Encoding Mathematics

CSC 2508

Professor Sevcik

Tuesday, April 25, 2000.

David Tam

Abstract

This report describes a concise encoding of mathematical formulas based on MathML. A significant reduction in file size can be achieved by using a simple encoding scheme of identifying and reducing all content tags to a length of 8 bits, along with the removal of obvious redundancies, and a final pass through gzip. The encoder can be viewed as a front-end preprocessor to gzip, enabling gzip to achieve higher compression ratios than in its absence. This paradigm of transforming data to suit gzip can be applied to many types of data files. The encoder itself is also fairly extensible, allowing the support of additional markup tags.

1 Introduction

This report briefly examines current methods of encoding mathematical formulas and attempts to develop a more concise encoding. The project proposal and approval forms can be found in Appendix A.

A fundamental issue in representing mathematical equations is determining whether to encode the presentation or content information. Presentation encoding describes how to render an equation, such as specifying the location of elements and their appearance. Content encoding describes the underlying semantics of an equation. It is usually more concise than presentation encoding. It offers the possibility of data reuse and expression evaluation since the underlying semantics are preserved. For example, the equation:

$$x = \frac{-b \pm \sqrt{b^2 - 4ac}}{2} a$$

can be presentation encoded by specifying each symbol, its location, and any attributes. Content encoding could specify the equation as:

```
x = numerator / denominator
denominator = 2 * a
numerator = plusminus(numerator2, numerator3)
numerator2 = -1 * b
numerator3 = sqrt(b * b  -  4 * a * c)
```

Content encoding preserves the most critical elements of an equation. It is not concerned with superficial details, which is an inherent problem of presentation encoding. Therefore, content encoding is very concise because it extracts only the essence of the equation. Rendering an equation for viewing can be accomplished by using default presentation attributes. In general, information becomes extremely valuable when computers able to

perform operations such as process, evaluate, compare, sort, search, and analyze. Content encoded equations offer this important property whereas presentation encoded equations do not.

1.1 Existing Encoding Methods

This section briefly describes a number of encoding schemes that are currently used.

Image-based:

This method simply encodes the equation as a bitmap image. Popular image formats include GIF and JPEG. This method is an example of presentation encoding at the extreme, where every pixel in a rendering of the equation is specified precisely. The underlying semantics are completely lost under this encoding scheme.[1] Currently, this method is popular on the web since the GIF and JPEG image formats are widely supported on web browsers.

ASCII Pictures:

This method uses the existing set of ASCII characters in a primitive attempt to pictorially describe an equation. Such a method is typically used in email messages when a sender needs to convey an equation to a recipient. Perhaps due to limited resources or for its simplicity, this method is chosen. Naturally, this scheme is an example of presentation encoding.

Macro Recording:

A possible method used by equation editors may be to record a user's menu selections and locations of text input during the process of building an equation. Such a scheme is an example of presentation encoding because most editors simply assist a user in the selection and placement of symbols. Rules concerning symbol usage are not enforced.

Unicode[26]:

In contrast to the ASCII standard of 8 bit characters, unicode typically uses 16 bits. Unicode allows for the encoding of alphabets of various languages. In the context of mathematical equations, having access to the Greek alphabet is particularly important. As well, there is enough capacity to encode various math symbols, such as integration, square root, intersection, and union. This scheme is an example of presentation encoding since restrictions on symbol usage are not enforced.

1 Optical character recognition may be applied in an attempt to partially recover the underlying semantics. Even if all the symbols are successfully recovered, the recognition of semantics still remains difficult.

Recursive Functions:

As hinted in the first example, math equations can be described in a recursive fashion. Such a schemed is an example of content encoding since the meaning of a function and the order of the arguments are well-defined.

Tex[28]:

Tex is typesetting language developed by Donald Knuth [20]. This method is popular among the academic and research community and is an example of presentation encoding. The presentation of an equation is described using Tex markup tags.

SGML ISO12083[3]:

Standard Generalized Markup Language is used mainly by the publishing industry. It is a very large and complicated specification that allows for precise control of all elements. It is also extensible, allowing users to define custom SGML tags. Due to its large size, a subset designated as the ISO12083 standard has been created to define a set of common math tags. These tags support both presentation and content encoding.

OpenMath[2][24]:

OpenMath is an example of content encoding. This standard was conceived in Europe by the Open Math Society. The standard rigorously defines the semantics of various tags and is based on XML [14][29][33][35].

MathML[36]:

MathML is also based on the XML standard and is rapidly gaining wide support among the internet community [11][17]. It will likely become the standard method of representing equations on the world wide web. The standard is currently under review by the World Wide Web Consortium. MathML allows for both presentation and content encoding, as well as a mixture of both types within one document. MathML support in web browsers and equation editors will be available soon [18][32][34]. Mathematical software companies such as Mathematica and Maple are also strongly supporting this standard [9][22][31]. MathML may become the universal standard for electronically describing mathematics.

2 Proposed Encoding Scheme

Since MathML is a promising standard that is widely supported, this report describes an encoding scheme based upon the MathML standard. Since content encoding offers many advantages over presentation encoding,

only the former will be considered. That is, only equations encoded using the content encoding tags of MathML will be considered. Examples of equations encoded using MathML content tags are given below.

1. $y = ax + b$

```
<apply>
      <eq/>
      <ci> y </ci>
      <apply/>
            <plus/>
            <apply>
                  <time/>
                  <ci> a </ci>
                  <ci> x </ci>
            </apply>
            <ci> b </ci>
      </apply>
</apply>
```

2. $\int_a^b \cos(x)\,dx$

```
<apply>
      <int/>
      <bvar>
            <ci> x </ci>
      </bvar>
      <interval>
            <ci> a </ci>
            <ci> b </ci>
      </interval>
      <apply>
            <cos/>
            <ci> x </ci>
      </apply>
</apply>
```

As evident in the examples above, MathML contains many redundancies. It appears that tags occupy the majority of the encoding. These redundancies are required to satisfy the grammar specifications of MathML. There are approximately 135 possible content tags, covering all math concepts ranging from kindergarten to the end of second year university. Areas covered include arithmetic, algebra, logic, relations, calculus, vector calculus, set theory, sequences, series, trigonometry, statistics, and linear algebra. As well, MathML is extensible to allow user-defined tags. This mechanism should allow for other areas mathematics to be supported.

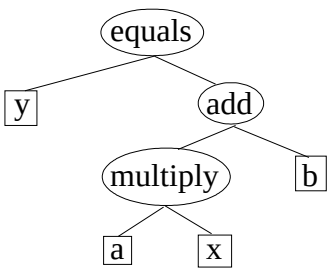
3 Discrete Information Theory Analysis

This section will attempt to define a guideline for estimating the entropy of math

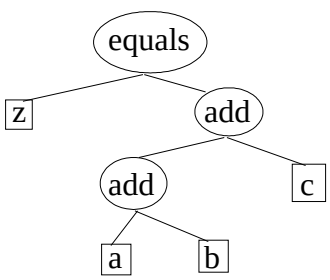
equations. That is, how to determine H(equation).

Math equations can be represented in an expression tree structure. Each node is further defined by its two children. This tree structure organization leads to a fairly recursive pattern of function calls. Leaves of the expression tree represent variables, such as x, y, π, and 42. Inner nodes represent functions, such as integral, exponent, square root, addition, and subtraction. The basic unit of evaluation appears in the form of operation(arg1, arg2). This basic unit can be built upon to represent any equation. The rules of the construction are fairly simple: arg1 and arg2 may be replaced with another basic unit of evaluation. The following two examples illustrate these concepts.

1. y = ax + b ==> equals (y, add (multiply (a , x) , b))



2. z = a + b + c ==> equals (z , add (add (a , b) , c))



With some minor re-organization, an equation can be viewed as a sequence of messages in the form: operation(arg1, arg2). Determining the entropy of a math equation can be reduced to determining the entropy of a message: H(message), which is independent of the length of the equation. This provides a common unit of evaluation since the number of messages in an equation can vary due to varying equation lengths.

H(message) = H(operation, arg1, arg2)
H(operation, arg1, arg2) = H(operation) + H(arg1) + H(arg2)

IFF operation, arg1, arg2 are independent

Since only a rough guideline is desired, the random variables `operation', `arg1', and `arg2' may be assumed to be independent. If they are not independent, the resulting value will be larger than the true value. Under these simplifying assumptions, all that is needed are the frequencies of `operation' and `arg' values. Due to the recursive nature of the messages, it is also important to determine the frequency that an `arg' is an `operation'.

3.1 Estimating Entropy in MathML Encoding

The entropy of content-based MathML encoding is determined by the frequency of content tags and the frequency of identifiers, numbers, and symbols. In this report, the latter three will be grouped together as one and referred to as identifiers. These frequencies were gathered from a testbed assembled by aggregating all content encoding examples from the World Wide Web Consortium MathML 2.0 Working Draft web site [36]. These examples totalled to 1,762 lines of MathML code (24,827 bytes). See Appendix B for the example code. This testbed was chosen because existing example of MathML code were scarce. Ideally, entire math textbooks encoded in MathML would offer more representative samples. However, these sources do not exist and encoding even a few chapters would have been quite time consuming. The examples of the testbed file cover the entire spectrum of possible content tags. The following approximate distributions of the testbed file are shown below.

Approximate Distributions:

Identifier	Frequency	Tag	Frequency
x	140	<ci>	434
a	58	</ci>	434
b	43	<apply>	307
f	30	</apply>	307
y	25	<cn>	116
2	23	</cn>	116
A	22	<bvar>	60
0	21	</bar>	60
1	17	<plus/>	32
3	18	<fn>	30
B	16	</fn>	30
etc...	etc...	etc...	etc...
	===		===
	556 total	+	2459 total ==> 3015 messages

Based on these distributions, the entropy of the tags was approximately 3.5 bits and the entropy of the identifiers was approximately 2.6 bits. A message is defined as one tag or identifier. For a rough approximation, if independence between tags and identifiers is assumed, then the entropy of a message is approximately 6.1 bits. Except for a few special cases[2], tags and identifiers should be mostly independent since knowing the outcome of

2 Some special cases of high correlation include: (1) the "log" tag and the identifiers "10" and "e"; (2) the "sqrt" tag and the identifier "2".

a tag should provide little knowledge about the up coming identifier and vice versa. As indicated in the above distributions, the testbed file consists of 3015 messages. With an entropy of 6.1 bits per message, this leads to a theoretical minimum file size of approximately 18,392 bits (2,299 bytes). Since independence was assumed in a number of situations, this value is most likely higher than the true minimum file size. A number of short C programs were written to aid in these calculations. The source code for these programs can be found in Appendix C. The "cicncsymbol.c" code was used to obtain identifier frequencies. Obtaining tag frequencies was accomplished in the "simpleencode.c" code found in Appendix G.

4 Details of Encoding

A number of attempts were made to encode content-based MathML in a more concise manner. The task of encoding equations can be viewed as an exercise in encoding specialized text. Rather than re-invent the wheel, in all trials gzip was applied on top of the encoding to further compress the testbed file, since gzip is known to be a very good compressor.

4.1 Trial #1

Since there are approximately 135 content tags, each of these tag can be encoded in an 8-bit character. This scheme requires a fixed dictionary (927 bytes) of known tags and their 8-bit encodings. The dictionary can be found in Appendix D. If a tag is unknown, it is left alone and encoded in plain ASCII characters. Unknown tags seldom occurred in the testbed file but in general, they are a likely possibility since other XML tags may be embedded in the MathML data. A number of obvious redundancies were removed, such as carriage returns, and unnecessary tabs and spaces. The source code is incorporated into the source code for trial 3, which can be found in Appendix G.

This encoding produces a mixed stream of regular 8 bit ASCII characters and 8 bit content tag encodings. Since the tag encodings overlap with regular ASCII encodings, the "<" character of the tags were preserved to indicate that the next 8-bit character should be interpreted as a content tag encoding. As an example, the following MathML code and its new encoding are shown below.

Original MathML code:

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

New MathML encoding:

```
<a<p<cx</c<cy</c</a
```

With a total of 2,459 tags and an average tag length of 7.2 bytes, an average savings of 5.2 bytes per tag can be achieved. This meant an estimated savings of 12,787 bytes. A simple program was written to aid in this calculation. The source code can be found in Appendix E.

4.2 Trial #2

Huffman encoding of the content tags was attempted in trial 2. The objective was to achieve a new average tag length of less than 8 bits in order to perform better than trial 1. Using the frequencies of the tags in the testbed file, a Huffman encoding was manually generated and fed into the encoding program. Due to the variable length nature of Huffman codes, the mixed stream of tag encodings and regular ASCII character encodings were bit-packed. This packing resulted in ASCII characters that were no longer aligned on 8 bit boundaries. The source code for this program and the Huffman encoding dictionary can be found in Appendix F.

4.3 Trial #3

Further optimizations on the first trial were attempted. The source code can be found in Appendix G. The encoding of end tags such as </apply>, </ci>, </cn> could be encoded as a generic end tag. This simplification is possible due to the regular grammar structure of MathML. If a end tag is encountered, it must match the last encountered start tag. An additional optimization reduced the number of "<" characters by placing delimiting characters such as "<" and ">" in strategic locations. Since a large number of the tags occur in sequence before a regular ASCII character is encountered, delimiting characters were placed around the regular ASCII characters instead. For example:

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

would normally be encoded as:

```
<a<p<cx</c<cy</c</a
```

Under the new optimization, it is encoded as:

```
<apc>x</c>y<ca
```

The "<" and ">" characters can be viewed as delimiter symbols that mark the beginning and end of tag encodings.

5 Results of Encoding

The new encodings were successful in reducing the redundancies of MathML. Compression using only gzip and unix compress were also compared against the new encodings. Gzip was used with the "--best" flag to indicate the use of the best possible compression at the expense of speed. The table below presents the resulting file sizes. It also presents results from a larger file size that was generated by simply replicating the contents of the testbed file 100 times.

Original Testbed File: (file sizes are in bytes)

plain text	24,827				
compress	6,611				
gzip	3,639				
trial#1	8,747	trial#2	6,179	trial #3	6,271
#1+compress	3,500	#2+compress	4,888	#3+compress	2,939
#1+gzip	2,263	#2+gzip	3,820	#3+gzip	1,990

Testbed File X 100: (file sizes are in bytes)

plain text	2,482,700				
compress	288,791				
gzip	21,923				
trial #1	874,700	trial #2	619,775	trial #3	627,100
#1+compress	125,089	#2+compress	193,013	#3+compress	99,781
#1+gzip	8,419	#2+gzip	16,210	#3+gzip	6,160

With the original testbed file, trials 1, 2, and 3 were able to achieve a size reduction of 65%, 75%, and 75%, respectively. Gzip by itself is also a very effective compressor, obtaining a reduction of 85%. However, by applying the new encoding as an initial step followed by gzip, the gzip algorithm was able to reduce the original gzip file by 38% to 45%. Trial 2 on the original testbed file was an exception to the rule, where the final file size was slightly larger than the gzip version. The encoder can be viewed as a front-end preprocessor to gzip that transforms the input file to a more suitable format for gzip's compression algorithms. Compared to the original file, an encoder preprocessed gzip file reduced the original testbed file by 91%, 85%, and 92%, respectively. For the larger testbed file, this preprocessor enabled the original gzip file to be reduced by 62%, 26%, and 72%, respectively. Results using unix compress produced results that show similar general trends.

6 Discussion of Results

The theoretical minimum file size approximated in the analysis section was fairly accurate. It predicted a minimum file size of approximately 2,299 bytes. However, trial 3 reduced the file to a size of 1,990 bytes. As mentioned previously, since only rough approximations were desired and

independence was assumed for the sake of simplicity, the predicted minimum file size should be slightly higher than the true minimum value. Trial 3 combined with gzip should be considered very compact due to the drastic reduction in file size (92%) and therefore, should be fairly close to the theoretical minimum file file.

The estimation of savings in trial 1 were also fairly accurate. Savings of 12,787 bytes were estimated while the results indicated a value of 16,080 bytes instead.

Verification of correctness for trials 1 and 3 was accomplished by simply inspecting the encoded output file. Although writing a decoder would offer a stronger verification process, trials 1 and 3 were fairly simple to just visually inspect. Trial 2 proved to be difficult for visual inspection due to the unalignment phenomenon. To determine correctness, the estimated savings was calculated. Based on the frequency distribution of tags and the Huffman encoding (both shown in Appendix F), the following table was generated.

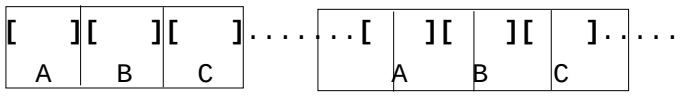
Tag	Count	Huffman Length	Savings
known end tags	1074	1 bit	7 bits
ci	434	3 bits	5 bits
apply	307	3 bits	5 bits
cn	116	5 bits	3 bits
known w/attr	75	5 bits	3 bits
bvar	60	6 bits	2 bits
			------
	approximate savings =		11,916 bits
		=	1,490 bytes

The approximate savings compared against trial 1 is 1,490 bytes. An actual savings of 1,476 bytes was realized, which is quite close to the estimated value. From these results, the algorithm developed in trial 2 is most likely correct.

Surprisingly, the simple encoding scheme developed in trial 1 and further refined in trial 3 proved to be very effective. When combined with gzip, file size reductions were quite substantial. In contrast, the more sophisticated Huffman encoding in trial 2 combined with gzip did not produce the best results. This result is likely due to the unaligned bit boundaries that resulted. Character frequencies were disturbed and gzip was unable to apply its algorithms effectively. The encoding of trial 2 resembled a binary file since true sequences of ASCII characters were unaligned to the character bit boundaries. In contrast, trials 1 and 3 preserved the 8 bit character boundaries and allowed the encoding to appear more like a text file.

Gzip is a byte-oriented compression algorithm [4][5][6][10][16]. First, it applies Lempel-Ziv 77 adaptive encoding, replacing duplicate strings with a (distance, length) pair. It has a maximum distance of 32k bytes, and a

maximum length of 258 bytes. Next, it uses Huffman encoding. Two Huffman trees are created. One is used to encode literals and match lengths, while another is used to store distances. The effectiveness of the first stage depends on the distribution of characters from the preprocessor. The correlation of characters is also important since the existence of common sequences of characters is vital to the effectiveness of the LZ77 algorithm. In trial 2, most of the correlation was not detected due to the unpredictable unalignments that occurred. The example below illustrates a sequence of characters (ABC) that are aligned differently on each instance. LZ77 would detect the sequence in the first instance, while in the second instance, it would appear as a sequence of totally different characters.



The simple encoding schemes in trials 1 and 3 assisted gzip in a number of ways. Since gzip has a maximum history of 32k bytes, the simple encoding can be viewed as helping to increase this history size. Tags are shortened from an average of 7.2 characters to approximately 1 character, helping to fit more characters into the 32k window. In trial 3, known end tags were aggregated into a single generic end tag since the matching start tag could be easily determined. This optimization increases the possibility of finding common character sequences. The example below shows how different sequences of end tags may become common sequences.

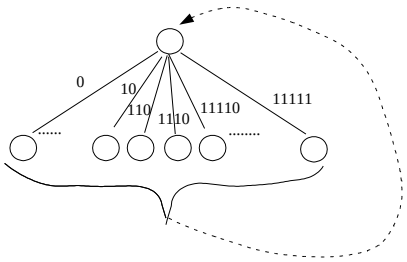
Original:
```
...</ci></apply>...</csymbol></apply>...
```

Trial 1:
```
...</c</a...</s</a...
```

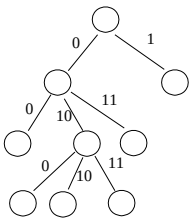
Trial 3:
```
...</</...</</...
```

A grammar-based encoding scheme was not attempted due to the large grammar of MathML. As well, some grammars might not undergo effective grammar encoding. Flat grammars, which contain a large number of fan-ins and fan-outs for each state, may exhibit poor compression. In contrast, a highly hierarchical grammar reduces the number of possible next states. It reduces the fan-in of a state as well. MathML is considered a flat grammar language because it has fairly little restriction on which state may be entered from any other state. For instance, once the <apply> state is reached, there is little restriction on the next possible tag. Due to this situation, grammar encoding becomes similar to traditional Huffman encoding. The benefit of grammar encoding is that at any particular state, the number of next possible states is small and can be encoded in less bits. As well, local probabilities in the form of P(next state|current state) are

used rather than global probabilities required in the traditional Huffman model. In grammar encoding, incremental costs are paid to transition from the current state to the next state. However, in flat grammars the benefit of incremental costs are not present. The example below illustrates flat and hierarchical grammars.



Flat Grammar



Hierarchical Grammar

The simple encoding developed in trial 1 and 3 can be applied to other applications of XML. It can be easily extended to support the presentation tags of MathML. The concepts can be applied to SGML in general, HTML, OpenMath, and other markup languages.

The encoding concepts described in this report may be extendible to equation derivations and mathematical proofs. Due to the usually high amount of redundancy between one line in a proof and the subsequent line, only minor changes in the expression tree are required. If an equation is viewed as an expression tree, substitutions of one part of the tree (subtree) can be specified.

An issue that was not addressed in the experiments was the mixture of MathML with regular text descriptions and possibly HTML tags. Normally, mathematical content does not exist by itself. It usually requires textual descriptions and annotations to clarify the mathematics. This mixture will be common on web pages in the near future since MathML is rapidly being accepted among the internet community. Since the encoding scheme is fairly flexible and extensible, this mixture should not be a major issue and can be accomodated. Regular text would be well-compressed under the LZ77 algorithm in the gzip stage.

7 Related Work

A good survey on the state of mathematics and computer technology can be found in [27]. Compression of other forms of data has been attempted. These include Java class files [19][25], various forms of semi-structured text [23][37], and fat binary executables [8][12][13]. These

works have focused mainly on grammar-based or tree syntax-based encoding methods. This report dealt only with MathML content tags while ignoring presentation tags. Reconciling between MathML content and presentation tags is discussed in [7]. Mathematical computation protocols are presented in [15] and [30] to handle distributed computation. The Emath component is described in [1]. It is an embeddable component based on Lisp that supports highly customizable display and editing of math formulas. The integration of MathML, OpenMath, and XML in general, is discussed in [21]. Information on the XML standard, its possible applications, and XML grammar specifications can be found in [14][29][33] [35].

8 Conclusions

The simple encoding scheme developed for MathML proves to be very efficient, achieving very high compression rates when used in conjunction with gzip. The encoding scheme can be easily applied to other markup languages. The encoder enabled gzip to realize further compression. The encoding scheme can be viewed as a special front-end preprocessor to gzip that transforms MathML input into a more compressible and suitable format for gzip. This preprocessor idea can be applied to other areas of specialized content.

References

[1] O. Arsac, S. Dalmas, M. Gaetano, "The Design of Customizable Component to Display and Edit Formulas," ACM Proceedings of the 1999 International Symposium on Symbolic and Algebraic Computation, July 28 - 31, 1999, Vancouver, Canada, pp. 283-290.

[2] O. Caprotti and D. Carlisle, "OpenMath and MathML: Semantic Mark Up for Mathematics," ACM Crossroads, January 2000, http://www.acm.org/crossroads/xrds6-2/openmath.html.

[3] R. Cover, "General SGML/XML Applications," The XML Cover Pages, Organization for the Advancement of Structured Information Standards, February 17, 2000, http://www.oasis-open.org/cover/gen-apps.html.

[4] P. Deutsch, "DEFLATE Compressed Data Format Specification Version 1.3," Network Working Group, RFC 1951, May 1996.

[5] P. Deutsch, "GZIP File Format Specification Version 4.3," Network Working Group, RFC 1952, May 1996.

[6] P. Deutsch and J-L. Gailly, "ZLIB Compressed Data Format Specification Version 3.3," Network Working Group, RFC 1950, May 1996.

[7] S.S. Dooley, "Coordinating Mathematical Content and Presentation Markup in Interactive Mathematical Documents," ACM Proceedings of the 1998 International Symposium on Symbolic and Algebraic Computation, August 13-15, 1998, Rostock, Germany, pp. 54-61.

[8] J. Ernst, W. Evans, C.W. Fraser, S. Lucco, T.A. Proebsting, "Code Compression," ACM SIGPLAN Conference on Programming Language Design and Implementation, June 16-18, 1997, Las Vegas, NV, pp. 358-365.

[9] K. Esplin, "W3C Issues MathML as a Recommendation - Testimonials," http://lists.w3.org/Archives/Public/w3c-news/1998AprJun/0002.html.

[10] A. Feldspar, "An Explanation of the Deflate Algorithm" August 23, 1997, http://www.cdrom.com/pub/infozip/zlib/feldspar.html.

[11] K.R. Foster, "Math on the Internet," IEEE Spectrum, Vol. 34, No. 4, April 1999, pp. 36-40.

[12] M. Franz and T. Kistler, "Slim Binaries," Communications of the ACM, December, 1997, Vol. 40, No. 12, pp. 87-94.

[13] M. Franz and T. Kistler, "Slim Binaries," Technical Report , University of California at Irvine, Department of Information and Computer Science, 96-24.

[14] L.M. Garshol, "BNF and EBNF: What are they and how do they work?" http://www.stud.ifi.uio.no/~lmariusg/download/artikler/bnf.html.

[15] S. Gray, N. Kajler, P. Wang, "MP: A Protocol for Efficient Exchange of Mathematical Expressions," ACM Proceedings of the International Symposium on Symbolic and Algebraic Computation, July 20-22, 1994, Oxford, England, pp. 330-335.

[16] "The gzip home page," http://www.gzip.org/.

[17] M. Hagler, "Mathematics and Equations on the WWW," 28th Annual Frontiers in Education Conference, November 4-7, 1998, pp. 583-586.

[18] ICE Soft AS, "e-Lite - The 100% Pure Java Web Browser," http://www.icesoft.no/ELite/.

[19] T. Kistler, M. Franz, "A Tree-Based Alternative to Java Byte-Codes," February 1999, http://www.ics.uci.edu/~franz/publications/TreeBasedAlternativePrepub.pdf.

[20] D. Knuth, "Don Knuth's Home Page," http://Sunburn.Stanford.EDU/~knuth/.

[21] X. Li, "XML and the Communication of Mathematical Objects," M.Sc.

Thesis, The University of Western Ontario, April 1999.

[22]Mathematica Home Page, http://www.wolfram.com/.

[23]C.G. Nevill-Manning, I.H. Witten, D.R. Olsen Jr., "Compressing Semi-Structured Text Using Hierarchical Phrase Identifications," IEEE Proceedings of the Data Compression Conference, March 31 - April 3, 1996, pp. 63-72.

[24]The OpenMath Society, http://www.openmath.org/omsoc/index.html.

[25]W. Pugh, "Compressing Java Class Files," Proceedings of the ACM SIGPLAN'99 conference on Programming language design and implementation, April 1 - 4, 1999, Atlanta, Georgia, pp. 247 - 258.

[26]M. Sargent III, "Unicode Plain-Text Encoding of Mathematics," Microsoft Corporation, Redmond, WA, August 23, 1994.

[27]R.M. Timoney, "Style Sheet Languages and Mathematical Material," http://www.botik.ru/PSI/EmNet_NIS/transactions/timoney/timoney.html.

[28]TeX Users Group, "TeX Users Group Home Page," http://www.tug.org/.

[29]T. Usdin and T. Graham, "XML: Not a Silver Bullet, But a Great Pipe Wrench," ACM StandardView, Vol. 6, Issue 3, 1998, pp. 125-132.

[30]P.S. Wang, "Design and Protocol for Internet Accessible Mathematical Computation," ACM Proceedings of the 2nd International Symposium on Parallel Symbolic Computation, July 20 - 22, 1997, pp. 188-195.

[31]Waterloo Maple Home Page, http://www.maplesoft.com/corporate/press/press_010.html.

[32]WebEQ Home Page, http://www.webeq.com/index.html.

[33]A. Wess, "XML Gets Down to Business," ACM netWorker: The Craft of Network Computing, Vol. 3, Issue 3, 1999, pp. 36-43.

[34]World Wide Web Consortium, "Amaya - W3C's Editor/Browser," W3C User Interface Domain, http://www.w3.org/Amaya/.

[35]World Wide Web Consortium, "Extensible Markup Language (XML)," W3C Architecture Domain, http://www.w3.org/XML/.

[36]World Wide Web Consortium, "Mathematical Markup Language (MathML) Version 2.0," W3C Working Draft, March 28, 2000. http://www.w3.org/TR/MathML2/.

[37]E.-H. Yang and J.C. Kieffer, "Universal Source Coding Theory Based on Grammar Transformations, " IEEE Proceedings of the Information Theory and Communications Workshop, June 20-25, 1999, Kruger National Park, South Africa, pp. 75-77.

Appendix A

-- Project Proposal and Approval Forms --

Appendix B

-- MathML Testbed File --

Appendix C

-- Entropy Calculation Programs --

Appendix D

-- Fixed Dictionary of Trial #1 --

Appendix E

-- Average Tag Length Calculation Program --

Appendix F

-- Trial #2, Huffman Encoding Source Code,
Huffman Encoding Dictionary,
Tag Frequency Distribution --

Appendix G

-- Trial #3, Source Code --