A Public Converter for MathML, MP, and Infix

Xiao Zou Paul S. Wang
Email: {xzou,pwang}@mcs.kent.edu
Institue of Computational Mathematics
Deparment of Mathematics & Computer Science
Kent State University
Kent, OH 44242, U.S.A

Abstract

The public converter will translate mathematical expression from one encoding standard to another. This technical report will talk about several different encoding standards; compare their merits and defaults; and explain why we need a public converter component for these representations. Then we'll focus on how to build such a converter.

1 Background

Since 1990s, scientists have been focusing on how to publish mathematical documents and how to provide advanced math computation on Internet. Many works have been done in the past years and are still ongoing today. Clues collected from different aspects of research show that the first-of-all work is figuring out a reasonable way to encode various kinds of math expressions. That is Common Mathematic Representation.

MathML [1], MP [2], OpenMath [7] [8] are three main representations currently in using. We also treat Infix as a standard encoding method because people have been using Infix to write and publish math expression for too many years. Different encoding has different merits and defaults. They take different guiding rules in design and have different technical focus.

The request for a converter, which can translate math expression between different representations, is aroused in project IAMC (Internet Accessible Mathematical Computation) [9][10]. IAMC is an ongoing project in Kent State University. Its purpose is to provide an online mathematic computation service for Internet users. However, building a converter is not only the request of IAMC because it is not only IAMC that is contributing on online math computing service.

The key idea is that everyone must catch up with standard. But unfortunately, engineers are facing more than one choice. None of them can fully solve the questions of online computing. We'll introduce each of them as well as IAMC Framework, which will help us to address the questions, in order to explain why we need a converter.

MathML

In order to meet the diverse needs of the scientific community, MathML has been designed for encoding both mathematical notation and mathematical meaning.

MathML brings a hope to establishing a standard for rendering and handling math expressions in Internet browser or other client front-end. However, it is still not powerful enough to fit in the requirement of online computing. Presentation encoding may lead to misunderstanding of math expression under some circumstance. An example in sub-section Restriction on MathML gives some further explanation. Content encoding binds semantics of math notation with their syntax. But it is far from enough to encoding various kinds of math elements. Meanwhile, it is obvious that MathML encoding is very inefficient. An expression tree in MathML may be very large. So, current MathML is unsuitable to support advanced math computation in distributed environment.

MultiProtocol

MP is a mathematic data encoding/exchanging format that uses a binary annotated tree for the efficient exchange of data between scientifically oriented software tools. It defines encoding for a set of atomic types. [3] Constant and operators/functions also have correspondent definitions in MP libraries.

The most important advantage of MP is its efficiency, especially when we plan to transmit large amount of data from one place to another place. MP can also compress expression by eliminating duplicated subtree in MPTree. Another important characteristic is extensibility. We can create an expression tree to be self-defined to help math agent understand new data structures and operators/function prototypes.

MP doesn't provide rendering facility for math expression. And it has no a well-defined dictionary like OpenMath. However, its extensibility complements this default. In our opinion, extensibility is more important than dictionary in a scalable distributed environment. Maybe, we can never get a well-defined dictionary that contains everything we need.

OpenMath

OpenMath also defines a kind of method for Common Mathematical Representation. It tries to define semantic-rich tags for all kinds of math objects. This is an ambitious objective and its Content Dictionary limits the describing ability of engine for math objects. We know, in a distributed and scalable environment, one can never list out all possible function prototypes of engines because of the scalability.

OpenMath is designed for interoperability among math engines. However, the goal of online computation is integrating different math services to provide standard computing service. It doesn't require strong interoperability like OpenMath's estimate.

IAMC Framework

The IAMC framework aims to make it easy to connect and interoperate heterogeneous mathematical clients and servers, to support both interactive and transparent access to mathematical computation on the Internet/Web through a suit able protocol, and to provide customizable prototypes and libraries to make setting-up Internet-based mathematical services easy. [12]

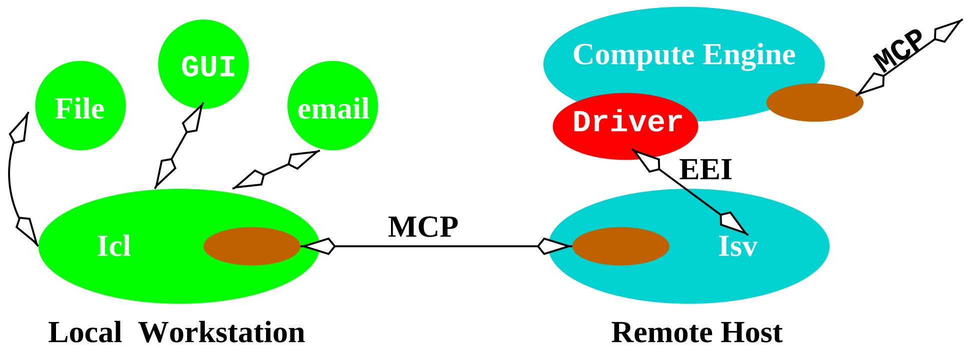

Figure 1: IAMC Architecture

As shown in Figure 1, IAMC framework has an open architecture. It defines an open service model and standard protocols between various kinds of Icl and Isv. It presents currently technical direction of online computing service for mathematic.

Users who wants to access advanced computing service need some fundamental support in their local PC. That is exactly the target of IAMC client (Icl). Icl is an end-user agent working on Internet for accessing services provided by IAMC server (Isv).

As MathML is becoming the rendering standard on Internet client, Icl must support MathML to be one of its official languages. However, it is not a language that most of compute engines like to support. Compute engines have their own languages - usually take Infix as basis - to let users express their computing requests. It is easy to understand the reason when we think about MathML's inefficiency, possible incorrect presentation, and inadequate of content markups to support multiple engines. Wolfram Research adds support for MathML into their newest version of Mathematica. But it is not full support [11].

Coming back to IAMC framework, if user wants to submit an expression from Internet browser to compute engine, the expression must be converted into the form that engine can recognize. If user wants to perform a continuous computation, expression must be somehow exchangeable between different compute engines. So, it is better that there is another public language being used among Icl, Isv, and engines. MP and OpenMath are possible selections. They are much more professional than MathML and they support interoperability among compute engines. Currently, we choose MP in IAMC framework for it is more flexible and scalable in distributed environment. This is the reason why we need a public MathML converter.

2 Usage

As Figure 2 shows, our goal is clear and simple. However, there are a lot of work needing our cautiousness because of the distinctions among different CMRs.

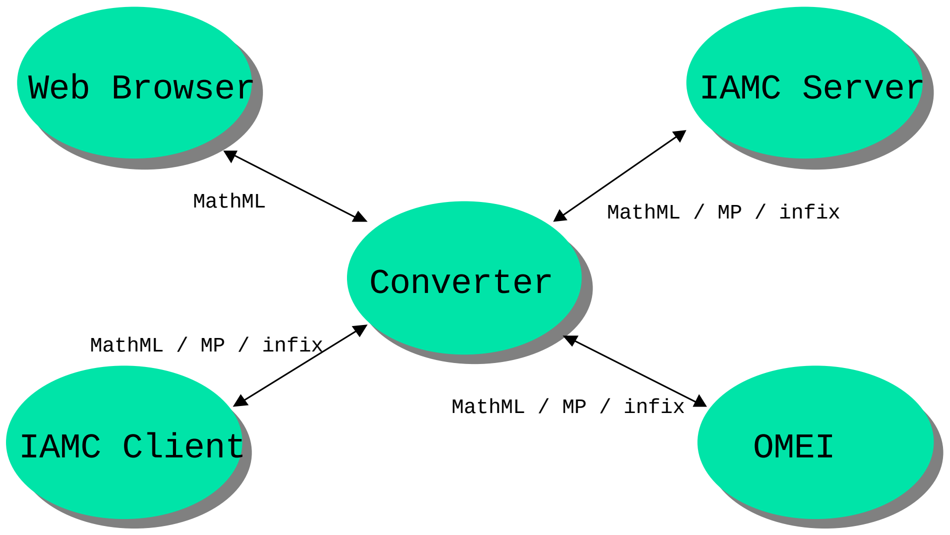

Figure 2: The Usage of MathML Converter

One possible Icl in IAMC framework is DragonFly and StarFish is an IAMC server we currently prefer. Other possible invokers include Web browser and Open Mathematic Engine Interface (OMEI). OMEI is responsible for converting expressions between public language in IAMC and special languages that engines use.

3 Solution

The technical routine of building converter is as following. This routine is also suitable for MathML-Infix conversion or MP-Infix conversion.

1. Parse MathML text into internal MathML Tree or parse MP packet into binary MP Tree;
2. Convert MathML Tree into MP Tree or vice versa;
3. Serialize MathML Tree or MP Tree into their original format.

Because the converter may be called from different platform, we use Java to implement it. It contains four packages: MathML package, MP package, Infix package, and Converter package. We do not explain Infix package here for it is too usual.

3.1 MathML Package

Objective

MathML package is responsible for parsing MathML text into a MathML tree, which can be converted into MP Tree or Infix tree. It is a MathML parser and it can also serialize MathML tree into normal MathML text.

One possible way is using some public XML parsers like Microsoft XML Parser to get parsed tree. However, there are several reasons that prevent us from adopting such suggestions.

First, not all MathML markups are meaningful for mathematical computation. For examples, ”mtext”, ”mglyph”, etc. These markups and the relative contents have to be eliminated before an expression can be submitted to compute engine because engine doesn’t intend to handle either help info or rendering format.

Second, public XML parser doesn’t care the semantic meaning of markups. They only work on the definition of DTD, which is only a syntactic declaration of MathML. However, converter must have an overview of the mathematical meaning of each part of MathML tree or it will fail to complete converting work.

Third, embedding a public XML tool into converter package will dramatically increase code size, which we must control carefully because converter may work as downloadable code.

So, using a public XML parser won’t reduce the time and code size for converting procedure. Moreover, we have to do some additional work to convert a parsed XML tree into an expression tree that engine can accept.

Object Model

Figure 3 shows the Design Object Model of MathML package.

From the view of notation, MathML can be categorized into three parts: Presentation markups, Content markups, and Entities.

Presentation encoding defines correspondent visual rendering rules and focuses on syntactic structure of mathematic notations. There are five kinds of presentation elements: token, general layout schemata, script and limit schemata, table and metrics, and enlivening expressions.

The goals of content encoding are combining syntax and semantics of mathematic notations and encoding expression tree structure. It contains seven subsets: containers, operators and functions, qualifiers, relations, conditions, semantic mappings, and constants and symbols.

Entities include special characters, constants, and some operators that are visible or invisible during rendering.

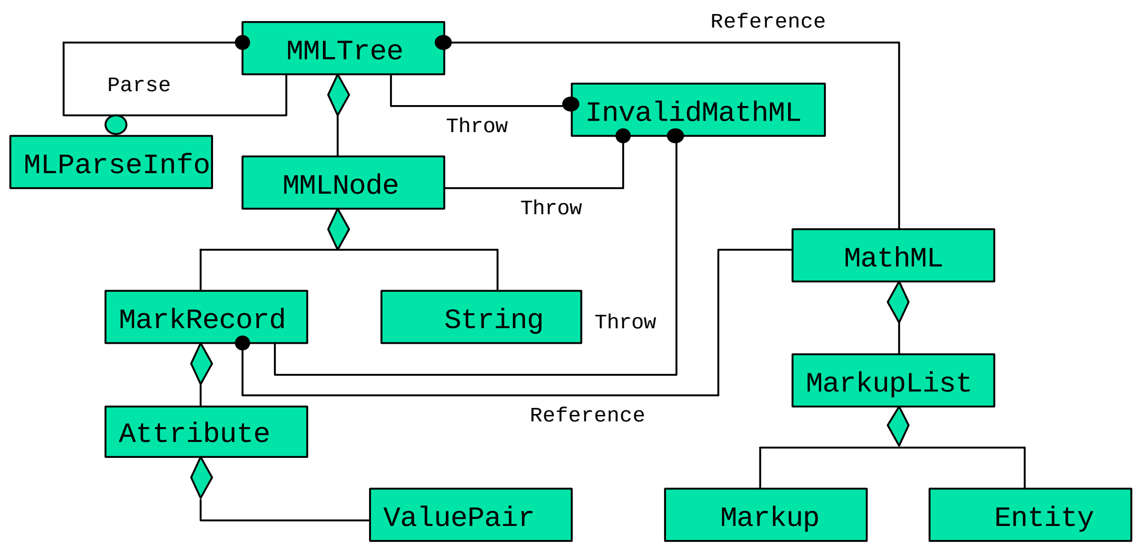

Figure 3: DOM of MathML Package

’MarkupList’ can be looked at as a dictionary that controls the semantics of markups. It also defines some important attributes for all markups. Both presentation tags and content tags are given out through ’Markup’ class. ’MMLTree’ presents the internal format for MathML text. However, not all MathML text, which is syntactically correct in XML, can successfully initialize ’MMLTree’. During parsing process, ’MMLTree’ will eliminate all rendering info and check whether it is a valid expression that is suitable for computation. If the check fails, ’MMLTree’ will throw an ’InvalidMathML’ exception.

Some entities have same mathematical meaning with correspondent content tags. For examples, ’&InvisibleTimes;’ and ’<times/>’, ’&ApplyFunction;’ and ’<apply>, ’&DifferentialD;’ and ’<diff/>’. Their math meanings are all controlled by ’MarupList’ and will influence converting result. This is also a problem that a public XML parser can’t resolve.

Many math constants are also defined using entities such as ’&pi;’, ’&ImaginaryI;’, ’&ExponentialE;’, etc. These elements must get recognized before followed converting work.

3.2 MP Package

MP is much more flexible and efficient than MathML or other XML-based encoding methods. Of course, it’s parsing process is more complex than MathML. Figure 4 shows the DOM of MP package.

’MPTree’ is what we’ll focus on. It is initialized using serialized ’MPTree’ that is a kind of binary encoding of expression. ’MPPrototypeTree’ has similar structure with ’MPTree’ but has no ’DataLimb’. Meta type can only exist in ’MPPrototypeTree’. ’MPDictionary’ maintains the content of Basic, Protocol, and Prototype dictionary of MP. It is compatible with its original C version but with more attributes. The detailed info about MP can be found in S. Gray’s doctor dissertation [6] and several publications. [3][4][5]

MP encoding has great difference with MathML encoding.

All data types in MP are in binary format. They are classified into basic types and regular types. Basic types include MP_Sint8, MP_Uint8, MP_Boolean, MP_CommonLatinId (ISO8859-1) [13], MP_CommonGreekId (ISO8859-7), MP_CommonMetaOperator, MP_CommonMetaType, MP_CommonOperator, and MP_CommonConstant. Regular types include MP_Sint32, MP_Uint32, MP_ArbInt (GMP 2.0)[14], MP_Real32, MP_Real64 (IEEE Standard)[15], MP_ArbReal (GMP 2.0)[14], MP_Identifier (ISO8859-1), MP_Constant, MP_String, MP_Raw, MP_MetaType, MP_MetaOperator, and MP_Operator.

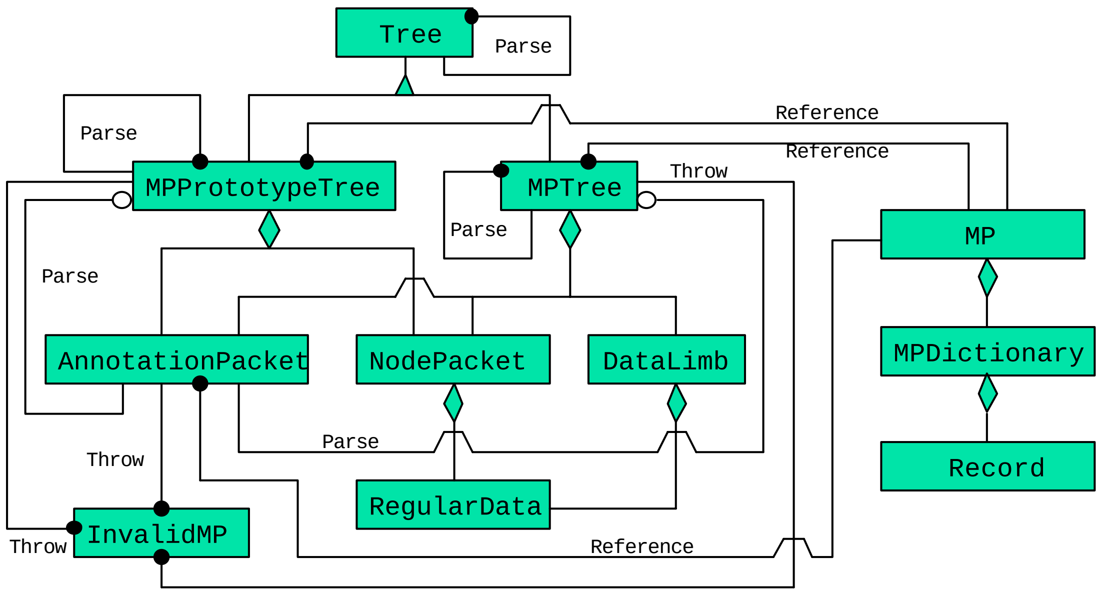

Figure 4: DOM of MP Package

So, MP can handle many kinds of data from machine precision number, boolean, string to arbitrary precision number and raw data. These data types obey IEEE standards and they can be adapted by compute engines easily and quickly.

MP also permits engineers to design complex data structures through declaring prototypes and utilizing several constructors. The use of prototype tree and data limb can greatly decrease the expansion of transmitting large data packets such as vector and matrix.

3.3 Converter Package

Object Model

Converter package takes MathML, MP, and Infix packages as basis. Infix-MathML and Infix-MP conversion are relatively easy. Here we only give out the DOM of MathML-MP converter. Figure 5 shows the detail.

Figure 5 is not a strict DOM graph because DragonFly, which acts as Icl, is out of the system boundary of converter's design. We put it here in order to show the easy interface and usage of MathML-MP converter.

'Converter' accepts 'MMLTree'/'MPTree' as input and returns a new 'MPTree'/'MMLTree'. Because both MathML package and MP package define dictionaries, mapping between different dictionaries is very important to guarantee correct conversion.

Technical Routine

To convert an expression from one encoding to another encoding, we must deal with three kinds of elements: Constant, Operator/Function, and Data Type/Data Structure.

1. Constant
   Many mathematical constants have special meaning. Some of them are impossible to be presented using their precise value such as Pi, Infinity, Exponential E, Golden Ratio, etc. Some of them can only be presented using symbol like Imaginary I, True, False, etc. Different encoding may have different

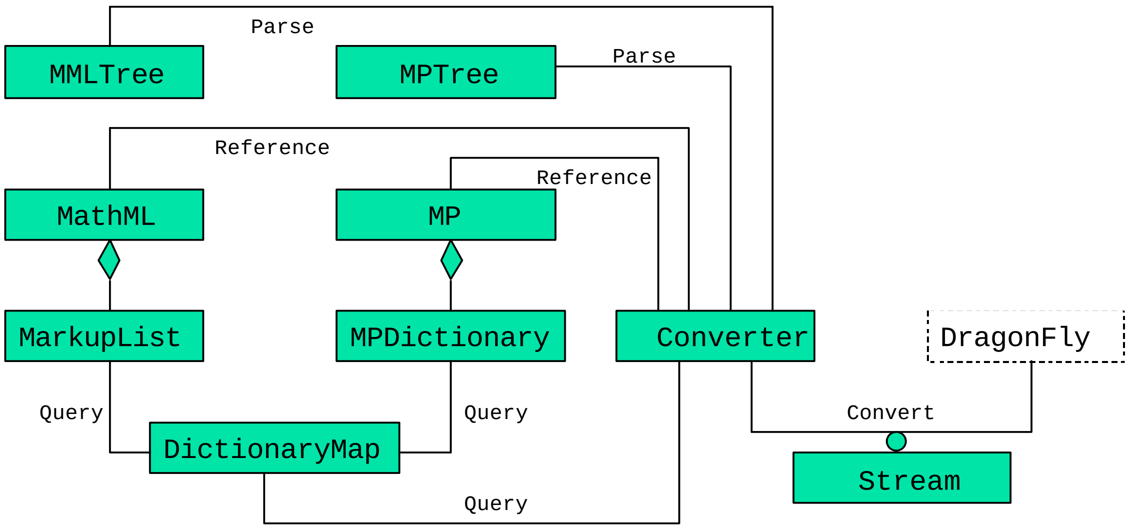

Figure 5: DOM of Converter Package

presentations to these special math elements. They must be correctly mapped between different encoding. If a constant can't find its counterpart in target encoding, converting work may fail and return error. However, there are some exceptions. Take Golden Ratio as example. If system fails to map it into target encoding, converter can use an approximate value like 1.61803 to replace it. Such possible approximate value is introduced through an attribute in our dictionary. User must keep it in mind that such kind of replacement is non-reversible. We mean converter can't replace number 1.61803 with constant Golden Ratio.

2. Operator/Function
   We process the conversion of functions in two ways.

   - Syntax only
     Only the syntax of function gets changed after conversion. This will happen if a function's prototype doesn't belong any encoding's dictionary (function is engine-oriented) or target encoding's dictionary at least. We know no dictionary can contain all possible math functions' declarations. So, Infix doesn't care function prototype, MP and MathML only define functions in common use, but they provide method to describe new functions. Converter needn't care about those functions' semantic. It is enough to change their syntax into appropriate format. Something needing more attention is that presentation encoding of MathML may lead to an invalid function or may have ambitious meaning. Such situation must get processed during MMLTree's initialization.

   - Semantic
     Converter must keep semantic of function consistent before and after conversion. The functions must belong to both source and target's dictionary. These functions are important because they are the bases to ensure that some types of expression, which are used most commonly, are exchangeable between engines. Correct name mapping is first-of-all condition. Meanwhile, converter may check the parameters' type and adjust parameters' sequence on need.

Operators are special functions that usually have semantic meaning. They are special because of their symbols and operands' position. They must be bound with formal function declarations in dictionary.

Then converting work is same as functions' after necessary adjustment.

3. Data Type/Data Structure
   Computer engines have strict definitions for data types and data structures. That is important for scientific computation. MP has its own definition for basic data types and it can describe all complicated data structures that math engines use because of its extensibility. However, Infix has no definitions for data types and data structures. It can only contain very simple data structure like vector in expression. Parser must recognize them according to some default rules. MathML has no strict definitions too. All its data are strings. But MMLTree can retrieve attributes '<base>' and '<type>' of content markups to correctly recognize data's type. MathML also provides several tags to construct mathematical object, for examples, '<set>', '<list>', '<union>', '<matrix>', '<interval>', etc. No matter which kind of encoding, its parser will treat the constructors of data structure as special operators. So, once an expression is successfully built into a native tree. We can automatically convert it into other formats by reference dictionary map.

4 Interface defition

Here we give the Interface definitions of MMLTree, MPTree, and Converter.

```
Interface Tree {
    int         NumOfSubtree;
    boolean     isRoot;
    Tree        parent;
    Tree[]      subtree;
    Tree        GetParent();
    int         GetNumOfSubtree();
    boolean     isLeaf();
    Tree        GetFirstChild();
    Tree        GetNextChild();
}

Interface MMLTree extends Tree {
    MMLNode     node;
    void        ParseMMLTree(String HText, int pos, MLParseInfo info);
    void        AddSubtree(MMLTree subTree, int index);
    void        DelSubTree(int index);
    String      Serialize();
    void        Serialize(OutputStream os);
}

Interface MPTree extends Tree {
    NodePacket         node;
    AnnotationPacket[] annotation;
    DataLimb           limb;
    int                ParseMPTree(byte[] stream, int pos);
    void               AddNode(NodePacket node);
    void               AddNode(MPProtortpeTree proto, MPTree[] pTree);
    void               AddSubTree(MPTree tree);
    byte[]             Serialize();
    void               Serialize(OutputStream os);
}

Interface Converter {
    void       Convert(InputStream is, OutputStream os, inf flag);
    void       Convert(MMLTree mltree, MPTree mptree);
    void       Convert(MPTree mptree, MMLTree mltree);
    void       Convert(InfixTree itree, MPTree mptree);
    void       Convert(MPTree mptree, InfixTree itree);
}
```

5 Converting Examples

Example 1: $\sin(x^2 + 1)$
Infix: `sin(x^2+1)`
MathML:

```
Presentation Encoding:
<mrow>
   <mi> sin </mi>
   <mo> &ApplyFunction; </mo>
   <mrow>
      <mi> x </mi>
      <mo> ^ </mo>
      <mn> 2 </mn>
      <mo> + </mo>
      <mn> 1 </mn>
   </mrow>
</mrow>

Content Encoding:
<apply>
   <sin/>
   <apply>
      <plus/>
      <apply>
         <power/>
         <ci> x </ci>
         <cn> 2 </cn>
      </apply>
      <cn> 1 </cn>
   </apply>
</apply>
```

MP:

```
    Type            Dictionary      Value        #Annot:Arg
    ---------------------------------------------------------
```

```
Cop             Elementary      sin             0:1
Cop             Basic           +               0:2
Cop             Elementary      power           0:2
MP_Identifier           x
MP_Uint8                2
MP_Uint8                1
```

Note: Total length is 6 bytes.

Example 2: Matrix $\begin{bmatrix} -1 & 2 & 0 \\ 3 & 1 & 6 \\ -5 & 2 & 9 \end{bmatrix}$

Infix: `((-1, 2, 0), (3, 1, 6), (-5, 2, 9))`

MathML:

```
Presentation Encoding:
<mtab>
  <mtr>
    <mtd> <mn> -1 </mn> </mtd>
    <mtd> <mn> 2 </mn> </mtd>
    <mtd> <mn> 0 </mn> </mtd>
  </mtr>
  <mtr>
    <mtd> <mn> 3 </mn> </mtd>
    <mtd> <mn> 1 </mn> </mtd>
    <mtd> <mn> 6 </mn> </mtd>
  </mtr>
  <mtr>
    <mtd> <mn> -5 </mn> </mtd>
    <mtd> <mn> 2 </mn> </mtd>
    <mtd> <mn> 9 </mn> </mtd>
  </mtr>
</mtab>

Content Encoding:
<matrix>
  <matrixrow>
    <cn> -1 </cn> <cn> 2 </cn> <cn> 0 </cn>
  </matrixrow>
  <matrixrow>
    <cn> 3 </cn> <cn> 1 </cn> <cn> 6 </cn>
  </matrixrow>
  <matrixrow>
    <cn> -5 </cn> <cn> 2 </cn> <cn> 9 </cn>
  </matrixrow>
</matrix>
```

MP:

```
Type            Dictionary      Value        #Annot:Arg
---------------------------------------------------------
Cop             Proto           Array           1:3
AP              Proto           Prototype
Cmop            Proto           Array           1:3
AP              Proto           Prototype
Cmt             Proto           IMP_Sint8
MP_Sint8               -1
MP_Sint8                2
MP_Sint8                0
MP_Sint8                3
MP_Sint8                1
MP_Sint8                6
MP_Sint8               -5
MP_Sint8                2
MP_Sint8                9
```

Note: Total length is 15 bytes.

6 Further Comprehension

There is something worth paying attention to when using MathML converter.

MathML Presentation Encoding

We support full conversion between MP and MathML content encoding and conversion from MP to MathML presentation encoding. However, we do not support conversion from MathML presentation to MP currently. Look at following example.

```
<msub> <mi>x</mi>  <mn>1</mn> </msub>
```

We can hardly distinguish that it is a variable x1 or the first array element x[1], or the first derivative of x with respect to the 1st variable.

We can easily find many similar encoding that may lead to misunderstanding of mathematical expression. Such encoding only works for rendering, not for computation.

Another example:

```
<mrow>
   <msubsup>
      <mo> &int; </mo>
      <mn> 0 </mn>
      <mn> 1 </mn>
   </msubsup>
   <mrow>
      <msup>
         <mi> &ExponentialE; </mi>
         <mi> x </mi>
      </msup>
```

```
         <mo> &InvisibleTimes; </mo>
         <mrow>
            <mo> &DifferentialD; </mo>
            <mi> x </mi>
         </mrow>
      </mrow>
   </mrow>
```

This is an example coming from the specification of MathML 2.0. Its rendering result will tell us it is expression $\int_0^1 e^x \, dx$. However, If we use standard parsing method for XML to parse this MathML expression into a tree, we'll find the parameters of Integrate "`<mo> &int; </mo>`" are not its sub nodes. From the view of expression tree, these parameters don't naturally belong to operator "`<mo> &int; </mo>`".

Based on these facts, current MathML converter doesn't support conversion from MathML presentation encoding to MP encoding.

Another important fact is that the conversion from MP to MathML won't generate rendering information no matter the target is presentation encoding or content encoding. The reason is very simple. MP support math computation well but it doesn't care about how to render the display of expression. So, there is no rendering info in MPTree. It only concern on the expression structure tree and its correspondent semantics. But Internet browser supporting MathML will know how to render the display of content encoding by default.

Reversibility Property

Here let us consider an interesting question. It will help us understand converter's internal working process better.

Let EXP be in MathML content code. We take a serial of steps as:

$$EXP \rightarrow EXP1(inMP) \rightarrow EXP2(inMathMLcontentencoding)$$

What will happen between EXP and EXP2?

In most cases, EXP2 keep its original form like EXP. Converter works as normal with no question. But there are several exceptions when EXP contains some special attributes or qualifiers.

Complex Data

Do reversal conversion to complex number 3+4*i*.

Before conversion:

```
   <cn type="complex"> 3 <sep/> 4 </cn>
```

After conversion:

```
   <apply>
      <plus/>
      <cn> 3 </cn>
      <apply>
         <times/>
         <cn> 4 </cn>
         <ci> &ImaginaryI; </ci>
      </apply>
   </apply>
```

Converter gets this result because MP doesn't define data type - *complex*. So, MP treats 3+4*i* as an expression, not common data.

Qualifier and Attribute
Take a look at an interval `(a, b]`.

```
   <interval closure="open-closed">
      <ci> a </ci>
      <ci> b </ci>
   </interval>
```

Current version of MP has no definition for interval structure and it can't handle attribute 'closure'. MP will recognize it as an operator/constructor and eliminate unknown attribute. But we can easily achieve correct conversion through extending MP's dictionary slightly.

Functional Operation
Compute $f \circ g(x)$
Before conversion:

```
   <apply>
      <apply><compose/>
         <fn><ci> f </ci></fn>
         <fn><ci> g </ci></fn>
      </apply>
      <ci> x </ci>
   </apply>
```

After conversion:

```
   <apply>
      <fn><ci> f </ci></fn>
      <apply>
         <fn><ci> g </ci></fn>
         <ci> x </ci>
      </apply>
   </apply>
```

The reason why converter gets this result is that MP can't deal with functional operation currently.

7 Discussion

MathML is current industrial standard for rendering mathematical expression in Internet browser. This is a fact that every engineer has to accept when they want to implement online computation for mathematic. It is defined by W3C Group and is getting more and more support from ad-hoc software vendors.

However, an actual standard doesn't mean it is a good standard. We have shown some crucial examples above, which indicate MathML is indeed not suitable for strict math computation. It is not surprise because

MathML is originally designed for rendering display and organizing documents. For online computation, its defaults are:

- Improperly mixing rendering rules with the tree structure of math expression in presentation encoding
- Having no enough content tags to bind semantics with math notations

MP is an existing protocol for math computation in distributed environment. It has pretty function scalability and good efficiency to transmit math objects among math agent and compute engines. But it is not a standard for Internet client.

Infix is current a most often used presentation to input mathematical expression because of the absence of visual edit tools for Internet client.

With all these reasons above, a MathML converter is necessary to support online mathematical computation. We can get a clear sight if we take this question into IAMC framework for consideration. During different stage of processing online computation, we need different encoding format to feed different requirements.

Converter's functionality is pretty simple. It is implemented as Java package and it can be linked into other components.

Some weaknesses of converter are:

- When converting MP encoding or Infix encoding to MathML encoding, we may not generate a MathML text with rich rendering info because neither MP nor Infix encoding cares rendering rules except expression structure
- Converter requires that computable MathML text must be written in good fashion to be a legal expression. We mean it should not cause confusion when it is built into expression tree.

In actual implementation, we didn't create MathML-Infix converting component. Conversion between MathML and Infix has to take routine "$MathML \leftrightarrow MP \leftrightarrow Infix$". It looks to be time-cost. Users will understand the reason when they think about the run-time environment of converter.

We have mentioned at the very begin of this article that different math engines may have different math describing languages of themselves. Expression must be converted into engine specified format to become executable. So, converter does provide service for these engines. Furthermore, keeping compatible with OpenMath in the coming future is also important to extend converter's usage. Having these in mind, we choose MP as transmitting media for all possible languages and encoding plans because of its extensibility and flexibility.

Suppose we have *n* encoding plans, $p_1 \cdots p_n$. Any converting action happening between p_i and p_j will take routine "$p_i \leftrightarrow MP \leftrightarrow p_j$". This rule greatly reduces the complexity of realization.

We should keep it in mind that this MathML converter is a special product for special requirements. It helps to ensure correctly conversation between Internet clients and compute engines. It is a very useful tool unless: MathML obeys strict constructing rules of mathematical expression tree while realizing rendering rules as accessorial attributes of math notations; visual editing tools are popular enough for Internet user. We may research on previous one as a functionality enhancement for MathML in the future. It can help MathML to be more professional for advanced online computation of mathematic.

References

[1] R. Ausbrooks, S. Buswell, S. Dalmas, S. Devitt, A. Diaz, R. Hunter, B. Smith, N.Soiffer, R.Sutor, S. Watt. Mathematical Markup Language (MathML) v.2.0. `http://www.w3.org/TR/2000/CR-MathML2-20001113`. Nov, 2000.

[2] S. Gray, N. Kajler and P. S. Wang. MP: A Protocol for Effcient Exchange of Mathematical Expressions. ISSAC'94. ACM Press, 1994.

[3] S. Gray, P. Wang, N. Kajler, Multi Protocol Specification, Version 1.2. July 13, 1997

[4] S. Gray, N. Kajler and P. S. Wang. Design and Implementation of MP, a Protocol for Effcient Exchange of Mathematical Expressions. J. of Symbolic Computation. Feb. 1998.

[5] S. Gray, L. Tong, and P. S. Wang. The MP Encoding for Distributed Mathematical Computations: An Object-oriented Design and Implementation. In Intl. Conf. on Parallel and Distributed Processing Techniques and Applications (PDPTA'99), Monte Carlo Resort, Las Vegas, Nevada, 1999.

[6] S. Gray, MP: A protocol for the efficient exchange of mathematical data, PhD. Thesis, Dept. of Math.and Computer Science, Kent State Univ., 1998.

[7] J. Abbott, A. Diaz. and R. S. Sutor. Report on OpenMath. ACM SIGSAM Bulletin, Mar. 1996.

[8] S. Dalmas and M. Gatano and S. Watt An OpenMath 1.0 Implementation 1997 Proceedings of ISSAC 97. ACM Press.

[9] P. S. Wang. Internet Accessible Mathematical Computation. 3rd Asian Symp. on Computer Mathematics (ASCM'98), Lanzhou Univ., China, 1998.

[10] P. S. Wang. Design and Protocol for Internet Accessible Mathematical Computation. ISSAC'99. ACM Press, 1999.

[11] MathML Intl. Conf. 2000, UIUC Illinois USA, `http://www.mathmlconference.org`, Oct. 20-21, 2000.

[12] W. Liao and P. S. Wang. Dragon y: A Java-based IAMC Client Prototype. 4th Asian Symp. on Computer Mathematics (ASCM'2000), 2000

[13] I.S.O ISO 8859-1:1987 Information processing - 8-bit single-byte coded graphic character sets - Part 1: Latin alphabet No. 1. International Standards Organization, 1987.

[14] T.Granlund. GNU MP: The GNU Multiple Precision Arithmetic Library, Edition 2.0. Technical report, The Free Software Foundation, April 1996.

[15] Institute of Electrical and Eclectronics Engineers. IEEE Standard for Binary Floating- Point Arithematic. ANSI/IEEE Standard 754-1985, August 1985.