function with its argument or just the argument, a summation or just the summand, and so on. This ability to search with queries that specify the patterns to be found in *named parts* of a structure is possible with the mathematical indexing approach but cannot be fully achieved through the augmentation approach.

As for incorporating mathematical synonyms and equivalences, some aspects of automated theorem proving [14] and symbolic processing in computer algebra can be drawn upon. Specifically, a thesaurus-like dictionary of abstract equivalence rules will have to be used by the search system to detect concrete instances of equivalences between the query form of an expression and the contents' form(s) of the same expression. Applying abstract inference (equivalence) rules in this context is tantamount to matching (i.e., instantiating) them to concrete instances, which is a task that automatic theorem provers generally carry out. Sophisticated equivalence detection, however, is not a mature technology, and is probably too slow for online searching. Nevertheless, for certain kinds of equivalences, equivalence detection is feasible and will be addressed in our future work.

4.2. Search Engine Requirements

To meet the imminent search needs of the DLMF, we pragmatically opted for the augmentation approach. The first step was to find a text search engine which has a wide range of capabilities and features to make it amenable to augmentation for math search. The second step was to carry out the actual augmentation.

There are a number of search systems, some of which are in the public domain [16, 20, 12, 15, 19] while many others are commercially available[2]. Clearly, what is important is not which specific system to go

[2] Identification is not intended to imply recommendation or endorsement by the National Institute of Standards and Technology, nor is it intended to imply that these systems are necessarily the best available for the purpose

with, but what features are the most relevant and useful for adaptation and augmentation to math search.

In our experience, we found that to ease augmentation to math search, certain capabilities and features are particularly desirable to have in a text search system. These capabilities and features are described in the next 4 subsections.

4.2.1. *Boolean and proximity operators*
All three Boolean operators (AND, OR, and NOT) and many of the proximity operators should be allowed in queries. Examples of proximity operators are word adjacency operators (for phrase search) as well as unidirectional and bidirectional distance operators that specify the distance between the operands in terms of the number of intervening words or intervening paragraphs.

4.2.2. *Fields and field-based search*
It is desirable to be able to partition any document in the database into parts (called fields), and label each part by a field name. Queries can be targeted not only to whole documents but also to specific fields. An important field might be "equation", another might be "theorem", and yet another might be "chapter-title". Hierarchical fields (i.e., a field could be made up of several fields, to any level of nesting), if available, give more flexibility to system designers and to search users.

4.2.3. *Thesaurus support*
The thesaurus would allow the math search system designers to define various terms not only in terms of equivalent synonyms, but also as full-fledged queries (i.e., an expression involving Boolean and/or proximity operators). For example, it was mentioned earlier that markup such as `BesselJ` (and `BesselI`, `BesselK`, `BesselH` etc.) helps in semantics and formatting matters. Should a user enter `Bessel` as a keyword, all those markup words (`BesselJ`, `BesselI`, etc.) will not be recognized as matches of `Bessel`. However, if an entry for `Bessel` is defined in the thesaurus, where `Bessel` is made to stand for

```
(Bessel OR BesselJ OR BesselY ...),
```

a thesaurus lookup will cause the keyword `Bessel` to be replaced by its thesaurus entry, and the markup is then guaranteed to be matched.

Note, however, that such a thesaurus cannot handle mathematical equivalences that involve formulas.

4.2.4. *Allowance for surrogate files*
The underlying concept is to view a document as a pair of files (D, S). File D is the display file that is presented to the user when the document matches a query and is retrieved. The file S (the surrogate file) is what is indexed at the time of creating the database or when adding the document to the database. The key point is that the two files D and S need not be identical. Indeed, the surrogate file S can have:

- Non-displayable metadata, such as (a) authors' notes, (b) descriptive terms and keywords for equations, formulas, graphs and tables, and (c) annotations.

- Modified representations of mathematical equations, symbols and other constructs so that those representations are indexable and searchable by text search systems. Roughly speaking, the modified representation corresponds to *content markup* in the parlance of MathML [23] and OpenMath [21], and the original representation corresponds to presentation markup as in MathML, LaTeX, and other word processing systems. Note that the modified representation could be standard markup such a in MathML, but it need not be, depending on the intended use.

The main reason for having a pair of files is that display files do not have all the textual (and thus indexable) contents needed, and, on the other hand, the surrogate files have contents that are not for display. In practice, D is a subset of S. File S is created at indexing time, and can later be discarded (after the index has been generated); only file D need be kept.

4.3. Search Augmentation

For the augmentation strategy to work, we need to perform the following fundamental tasks:

Textualization of all symbolic entities in the contents and in the queries. This trivial but necessary step means mapping non-alphanumerical characters to alphanumerical strings. For example, + is turned into "plus" and < into "lt". This is needed since text search systems do not recognize symbols like +, /, < or ^ as known mathematical operators. The textualizing process merely involves substituting non-alphanumerical characters with pre-defined alphanumerical counterparts.

Flattening of mathematical equations & structures in contents and in queries, that is, converting them into a linear, sequential form.

Table I. Illustrations of Textualization and Flattening

Original	Textualized & Flattened Form
`x^{t-2} = 1`	x BeginExponent t minus 2 EndExponent Equal 1
`f(x)` or `f@(x)`	f ApplyAt BeginArgument x EndArgument
`(x+1)/(x-1)`	fraction (x plus 1)(x minus 1) *-alternatively-* fraction BeginNumerator x plus 1 EndNumerator BeginDenominator x minus 1 EndDenominator

Flattening is needed because the closest thing to capturing structures in text search is through linear contiguity & proximity operators.

The main task of flattening is to identify and scope the parts of a formula such as numerators, denominators, exponents and arguments, and to surround them with pairs of matching markup tags. Table I below illustrates the concept.

The main approach, which performs both textualization and flattening, is as follows. First, each formula is parsed into a tree where each node is labeled with the corresponding entity in the formula (e.g., "+", "2", "x", and "^"). Second, the parse tree is traversed and converted into a textualized sentence with all the proper matching pairs surrounding formula parts. The granularity of the parts to be scoped depends on how much specificity we want to allow the users to have when formulating queries.

Flattening deserves special attention, not only because it is inherent to text search, but also because it creates a new problem stemming from the multiplicity of linear forms that an expression can be mapped to. For example, x_2^3 can be flattened in two ways, depending on whether we put the subscript before or after the superscript. If the order in the contents is different from the order in the query (after flattening), then an undesirable mismatch results. Hence the need for the next task.

Normalization of the linear forms, that is, converting the linear form of each expression/formula into a single format, called a normal form.

Table II. A Sample of Operators in the Math Query Language

Operator	Meaning
`+, -, /, *`	the standard arithmetic operators
`^, **`	Superscript or power
`_`	subscript
`@`	apply at as in f(x) or `f@(x)`
`=>, <=>, <==`	imply, equivalent, if
`!=, ~=, not=`	not equal
`=-`	equivalence ($\equiv$)
`=~`	congruence ($\cong$)
`=.`	dot equal ($\doteq$)
`-~`	similar-to or equal ($\simeq$)
`~`	similar-to ($\sim$)
`~~`	approximate ($\approx$)
`|.|`	determinant, magnitude

The normal form is a data model that should be generic enough to model all expressions/formulas. We define a new data model called the sorted parse tree normal form. It is a parse tree where the children of any nodes are sorted from left to right whenever changing the order of the children does not alter the mathematical meaning, such as when the node is an associative and commutative operator, and the children are its operands. The ordering is based on any numerical or alphabetical associated key. For example, we can sort the children based on the number of descendent nodes that each child has in the parse tree. If two nodes have the same number of descendants, then we sort them alphanumerically based on the entity labels of the nodes. The normalization algorithm is an algorithm that sorts the parse tree.

Development of a query language to express math queries as well as text queries. The math query language will be similar to but much simpler than LaTeX, where many LaTeX word-commands are replaced by short yet intuitive strings. The language incorporates notations from other languages. Table II gives a sample of operators, and Table III illustrates math queries.

We built an augmented search system for the DLMF. Its principal modules are described next.

Table III. Illustrations of Math Queries. Each query in the left column finds documents having formulas that contain the corresponding matched element in the right column of the table.

Query	Matched element
`x^2`	x^2
`(x OR t)^3`	x^3 or t^3
`$^3`	any single-character cubed
`sqrt(x^2+1)`	$\sqrt{x^2+1}$
Gamma(1/3)	$\Gamma(1/3)$
`^(x+2)`	(x+2) in an exponent part
`/(x+2)`	(x+2) as a denominator
`(... x+2 ...)/`	(x+2) embedded in a numerator

Math query language: Our language, illustrated in Table III, is flexible, intuitive, easy-to-use, and efficiently expressive. It is flexible in that it accommodates a variety of "idioms" for operator symbols, e.g., MATLAB, C, FORTRAN, LaTeX, and others. It is intuitive and easy to use in that it draws on what most users know and use in their daily math reading and writing. It is efficiently expressive in that a formula (or fragment thereof) can be expressed in a minimum number of keyboard characters, without resorting to lengthy and elaborate commands and markups. The details of the language will be in the help file of the system, and will be published elsewhere.

Math query parser/translator: This is a front-end layer to parse and translate formula queries to purely textual-and-numeric queries that use Boolean and proximity operators. This module textualizes, flattens, and normalizes the users' queries. Table I illustrates some aspects of the work of this module.

Contents transformation module: This transforms the mathematical content in the DLMF database, especially equations, formulas, and mathematical symbols, into a textual-numerical normalized form. This module and the query parser/translator perform some of the same processing – textualization, flattening, and normalization. However, they still differ in that the parser's input format, that is, the math query language syntax, is different from the transformation module's input format (e.g., the Latex encoding of math formulas in documents).

Surrogate-files generator: This module generates the surrogate files that contain the transformed math content as well as other metadata. It clearly calls upon the content transformation module to modify the mathematical representation into an indexable form, as discussed earlier.

Thesaurus: The thesaurus, which will continue to evolve, contains generic math entries, DLMF-specific entries, and markup related entries.

An example of a generic math entry is that of the term *derivative*, which includes among its synonyms the term *differential* or *diff* for short. An example of a DLMF-specific entry is:

```
U: (U OR VoigtU OR KummerU OR WhitU
    OR ChebyU OR ChebyUs)
```

which tells the system that U should be interpreted as the OR of generic U, the Voigt function U (through its local markup `VoigtU`), the Kummer function U (through its markup `KummerU`), and so on. Note that the latter example illustrates also the use of the thesaurus for capturing internal markup.

Those modules have been designed and implemented, and a working math search system is in place. The system has math search capabilities that far exceed those of text search engines. Still, the system is work in progress. Additional refinements are being made, and new capabilities are being added.

Among those new capabilities are (1) the proper handling of customary names and symbols that are widely used in the area of special functions, and (2) the support of basic equivalences, including commutativity and associativity, among others.

In the future, we will explore searching based on new and emerging encodings such as MathML and OpenMath. The precise direction that will be taken will be determined according to which of the new encoding standards will be adopted and become widely used and technology-supported.

Also, an all-math search engine, created from scratch for that purpose, may be developed, if and when the necessary research in indexing of mathematical structures yields the necessary understanding and techniques. Experience gained from the mathematical and scientific user community regarding math search, especially in reference to query languages, modes of search, and what kinds of math information users usually need to search for, will be an extremely valuable guide into building future math search systems.

5. Conclusions

At this stage of the DLMF project, we are still very much focused on obtaining the content. Much of the effort up to this point has gone into TEX programming to provide support for a subtly modified level of LATEX markup. The goal is to allow authors to minimize presentation markup and ambiguity, while maximizing content markup. We try to enable the authors to concentrate on the mathematics and develop the content as painlessly as possible. The ulterior motivation for this is that the material can then easily be presented in different media: in print and on the web.

We are moving into the next phase, focusing on the transformation to XML and Presentation MathML. This latter, using systems such as `tex4ht`[7, Ch. 4] seems within reach, although some refinement is still needed.

It is clear that as we move into further stages, parsing into fully Content MathML or OpenMath, additional refinement and markup of the mathematics will be necessary, even adoption of more verbose, unambiguous markup. But, with the approach outlined we should already be relatively close to that state. As long as *we* understand the author's intent, we will be able to augment the sources with additional markup and declarations to complete the parsing. But while the tools to carry out that parsing are not yet available, applications to *use* such content markup are also not yet widespread, so the pressure is not yet intense.

Between the metadata markup and modestly semantic mathematical markup, we already have sufficient data for the augmented text-based search techniques described here. We believe that with this approach powerful search capabilities will be available to DLMF users. Our approaches are rather generic, and should be immediately applicable to other mathematics databases. If these methods prove insufficient, and once we have made further progress in capturing the complete structure of the mathematical data, we will look into true mathematical search techniques.

References

1. Abramowitz, M. and I. A. Stegun (eds.): 1964, *Handbook of Mathematical Functions*, National Bureau of Standards Applied Mathematics Series No. 55. U.S. Government Printing Office, Washington, DC.
2. Aho, A., R. Sethi, and J. Ullman: 1985, *Compilers : Principles, Techniques, and Tools*. Addison-Wesley.
3. Downes, M. J.: 1997, 'Breaking equations'. *TUGboat* **18**(3), 182.

4. Einwohner, T. H. and R. Fateman: 1995, 'Searching techniques for integral tables'. In: *Proceedings of the International symposium on Symbolic and algebraic computation*. p. 216. http://torte.cs.berkeley.edu:8010/tilu.
5. Fateman, R.: 1992, 'Honest Plotting, global extrema, and interval arithmetic'. In: *Proceedings of the International symposium on Symbolic and algebraic computation*. p. 216.
6. Goossens, M., F. Mittlebach, and A. Samarin: 1994, *The LaTeX Companion*. Addison Wesley.
7. Goossens, M. and S. Rahtz: 1999, *The LaTeX Web Companion*. Addison Wesley.
8. Knuth, D. E.: 1986, *The TeXbook*. Addison Wesley.
9. Kowalski, G.: 1997, *Information Retrieval Systems, Theory and Implementation*. Kluwer Academic Publishers.
10. Lamport, L.: 1994, *LaTeX: A Document Preparation System*. Addison Wesley, 2nd edition.
11. Lozier, D. W.: 2002, 'The NIST Digital Library of Mathematical Functions Project'. *Annals of Mathematics and Artificial Intelligence*. (see preceding paper in this journal).
12. Manber, U., S. Wu, and B. Gopal, 'Glimpse and WebGlimpse'. http://glimpse.cs.arizona.edu/.
13. Muchnick, S.: 1997, *Advanced Compiler Design and Implementation*. Morgan Kaufmann.
14. Newborn, M. and M. Newborn: 2001, *Automated Theorem Proving : Theory and Practice*. Springer Verlag.
15. NIST, 'IRF'. http://www-nlpir.nist.gov/projects/irf/.
16. PLS/AOL, 'CPL and PLWeb'. http://www.pls.com/.
17. Salton, G. and M. J. McGill: 1983, *Introduction to Modern Information Retireval*. McGraw Hill, New York.
18. Saunders, B. and Q. Wang: 2000, 'From 2D to 3D: Numerical Grid Generation and the Visualization of Complex Surfaces'. Technical Report NISTIR 6555, NIST.
19. SWISH-E, 'Simple Web Indexing System for Humans — Enhanced'. http://swish-e.org/.
20. The ht://Dig Group, 'ht://Dig'. http://www.htdig.org/.
21. The OpenMath Society, 'OpenMath'. http://www.openmath.org/.
22. Wang, Q. and B. Saunders: 1999, 'Interactive 3D Visualization of Mathematical Functions Using VRML'. Technical Report NISTIR 6289, NIST.
23. World Wide Web Consortium: 2001, 'Mathematical Markup Language (MathML) Version 2.0'. http://www.w3.org/TR/MathML2/.