Extreme Markup Languages 2007®

Montréal, Québec
August 7-10, 2007

A More Canonical Form of Content MathML to Facilitate Math Search

Moody E. Altamimi
The George Washington University

Abdou S. Youssef
The George Washington University

Abstract

Math search is a new area of research with many enabling technologies but also many challenges. Some of the enabling technologies include XML [Extensible Markup Language], XPATH [XML Path Language], XQUERY [XML Query], and MATHML [Mathematical Markup Language] (in particular, the content part of MathML). Some of the challenges involve enabling search systems to recognize mathematical symbols and structures. Several math search projects have made considerable progress in meeting those challenges. One of the remaining challenges is handling notational equivalences. Even though Content MathML has a great impact on the area of math search since it enables a consistent and less ambiguous representation of mathematical expressions, different yet equivalent encodings for the same expression can still be used. In addition, author-defined functions and symbols, an important feature of Content MathML, proves to be a challenge to the area of math search. Authors can associate types with variables and also define functions that can be referenced later throughout the document. One attempt to handle these issues is through the implementation of techniques to transform mathematical documents into a normalized form. The definition of what constitutes a normalized form of a math document is important in that it sets the path for the creation of a more canonical version of Content MathML that is less ambiguous and thus enables better math search capabilities. This paper will discuss these challenges and describe the proposed canonical form of Content MathML in detail.

Extreme Markup Languages®

A More Canonical Form of Content MathML to Facilitate Math Search

Table of Contents

Extreme Markup Languages®

A More Canonical Form of Content MathML to Facilitate Math Search

Moody E. Altamimi and Abdou S. Youssef

§ Introduction

The need to facilitate scientific information exchange between researchers has resulted in the creation of a growing number of specialized mathematical library projects around the world. These projects aim to make scientific literature available on the Web. With the increasing online availability of electronic documents that contain mathematical expressions, the ability to find relevant information has become increasingly important. Yet, support for searching for mathematical expressions is only in its infancy. Even though search technology today is mature, it only facilitates text search and does not handle non-alphanumeric symbols or non-linear (i.e., non-sentence-like) structures. For example, a user cannot search for a simple equation such as $y = x + 3$ in a search engine like Google.

Search for math content is possible using keywords (text) describing the mathematical content in a web document. However, this capability is not enough to fully utilize the many mathematical resources available today. In addition, traditional indexing and search techniques, developed for text, are not applicable to mathematical expressions encoded in these documents. Text is a linear organization of letters and words, while math expressions are two-dimensional arrangements of symbols. Consequently, the nature of mathematical content poses several challenges to the indexing and search research communities.

A mathematical expression is used to communicate ideas using an arrangement of symbols. Different ideas can be expressed using the same notation, and different notations can be used to describe the same idea. As a result, to describe mathematical content properly, a description of how the symbols are arranged needs to be encoded, in addition to a separate encoding of the underlying organization of ideas. The conceptual structure eliminates the inconsistency and ambiguity problems of presentational structures. This is necessary when designing applications that process mathematical content.

A search tool cannot simply "guess" what the user really intended when searching for $g(x + y)$: Is it an arithmetic operation (multiplying a variable g by the sum of variables x and y), or is it an algebraic function (function g applied at the sum of variables x and y as an argument)? The development of Content MathML, an XML-based language, for encoding mathematics was a necessary technology for any attempt to solve the math search problem.

The development of math search capabilities is a new area of research with many technical challenges. Some of the challenges involve enabling search systems to recognize mathematical symbols and structures. Several math search projects have made considerable progress in meeting those challenges. Several research projects on math search have addressed many of the issues and challenges in math search. Notable among those math-search projects are the math search of the DLMF project at NIST [DL], and the math search system of Design Science [MD]. Math content in these projects however is encoded using LaTeX and Presentation MathML. Both encodings capture presentational aspects of the content which does not resolve the notational ambiguity and inconsistency inherent in mathematical notation. Addressing these limitations inevitably requires using an unambiguous markup language (Content MathML) that captures the underlying structure of mathematical constructs. Moving in that direction will enable the development of more powerful mathematical search techniques.

This research is part of an ongoing effort to develop more powerful mathematical search techniques utilizing XML-based technologies and standards. A prototype that implements these search techniques, taking into account the nature of mathematical notation, and enabling search for full and partial mathematical expressions is currently being developed. A math query language enabling science and math users to specify their information needs in a more precise and concise way has been developed.

The implementation of the language maps queries written in that language into XPath/XQuery queries [XP 05] [XQ 07]. It is assumed that the math documents are encoded in Content MathML [MML 03]. The justification for this assumption is based on current technological advances and expected future practices. For example, many conversion tools already exist for converting LaTeX to MathML, such as the Rice University tool for conversion to Content MathML [WIN 06], and Bruce Miller's LaTeXML and

associated software [MILL 06], which converts from LaTeX to a special XML syntax that includes presentation MathML and some content mark up. Furthermore, as the math authoring community becomes more comfortable with MathML and, more importantly, becomes more convinced of the need for and benefits of Content MathML, more conversion tools and authoring tools that yield Content MathML will become available and more widely used.

Even though Content MathML has a great impact on the area of math search since it enables a consistent and less ambiguous representation of mathematical expressions, different yet equivalent encodings for the same expression can still be used. In addition, author-defined functions and symbols, an important feature of Content MathML, proves to be a challenge to the area of math search. Authors can associate types with variables and also define functions that can be referenced later throughout the document. One attempt to handle these issues is through the implementation of techniques to transform mathematical documents into a normalized form. The definition of what constitutes a normalized form of a math document is important in that it sets the path for the creation of a more canonical version of Content MathML that is less ambiguous and thus enables better math search capabilities. This paper will discuss these challenges and describe the proposed canonical form of Content MathML in detail.

§ Background and Related Work

This section starts off with a discussion about the challenges posed by mathematical notation in the area of math search. This will be followed by a description of Content MathML, its imporance in the area of math search and how it tackled some of these challenges. The section also describes the canonical form of Presentation MathML used by UMCL [Universal Maths Conversion Library] project [UMCL 06].

The Nature of Mathematical Notation

The visual structure of a mathematical expression is referred to as the presentational structure [SK 82]. The underlying organization of ideas, the content to be communicated, is referred to as the conceptual structure of the mathematical expression. For example, the superscript presentational structure x^2 is used to communicate the underlying idea of *raising x to the power of 2*. A reader can infer the expression's implicit conceptual structure from its presentational structure and resolve ambiguity based on its context. For example, when a reader encounters a polynomial $ax^2 + bx + e$, e will not be mistaken to be the base of the natural log $e = 2.2789$. But when encountering $e^{ix} = -1$, the reader is safe to assume that e in this context is the base of the natural logarithm and conclude that x=pi [CAP 99]. A software system cannot resolve this ambiguity and *understand* the mathematics from the given context without additional information describing mathematical constructs in an unambiguous fashion [CAP 99].

The extensibility and reusability of mathematical notation is what makes the notation difficult to process due to its inherent ambiguity and inconsistency. Ambiguity is a result of the writing system being reusable. For example, the expressions $g(x + y)$ is used to communicate two different concepts: *multiplying a variable g by the sum of variables x and y*, a basic arithmetic operation, and *function g with the sum of variables x and y as an argument*, an algebraic function. Inconsistency is a result of the writing system being extensible. For example, *dividing x over y* is a basic math operation that can be expressed using three different notation $\frac{x}{y}$, x / y, xy^{-1}.

To resolve any questions about the underlying organization of ideas that is being communicated by the expression, the expression's conceptual structure must be specified. More importantly, accessing conceptual structures of mathematical expressions facilitates the indexing and retrieval of those expressions [CAP 99]. When searching for a mathematical expression such as $1 / a$, the search will be for the underlying conceptual structure *one over a* rather than for mathematical expressions that are visually rendered in a specific way. Documents where *one over a* exists should be retrieved regardless of whether the concept is visually rendered as a^{-1} or $1 / a$. In order to do that, it is necessary use specialized markup languages that capture that structure .

Content MathML

MathML provides two different sets of tags for capturing mathematical content's visual and conceptual representations [MML 03]. Presentation MathML is like LaTeX in providing a wide range of symbols and constructs such as superscripts and alignments [CAP 99]. Presentation MathML is used to encode the 2-dimensional layout and formatting of the notation that visually illustrate the expression, while Content MathML is used to encode the underlying structure of mathematical concepts represented by that collection

of notation. MathML, with its two separate encodings for both presentation and conceptual aspects of mathematics, allows mathematics not only to be displayed, but also to be exchanged and processed on the Web. In addition, it is a W3C [World Wide Web Consortium] recommendation [MML 03].

For the purpose of developing powerful math search systems, Content MathML is more appropriate than Presentation MathML because encoding the underlying conceptual structures resolves ambiguities and clarify inconsistencies inherent in mathematical notation; therefore, Content MathML will be our focus rather than Presentation MathML. There are 120 Content MathML elements that explicitly encode widely used operators, functions and relations [MML 03]. Some of the topics covered by Content MathML elements are arithmetic, algebra, logic, calculus, relations, set theory, etc. Content MathML also offers an additional method for encoding mathematical notations not codified using these explicit elements. The conceptual structure of a mathematical expression, its expression tree, is explicitly encoded using Content MathML elements. Data types such as identifiers, numbers and user-defined symbols are encoded using token elements (ci, cn, and csymbol), while operators and functions are encoded using explicit operator elements such as plus, times, etc. [SAN 03].

Using constructor elements, such as apply, operator and token elements become related to each other in prefix notation. For example, to encode an expression $x + y$ using apply, the operator element plus is related to its token element arguments x and y:

<apply> <plus/> <ci>x</ci> <ci>y</ci> </apply>

Through the recursive use of constructor elements, more complex expression trees can be built. For example $z(x + y)$:

<apply> <times/> <ci>z</ci> <apply> <plus/> <ci>x</ci> <ci>y</ci> </apply> </apply>

In addition, some elements are used to represent constants and widely used mathematical symbols. For example, <pi/> is used to represent pi. Qualifier elements are elements that are used to add more meaning to other elements [SAN 03]. For example, using the int element to represent the integral operator, uplimit and lowlimit elements can be used to indicate the upper limit and lower limits:

<apply> <int/> <bvar><ci> x </ci></bvar> <lowlimit><cn> 1 </cn></lowlimit> <uplimit><cn> 2 </cn></uplimit> <apply> <fn><ci> f </ci></fn> <ci> x </ci> </apply> </apply>

UMCL: Universal Maths Conversion Library

Archambault and Moco describe a canonical form of the presentational aspect of MathML to help transform mathematical content into mathematical Braille [AR, MO 06]. Their work is part of the UMCL project that allows the conversion between different Braille code notations and Presentation MathML. Presentation MathML as discussed earlier is inherently inconsistent, the same operation can be presented using different notation. This makes it more difficult to transform into Braille. They propose a more canonical form of Presentation MathML that creates unique presentations of mathematical content. The will make the transformation into different Braille codes easier and more consistent.

§ Transformations from an arbitrary form of Content MathML to a more canonical form

Content MathML has a great impact on the area of math search since it enables the explicit encoding of tree structures of mathematical expressions, thus resulting in a more consistent and less ambiguous representation of mathematical content. However, there is a need to create a more canonical form of the language to handle the following issues:

- Notational equivalences: Even though ambiguities are resolved as to what the author intended when using Content MathML encoding, Content MathML allows the author to use different yet equivalent encodings for the same expression. For example, when the user makes a search for $\int_0^a \sin(x)$, documents where the integral's domain is specified using an interval and encoded as $\int_{[0,\, a]} \sin(x)$ need to be retrieved as well. These equivalences need to be identified and a decision as to which form should be used in a document must be made.
- Author defined functions: One of the most important features of Content MathML is allowing the author to associate types with variables and also define functions that can be referenced later throughout the document. This is a challenge to the area of math search because if the author declared

f as $x + y$ and the user is searching for $x + y$, expressions including f where f is defined as $x + y$ need to be retrieved.

One attempt to handle some of these issues is through the implementation of techniques that transform mathematical documents into a normalized form. The definition of what constitutes a normalized form of a math document is important in that it sets the path for the creation of a more normalized version of Content MathML that is less ambiguous and thus enables better math search capabilities. Algorithms need to be developed to take user authored MathML documents and transform them into a normalized canonical form according to well-defined specifications. This process does not validate MathML documents; the assumption is that these documents are syntactically correct. For those issues that cannot be resolved in the normalized form of a document, Boolean OR queries need to be created that query for multiple MathML representations during the math query-processing phase.

Notational Equivalences in Content MathML

One of the major issues when dealing with the math search problem is the handling of equivalences. In mathematics, mathematical concepts can be expressed in different and equivalent forms, where each form has its own underlying conceptual structure to be encoded. When developing a math search system, it is important to develop techniques that take these equivalences into account when searching for a mathematical expression or a term. The use of a thesaurus has solved the equivalence problem in the area of text search, but not enough to fully capture the complex structure of mathematical expressions [MI, YO 03]. Special data structures and thesaurus-like dictionaries are needed to resolve this issue but this is out of the scope of this research. This research focuses on mathematical equivalences that are inherent in Content MathML, when a mathematical expression has equivalent Content MathML representations.

The identification of areas where different yet equivalent Content MathML representations appear is an important research issue. These notational equivalences can be classified into equivalences that can be resolved using a canonical form and equivalences that need to be handled during the math query-processing phase. The following sections identify notational equivalences in Content MathML version 2.0 that are addressed in this research.

Handling of Equivalences due to Commutative and Associative laws

Many common equivalences in mathematics are due to commutative and associative laws. In the math search problem domain, handling these equivalences poses a challenge, but the nature of Content MathML and the use of XPath queries make way for an easy solution. Using Content MathML, a mathematical expression is encoded as an ordered labeled tree where nodes correspond to operators, relations, functions and values that correspond to the prefix notation of that expression; where an apply element is used to apply operators and functions to a set of arguments. For example, when encoding $2 + 3$, a prefix tree will be created where the apply element is the parent of three children $+$, 2 and 3

```
<apply> <plus/> <cn>2</cn> <cn>3</cn> </apply>
```

When creating an XML Query to search for that expression, the search will be for a specific structural pattern *(an apply with three children: plus, 2 and 3)* where the order of the children can be specified or ignored. As a result, in the case of commutative binary operator: $+$, $\times$, and, or, xor, order is not important. Consequently, the equivalences related to commutative law will be handled on the math query processing side by simply identifying these operators and creating the corresponding XML Query where the order of these operands is immaterial.

Handling equivalences due to associativity is also straightforward because binary operators where the associativity law holds ($+$, $\times$, and, or, xor, $>$, $>=$, $<$, $<=$) correspond to n-ary operators in Content MathML. For example, when encoding the following expressions $2 + 3 + 4$, there can be two equivalent Content MathML encodings, the first encoding treats plus as a binary operator applied to two operands 2 and sub expression $3 + 4$:

```
<apply> <plus/> <cn>2</cn> <apply> <plus/> <cn>3</cn> <cn>4</cn> </apply> </apply>
```

The second encoding treats plus as an n-ary operators; the following prefix tree is created where the plus operator has three operands:

```
<apply> <plus/> <cn>2</cn> <cn>3</cn> <cn>4</cn> </apply>
```

Converting binary operators that satisfy the associativity law into n-ary operators offers an easy solution to this problem. As a result, during the processing of mathematical content, whenever an associative relation such as $(2 + 3) + 4$ or $2 + (3 + 4)$ is encountered, it needs to be converted into an n-ary relation

where + is applied to three operands 2, 3, and 4. This will dictate a conversion of the math query that searches for $2 + (3 + 4)$ or $(2 + 3) + 4$ into an n-ary relation and create the corresponding XML Query pattern for which to search. If the operator requires order of its elements (> , >= , < , <=) order will be specified in the XPath expression accordingly. Clearly, binary operators that are defined as n-ary operators in Content MathML are to be transformed into n-ary operators on the math query and content sides.

Inverse hyperbolic and trigonometric functions

The inverse of a function in Content MathML is handled by applying the inverse element to a function. For example the following is the encoding of the inverse of the sin function $\sin^{-1}(x)$:

<apply> <apply> <inverse/> <sin/> </apply> <ci>x</ci> </apply>

Additionally, Content MathML offers a set of explicit predefined inverse functions for hyperbolic and trigonometric functions, arcsin for example, and results in the following encoding.

<apply> <arcsin/> <ci>x</ci> </apply>

Inverse hyperbolic and trigonometric function have two equivalent representations $\arcsin(x)$ vs. $\sin^{-1}(x)$ for example. The two representations are allowed by Content MathML because some documents will include $\sin^{-1}(x)$ as a description of $\arcsin(x)$ [MML 03]. As a result, this equivalence issue cannot be resolved by transforming instances of one encoding into the other without changing what the author intended. Additionally, the user should be given the ability to use both formats when searching.

This raises an important issue: when the author of a document specifically chooses one equivalent encoding over the other. In this case, Content MathML should be extended to allow specifying that certain elements cannot be changed during the normalization of documents. This can be done by adding a **use** attribute to an element that has two values **fixed** and **changeable** where changeable is the default value.

Consequently, on the query side, the user should be given the ability to specify when a term can undergo transformations, and when it should be taken literally without any transformations.

Applying operators to functions directly (using functions as operands directly)

Functions can be used as operators within apply elements or can be used as arguments to other operators. For example, the expression $\sin + \cos$, can be represented in the document as follow:

<apply> <plus/> <sin/> <cos/> </apply>

This is considered the addition of sin and cos functions in some function space according to the MathML specification document. However, in most cases a function is applied to its operands by enclosing it in an apply element and results in the following encoding:

<apply> <plus/> <apply><sin/>....</apply> <apply><cos/>....</apply> </apply>

If the user is searching for an addition operator applied to a sin and cos functions, then the user is searching for $\sin + \cos$; the assumption is that he/she intends to search for sin added to cos in the second example regardless to what they're applied to. The two representations are similar where the first is a more general form of the other. However, both representations need to be allowed in the canonical form because an author can create a function as a complex expression, $(F + G)(x)$ for example, where the plus operator will clearly use the more general form of encoding.

Another example is applying the integral to a sin function directly where the user is looking for the integral of a sin function that can be represented in the document as $\int \sin$ or $\int \sin(x)$ with the following Content MathML representations:

<apply> <int/><sin/> </apply>

or

<apply> <int/> <apply><sin/><ci>x</ci></apply> </apply>

A more complicated example is when applying an operator to a function that is represented as an identifier, as in the application of the differentiation operator to a function f. When querying, the user enters **diff (f)**. On the document side, the derivative of a function f can be represented as f' or $f'(x)$

<apply> <diff/> <ci> f </ci> </apply>

or

<apply> <diff/> <apply> <ci type="function"> f </ci> <ci> x </ci> </apply> </apply>

When querying, both representations need to be retrieved.

As a result, when a user is searching for an operator with functions as its argument in the general form, the issue will be resolved on the query processing side by generating an XML Query to look for both representations.

Using identifiers as functions

When a variable f, for example, is considered as a function and applied to an argument, it can be represented in two ways, one way where the attribute type is used to specify that f is a function, and the other where the type attribute is omitted.

<apply> <ci type="function">f</ci> <ci>x</ci> </apply>

In the canonical form, when a variable is applied to any number of arguments, its type needs to be declared explicitly. This will provide for a better search when a user is looking for a function f. In this case, an XML Query expression will simply look for an identifier with the type attribute specified as a **function**.

Another issue is user-defined functions where the function itself is a compound expression. For example $(F + G)(x)$. Even though the plus operator is applied to two variables, clearly in this case the two variables are intended to be functions. These cases need to be identified and normalized accordingly by adding the type attribute set to **function**.

<apply> <apply> <plus/> <ci> F </ci> <ci>G </ci> </apply> <ci> x </ci> </apply>

Handling MathML representations from version 1.0 that are discouraged

The canonical form will not allow the use of deprecated representations from Content MathML version 1.0 that are discouraged. For example, the fn and reln elements are deprecated in MathML 2.0 in favor of the apply element and will not be supported by our canonical form. Another example is how MathML 2.0 represents mathematical constants. MathML 2.0 offers a more compact form where the constant is represented by a single construct. For example, imaginary I is represented using Content MathML 2.0 as <imaginaryi/>. In addition, MathML 2.0 supports two forms of representation of mathematical constants from version 1.0 but discouraged

For example, imaginary I can be also represented as: <cn type=constant> $ImaginaryI; </cn> and <cn type=constant>

During the normalization process, one way to handle these equivalences is to identify and transform the deprecated formats into the format that is introduced by MathML 2.0. For the purpose of this research, MathML documents in the repository will not include any discouraged representations from Content MathML 1.0

Log

The element log can be used as either a binary operator (taking logbase as a qualifier) or as a unary operator. If a logarithmic base is not provided, logbase 10 is considered as the default value. As a result, the following two Content MathML expressions are equivalent:

<apply> <log/> <logbase> <cn> 10 </cn> </logbase> <ci> x </ci> </apply>

and

<apply> <log/> <ci> x </ci> </apply>

When the user is searching for a log, the user can specify the logbase in the query that will be transformed into the corresponding structure pattern. If the user specified a logbase of 10 then the search will be for a pattern that includes that logbase. But if the document does not include logbase 10 because it is the default value, then that document will not be retrieved. This equivalence can be resolved during the content normalization process by simply adding the logbase 10 as a qualifier when log is used as a unary operator in mathematical documents.

Another equivalence issue that is related to the logarithm is the use of the natural logarithm ln. Content MathML offers an explicit unary operator <ln/> to represent the natural logarithm. $\ln(x)$ for example, has the following encoding:

<apply> <ln/> <ci>x</ci> </apply>

The same expression can be encoded implicitly by using the log operator as a binary operator with the exponential as the logbase. $\ln(x)$ is equivalent to $\log_e(x)$ and encoded as follows:

<apply> <log/> <logbase> <exponentiale/> </logbase> <ci>x</ci> </apply>

This equivalence will be handled the same way as the inverse of hyperbolic and trigonometric functions. Mathematical content that uses the implicit representation of the natural logarithm will be converted into the representation that used the ln operator. The user will be allowed to use both formats, and while processing the math query, a query that uses $\log_e(x)$ will be converted into the $\ln(x)$ format.

Sign

When searching for an integer 4 for example, the user can simply write +4 or just 4. In a document, an integer can be represented in two equivalent MathML forms

<cn>+4</cn>

and

<cn>4</cn>.

To deal with this equivalence on the content side, all real, integer, and rational numbers that do not have an explicit positive sign will be assigned a + sign. The user on the math query side will not be required to make the positive sign explicit. But during math query processing, an explicit sign will be added.

Encoding a negative number can cause an equivalence issue. For example, -2 can be represented applying the minus operator as a unary arithmetic operator thus creating the additive inverse of the number 2 or by simply adding a negative sign in front of the number.

<apply> <minus/> <cn>2</cn> </apply>

and

<cn>-2</cn>

Additionally, when encoding the expression $2 - 3$ for example, the number 3 can be encoded as the additive inverse of that number or can be encoded by simply applying the minus operator as a binary arithmetic (subtraction) on two positive numbers 2 and 3.

<apply> <plus/> <cn>2</cn> <apply> <minus/> <cn>3</cn> </apply> </apply>

and

<apply> <minus/> <cn>+2</cn> <cn>+3</cn> </apply>

In the canonical form, when minus is used as a binary operator, it will be considered as a representation of the subtraction operation and it will not be considered as the additive inverse of the second argument. In addition, any number where the minus is used as a unary operator to negate will be replaced by the same number representation with the negative sign added to it.

Function Composition

The composition of functions f and g, denoted $f \circ g$, can be represented as the application of the compose element to f and g. In addition, the same expression can be represented using a nested apply $f(g(x))$. Both representations are equivalent, and there needs to be a decision as to what MathML representation should be allowed in the normalized form while allowing the use of both representations in the math query language.

<apply> <apply> <compose/> <ci type="function"> f </ci> <ci type="function"> g </ci> </apply> <ci> x <ci/> </apply>

and

<apply> <ci type="function"> f </ci> <apply> <ci type="function"> g </ci> <ci> x </ci> <apply> </ apply>

The normalized form should support the use of the compose element and convert the nested application of functions into that format.

Additionally, the compose element in Content MathML is an n-ary operator, meaning, $fogoh$ can have two equivalent representations even when using the compose element. This issue is similar in nature to the issue of handling equivalences due to associative laws discussed earlier and will have a similar solution.

Integrals

Integrals can be made definite by specifying the domain of integration. These restrictions can be represented by specifying limits (upper and lower) on bound variables or by simply specifying an interval or a condition. The lowlimit and uplimit pair is used along with bound variables in Content MathML that correspond to this notation $\int_0^a f(x)$.

<apply> <int/> <bvar><ci>x</ci></bvar> <lowlimit><cn>0</cn></lowlimit> <uplimit><ci>a</ci></uplimit> <apply> <ci>f</ci> <ci>x</ci> </apply> </apply>

The same expression can be represented using an interval instead of limits $\int_{[0,\ a]} f(x)$ using the following Content MathML representation:

<apply> <int/> <interval> <cn>0</cn> <ci>a</ci> </interval> <apply> <ci type="fn">f</ci> <ci>x</ci> </apply> </apply>

The use of interval in the representation of an integral, as fas a search is concerned, can be viewed as a more general case of $\int_0^a f(x)$ since Content MathML does not require the use of a bound variable in that case. When querying, both syntaxes need to be allowed in the math query language. In the normalization process however, the canonical form needs to be restricted to one form over the other. In this case, the canonical form will be limited to the use of limits and any use of interval element will be converted into lowlimit and uplimit elements. This is possible because the type of closure in the case of intervals associated with integrals is considered irrelevant as far as search is concerned. This eliminates the need to create OR XQuery expressions that search for both forms.

Partial Differentiation

In Content MathML, the total order of differentiation can be specified using an optional degree element because each identifier used in the partial differentiation can have its own order specified. As a result, $\frac{\partial^2}{\partial x \partial y} f$ can have two equivalent representations: One with the total order of partial derivative specified and one without

<apply> <partialdiff/> <bvar><ci> x </ci><degree><cn> 1 </cn></degree></bvar> <bvar><ci> y </ci><degree><cn> 1 </cn></degree></bvar> <degree><cn> 2 </cn></degree> <ci type="function"> f </ci> </apply>

or

<apply> <partialdiff/> <bvar><ci> x </ci><degree><cn> 1 </cn></degree></bvar> <bvar><ci> y </ci><degree><cn> 1 </cn></degree></bvar> <ci type="function"> f </ci> </apply>

The total order of partial differentiation will remain optional in the canonical form because it can be inferred since each bound variable can have its own order specified, also the user will not query for the total order.

<apply> <partialdiff/> <bvar><ci> x </ci><degree><cn> 2 </cn></degree></bvar> <ci type="function"> f </ci> </apply>

or

<apply> <partialdiff/> <bvar><ci> x </ci></bvar> <bvar><ci> x </ci></bvar> <ci type="function"> f </ci> </apply>

This equivalence needs to be resolved on the canonical form where the author cannot declare that variable multiple times and instead the author is required to specify an order of the partial derivative associated with each identifier used even if it is of the first order. On the query side, however, the syntax of the query language will not allow the user to repeat an identifier, and if the user specified a variable without an order, the assumption is that the order of the partial derivative associated with that identifier is 1.

Additionally, order of the identifiers for the partial derivative is preserved using Content MathML. For example, $\frac{\partial^2}{\partial x \partial y} f$ is different from $\frac{\partial^2}{\partial y \partial x} f$. For the purposes of this research, the assumption is that order of partial differentiation is irrelevant as far as search is concerned. As a result, when the user writes **D_{x,y}(f)**, it is equivalent to **D_{y,x}(f)** query.

Matrices

Vectors are represented using a designated vector element. However, Content MathML regards vectors as equivalent to a single-column matrix where the transpose of a vector is equivalent to a single-row matrix. As a result, in documents, a vector [1, 2, 3] can be represented in three different equivalent formats: as a vector, as matrix with a single row, or as a matrix with a single column.

<vector> <cn> 1 </cn> <cn> 2 </cn> <cn> 3 </cn> </vector>

or

<matrix> <matrixrow> <cn> 1 </cn> <cn> 2 </cn> <cn> 3 </cn> </matrixrow> </matrix>

or

<matrix> <matrixrow><cn> 1 </cn></matrixrow> <matrixrow><cn> 2 </cn></matrixrow> <matrixrow><cn> 3 </cn></matrixrow> </matrix>

On the content side, however, because of matrix multiplication, the author should not be restricted to one representation of a vector and any notational equivalence issue needs to be resolved on the math query side through the creation of appropriate OR queries. On the math query side, however, there is not a designated syntax for a vector; the user can simply query for a vector by specifying a matrix with a single row or column. For example, the user query **matrix[a; b; c]** will generate an XPath expression looking for a vector OR a matrix with a single row OR a matrix with a single column with the elements 1, 2 and 3 according to that order.

Type attribute

The type attribute used in cn and ci elements indicates the type of the number or variable respectively. It is also used in the declare element when creating user defined variables or functions. The user is allowed to search for numbers and variables of specific types, but the authors are not required to set the type attribute in their cn and ci elements. These elements do have a default type value, *real* for cn and *empty string* for ci. One way to normalize these elements is to set their type value; however, this might interfere with the author's intent when not declaring a type. For example, an author can write <cn>2</cn> and intend this to be an integer. In this case, adding the default type value of *real* will be erroneous. Searching for variables or numbers with a specific type in the case of documents where the author did not specify a particular type is an issue. One solution could be to require the author to specify the type attribute associated with Content MathML elements. This can either be done directly in the case of the cn number or through the use of the declare elements where an identifier is associated with a particular type.

In addition, the attribute values for Content MathML elements, if present, should be normalized by removing surrounding white spaces. The assumption at this time is that authors do specify a value for the type attribute for cn and ci elements and that when attributes are present, their values are normalized.

Author-defined types and functions

Content MathML allows the authors of mathematical content to specify how an identifier is interpreted. This is done using the declare construct at the beginning of the document. The declare construct can be used in three ways: To associate a symbol with a set of attributes such as a type, to associate a symbol with an instance of an object such as a function or a vector, and finally a more complex way is to define a function using the declare construct in conjunction with a lambda construct. The fn construct that was used in version 1.0 to define a function has been deprecated and will not be considered in this research.

The first use of the declare construct, to set the attributes associated with a variable such as its type (real, cn, vector, function, etc.), has a single argument. For example, the variable X can be declared to represent a integer number as follows:

<declare type="integer"> <ci>X</ci> </declare>

As a result, all the occurrences of that identifier <ci>X</ci> within the scope of the declaration, will have their type attribute set to *integer* implicitly. Even though this simplifies things for the author because the

author does not have to set a type attribute every time the symbol X is used in the document, it causes a notational equivalence issue that is specific to the use of Content MathML and has an impact on how search is conducted. When searching, the user is allowed to specify a type when creating a search query for *an expression X=Y where X is integer*. The corresponding XML Query will be generated where a type is specified in the search pattern. But since the identifier X does not have a type attribute that is explicit, the document where X has a default type of integer will not be captured. This issue needs to be resolved on the content processing side where every instance of MathML elements with implicit attribute values, needs to be set. Referring to the previous example, when encountering the construct <ci>X</ci>, that construct needs to be transformed into <ci type="integer">X</ci> where the type attribute is added.

The second use of the declare construct, to associate a variable with an instance of mathematical object or a function definition, has two arguments: the variable and the constructor that is used to initialize that variable. For example, the following declare construct defines the identifier A as a vector with two components:

<declare type="vector"> <ci>A</ci> <vector> <cn>1</cn> <cn>2</cn> </vector> </declare>

When the variable is declared as a function with a specific number of arguments, its constructor (in this case, the function definition) is a MathML element that acts as a function. For example,

<declare type="function"> <ci>f</ci> <sin/> </declare>

which means any reference to the variable f means a sin function in that document.

Using the lambda construct in conjunction with declare is used to create more complex expressions. For example, $f(x)$ can be declared as $f(x) = x^2$ where f is associated with a single argument function ln.

<declare type="function" nargs="1"> <ci> f </ci> <lambda> <bvar><ci> x </ci></bvar> <apply> <ln/> <ci> x </ci> </apply> </lambda> </declare>

Documents where this declaration exists mean that $\ln(x)$ is equivalent to $f(x)$. Identification of these equivalences is a major research issue. The symbols declared at the beginning of mathematical content need to be identified during the content processing phase in such a way that when a user is searching for $\ln(x)$, documents where $f(x)$ is declared to represent $\ln(x)$ need to be retrieved since they are equivalent.

A related equivalence issue is when the user is searching for **y=x^2** in a document where $f(x) = x^2$ is declared and elsewhere in the document the expression $y = f(x)$ is used, that document needs to be retrieved. This issue needs to be resolved at the canonical level as well.

Another research issue is to enable the user to retrieve documents where functions are defined regardless of the identifiers used in its argument list. For example, when the user is searching for sin(x), documents where sin(y) is encoded need to be retrieved as well. There are two possible approaches to resolve this issue: through the math query language or during the query processing phase.

The math query language defines the $ symbol as a wild card that indicates any arbitrary single token. Referring back to the example, the user in this case needs to specify that he/she is searching for **sin($)**. Consequently, when the user is searching for sin(y), only documents where the identifier y is an argument will be retrieved.

During the query processing phase, the user search query **sin(y)** can be *relaxed* to include any arbitrary identifier. This will have a negative effect on the precision of search results if the user intended to search specifically for y, otherwise this approach will guarantee that the user gets all relevant results as *hits*.

In this research, the first approach will be adapted.

§ Content Transformation Module

The development of techniques that will transform math documents into the canonical form is currently a work in progress. The objective of the content transformation module is to transform user-authored content documents, where mathematics is encoded using Content MathML version 2.0, into a form according to specifications described so far.

As part of our research, the canonicalization of Content MathML will be complete to the extent of the specification of our user math query language. Our canonical form of Content MathML is valid Content Mathml.

§ Conclusion and Future Work

The canonical form of MathML addresses some of the challenges faced by the math search community. We hope that by defining what constitutes a canonical form of Content MathML and what extensions to the language are needed, this research could influence the direction of future development of MathML.

Bibliography

[AR, MO 06] Archambault, D. Moco, V., "Canonical MathML to Simplify Conversion of MathML to Braille Mathematical Notations", Journal title: LECTURE NOTES IN COMPUTER SCIENCE, 2006, NUMB 4061, pages 1191-1198, Publisher: SPRINGER-VERLAG

[CAP 99] Caprotti O. and D. Carlisle: 1999, "OpenMath and MathML: semantic markup for mathematics." In: Crossroads (the ACM Student Magazine). 6(2). ACM Press, New York.

[DL] NIST, "Digital Library of Mathematical Functions (DLMF)." http://dlmf.nist.gov/.

[MD] Design Science, "Mathdex". http://www.mathdex.com:8080/mathfind/search.

[MI, YO 03] Miller B. R. and A. Youssef: 2003, "Technical Aspects of the Digital Library of Mathematical Functions." Annals of Mathematics and Artificial Intelligence. 38(1-3): p. 121-136. Springer Netherlands.

[MILL 06] Miller B.: 2006, "DLMF, LaTeXML and some lessons learned." In: The Evolution of Mathematical Communication in the Age of Digital Libraries, IMA "Hot Topic" Workshop. http://www.ima.umn.edu/2006-2007/SW12.8-9.06/abstracts.html#Miller-Bruce

[MML 03] World Wide Web Consortium: 2003, "Mathematical Markup Language (MathML) Version 2.0." http://www.w3.org/TR/MathML2/.

[SAN 03] Sandhu P.: 2003, "the MathML Handbook". Charles River Media Inc., Massachusetts

[SK 82] Skemp R. R.: 1982, "Communicating Mathematics: Surface Structures and Deep Structures." In: Visible Languages Journal. 16(3). Sharon Helmer Poggenpohl.

[UMCL 06] http://inova.snv.jussieu.fr/umcl-demo/

[WIN 06] Winstead C.: 2006, "Creating Connexions Content Using LyX" module. Connexions Project, Rice University. http://cnx.org/content/m13238/latest/

[XP 05] World Wide Web Consortium: 2005, "XML Path Language (XPath) Version 2.0". http://www.w3.org/TR/xpath20/

[XQ 07] World Wide Web Consortium: 2007, "XQuery 1.0: An XML Query Language". http://www.w3.org/TR/xquery/

The Authors

Moody E. Altamimi
The George Washington University, Department of Computer Science
801 22nd Street NW
Washington
DC
United States
20052
tel: 2029947181
fax: 2029944875
maltamimi@gmail.com

Moody Altamimi is currently a doctoral candidate at the George Washington University, Department of Computer Science. She has a Masters of Science degree from The George Washington University in Software Engineering. Her research interests include information retrieval with focus on the search and retrieval of mathematical content, design and analysis of computer algorithms, XML-based technologies (MathML, XPath/XQuery), and object-oriented methodologies. She has also worked in industry with commerce-enabled Web technologies.

Abdou S. Youssef
The George Washington University, Department of Computer Science
801 22nd Street NW
Washington
DC
United States
20052
tel: 2029946569
fax: 2029944875
ayoussef@gwu.edu

Abdou Youssef received the B.S. degree in Mathematics from The Lebanese University, Lebanon, in 1981, the M.A. and Ph.D degrees in Computer Science from Princeton University, Princeton, NJ, in 1985 and 1988, respectively. He taught for a year in 1982 at the Institute of Applied Sciences, The Lebanese University. He has been with the Department of Computer Science Computer Science (formerly Department of Electrical Engineering and Computer Science) at The George Washington University, Washington DC, since 1987, serving as Assistant Professor from 1987 to 1993, Associate Professor starting from 1993 to 1999, and Professor since 1999. In 1994-95 he spent his Sabbatical at the National Institute of Science and Technology and, partly, at the Johns Hopkins University. He has published numerous papers in the areas of interconnection networks and computer architecture, parallel processing, fault tolerance, parallel algorithms, data compression, image processing, multimedia indexing, video processing and transmission, and watermarking. He co-edited a book titled "Interconnection Networks for High-Performance Parallel Computers", published by the IEEE Computer Society Press. He is the recipient of four Teacher of the Year Awards from the the Department and the School on Engineering and Applied Science at GWU. He is listed in the Who is Who Among America's Teachers. Professor Youssef is a senior member of IEEE, and a member of IEEE Computer Society and ACM.

Extreme Markup Languages 2007®
Montréal, Québec, August 7-10, 2007
This paper was formatted from XML source via XSL
by Mulberry Technologies, Inc.