EIDGENÖSSISCHE TECHNISCHE HOCHSCHULE LAUSANNE
POLITECNICO FEDERALE DI LOSANNA
SWISS FEDERAL INSTITUTE OF TECHNOLOGY LAUSANNE

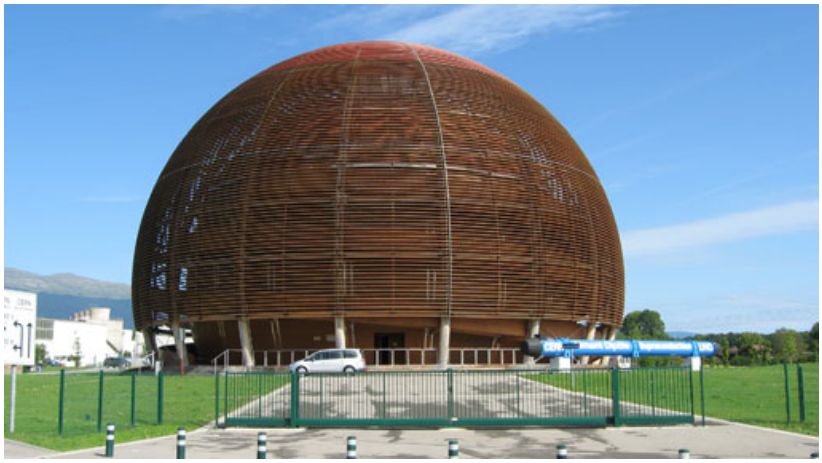
School of Computer and Communication Sciences
Computer Science Section | Distributed Systems Laboratory (LSIR)

ÉCOLE POLYTECHNIQUE
FÉDÉRALE DE LAUSANNE

CERN Digital Library

Fast and scalable mathematical expressions search engine

Master Project



Autor:
Arthur Oviedo

Supervisors:
CERN: Nikolaos Kasioumis
EPFL: Karl Aberer

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Chapter 2

Theoretical Background

Information Retrieval (IR) consists on the activity of identifying relevant documents from a collection based on some input parameters. Different models have been developed through the evolution of the field. In the subsequent sections, the most relevant to this work are presented.

2.1 Boolean Model

This model was the first one to be developed and is currently one of the most used in the current implementation of IR systems because of its simplicity.

2.2 Vector Space Model

2.3 Probabilistic Relevance Model

The probabilistic relevance model was developed in 1976 by Robertson and Jones[1]. The similarity of a document d_j to a query q is given by the probability of d_j being relevant to q. It assumes there is a set R that is preferred to be the answer set for q. The model assumes that the document in R are relevant to q and the documents in $\bar{R}$ are not relevant. R then is the set that maximizes:

$$sim(d_j, q) = \frac{P(R|d_j)}{P(\bar{R}|d_j)}$$

The importance of this model is that it serves as a solid base for state of the art information retrieval models.

2.4 Okapi BM25

The Okapi BM25 ranking function is one of the current state-of-the-art models in information retrieval[2]. The actual scoring function is called BM25 and Okapi refers to the first system implementing this function in the 1980s. Given a query Q as a vector of keywords $q_1, q_2, ..., q_n$ then the score for a document d is:

$$score(d, Q) = \sum_{i=1}^{n} IDF(q_i) * \frac{frec(q_i, D)}{frec(q_i, D) + k_1 * ((1 - b) + b * \frac{|D|}{avgdl})}$$

and

$$IDF(q_i) = \log \frac{N - df(q_i) + 0.5}{df(q_i) + 0.5}$$

where $frec(q_i, D)$ is the number of occurrences of keyword q_i in D, $avgdl$ is the average length (In keywords) of a document, and k_1 and b are free parameters. Usually $k_1 = 2$ and $b \in [0, 1]$ controls how important is the length normalization, being set normally to 0.75. N is the size of the collection, $df(q_i)$ is the number of documents that contain term q_i

A further variation that takes into account structure in the form of a document containing different fields and each of them having its own length and importance (boost) can be formulated as follows, known as BM25F can be expressed as follows:

$$score(d, Q) = \sum_{i=1}^{n} \frac{weigth(q_i, d)}{k + weigth(q_i, d)} * IDF(q_i)$$

and

$$weight(q_i, d) = \sum_{j=1}^{m} \frac{frec(q_i, D, field_j) * boost_j)}{((1 - b_j) + b_j * \frac{|l_j|}{avgfl_j})}$$

where m is the number of different fields, $boost_j$ is the relative importance for each field, b_j is analogous to the b constant in the BM25 group of equations, $|l_j|$ is the length of the $field_j$ and $avgfl_j$ is the average length of the $field_j$.

2.5 Tree Edit Distance

As it will be discussed, one of the most common formats to represent mathematical equations, MathML, is a XML based format, and whose structure represents a rooted tree. Because of this, it makes sense to

consider some models to compare and analyse mathematical expressions taking into account their natural tree structure. The most natural way of addressing this, is through the well studied abstract problem of Tree Edit Distance problem, where the idea is to compute a distance between a pair of trees. It should be noted that the development of this problem is very similar to the String Edit Distance problem. The Tree Edit Distance problem can be formulated as follows: Let Σ be a finite alphabet and let $\lambda \notin \Sigma$ be a special empty symbol. Finally let $\Sigma_\lambda = \Sigma \cup \lambda$. Given two trees T_1 and T_2 that are labelled (That is, each node N in the tree has assigned a label $l(N) \in \Sigma$) and ordered (That is, we are provided an order between sibling nodes), we define the following operations $(l(N_1) \to l(N_2))$ where $(l(N_1), l(N_2)) \in ((\Sigma \cup \lambda) \times (\Sigma \cup \lambda) \setminus (\lambda, \lambda))$:
Relabelling: If $l(N_1) \neq \lambda \wedge l(N_2) \neq \lambda$. We change the label of a node in the tree T
Deletion: If $l(N_2) = \lambda$. The children of N_1 become the children of N_1's parent (N_1 is not the root of the tree) and occupy its position in the sibling ordering.
Insertion: If $l(N_1) = \lambda$. This is the complement operation of deletion.

For each of this operations $\omega = (l(N_1) \to l(N_2))$, we assign a cost function $cost(\omega)$ and for a given sequence of this transformations $\Omega = (\omega_1, \omega_2, ..., \omega_n)$ we assign a cost function $cost(\Omega) = \sum_{i=1}^{i=n} cost(\omega_i)$ We also assume that $cost(\omega)$ is a distance metric. With the previous definition, the distance between T_1 and T_2 can be expressed now as $dist(T_1, T_2) = min\{cost(\Omega) | \Omega \text{ is a sequence of operations that transform } T_1 \text{ into } T_2\}$

A recursive formulation of the problem can be defined in the following way:

$$
\begin{aligned}
dist(T, T) &= 0 \\
dist(F_1, T) &= dist(F_1 - v, T) + cost(v \to \lambda) \\
dist(T, F_2) &= dist(T, F_2 - w) + cost(\lambda \to w) \\
dist(F_1, F_2) &= \begin{cases} dist(F_1 - v, F_2) + cost(v \to \lambda) \\ dist(F_1, F_2 - w) + cost(\lambda \to w) \\ dist(F_1(v), F_2(w)) + dist(F_1 - T_1(v), F_2 - T_2(w)) + cost(v \to w) \end{cases}
\end{aligned}
$$

Where F is a forest, v and w nodes from a free, $F - v$ the forest F with the deletion of the node v, $T(v)$ the tree rooted at v and $F - T(v)$ the forest obtained by removing all the nodes in $T(v)$ from F

The presented set of equations give a simple way to compute the edit distance between a pair of trees, and the most known algorithms like Zhang

and Shasha's[3] whose worst time case time bound is $O(|T_1|^2|T_2|^2)$, Klein's[4] that achieves a lower bound of $O(|T_1|^2|T_2|log|T_2|)$ or Demaine's[5], take into advantage the fact that the recursion relies in the solution of smaller sub-problems, so by carefully choosing which of this sub-problems to solve.

Chapter 3

Technical Considerations

In the previos chapter we presented different abstract models which can be suitable in the building of a math based search system. In this section, we present different aspects that need to be considered when mapping from these theoretical models to a concrete implementation.

3.1 Digital Representations of Mathematics

3.1.1 MathML

3.1.2 OpenMath

3.1.3 LaTeX

3.2 Math Extraction Tools

3.3 From LaTeX to MathML

3.3.1 From PDF to MathML

Maxtract

InftyReader

OCR analysis

3.3.2 Other projects

3.4 Computational Algebraic Systems

In the previous section we discussed about algorithmical ways to tranform, simplify and detect patterns in mathematical equations. Here, we explore what tools are available for performing this type of operations and how they are suitable for our system.

3.4.1 Mathematica

3.4.2 Mapple

3.4.3 Matlab

3.4.4 Octane

3.4.5 SAGE

Chapter 4

Related Work

4.1 Status of MIR

4.1.1 TREC

4.1.2 TREC

4.1.3 CLEF

4.1.4 NTCIR

4.2 Mathematical based search projects

4.2.1 MIaS

MIaS[6] (Math Indexer and Searcher), currently, is one of the flagship projects in the indexing and searching of mathematical content. As most of the other projects, it processes documents in MathML format. It allows the user to include in the queries mathematical expressions and textual content. During indexing time, equations are transformed following a set of heuristics that include:

- Ordering of elements: Taking into account commutativity of certain operators (Addition, multiplication), elements are ordered such that $3 + a$ would be converted into $a + 3$ (Since the <mi> tag comes first than the <mn> tag)

- Unification of variables: This process takes into the account the structure of the equations in despite of the naming of the variables. Expres-

sions like $a + b^a$ and $x + y^x$ would be converted into an expressions of the form $id_1 + id_2^{id_1}$ which would match.

- Unification of constants: This step consists of replacing all occurrences of constants (<mn> tags) by a const symbol.

The extracted tokens from a given equation consist of all its valid sub-expressions. The original expression is given a weight value of 1, and following sub-expressions are given smaller weights depending on how general or specific they are. The system, also as most of the current projects, is developed by using the Apache Lucene framework. The system adapts the default scoring equation such that the given weight is taken into account.

4.2.2 EgoMath

EgoMath[7] and its new version EgoMath^2[8], is a system oriented to index the mathematical content from the Wikipedia.org database. The processing steps before indexing are similar to the ones in MIaS (Rearranging of symbols, unification of constants). The extraction process started from a complete dump of the Wikipedia database and filtering only the math articles, which are identified by the "<math>" tag. This consists on around thirty thousand articles, from which 240.000 equations were identified. The processing step includes translating the equation from LaTeX into MathML format and then indexing both representation. Some additional effort is done into splitting large mathematical blocks like tables into single mathematical expressions.

4.2.3 DLMFSearch

4.2.4 MathWebSearch

4.2.5 LaTEXSearch

LaTEXSearch `http://latexsearch.com/` is a system developed by the scientific publisher Springer that allows to search over a database of around eight million latex snippets extracted from their publications. The system, unfortunately, is proprietary and no details are provided about the internals of the system.

4.3 Other related projects and initiatives

4.3.1 arXMLiv

Chapter 5

CERN Math Explorer

In this chapter we propose our solution for a scalable math based oriented search engine. We present the key points of our system, and how they are related to previous built systems.

5.1 Information Retrieval Model

Our main goal is to develop a system that compliments the already available search capabilities in Invenio and in particular the CDS instance. Our system was developed using the Apache Solr/Lucene framework.

5.1.1 Notational Features

Notational features correspond to the elements that relate to the actual naming of the variables, constants and numeric quantities found in an equation. Here we extract tokens that represent single leaf nodes (E.g. <mi>, <mn>, <mo> tags), and simple constructs like subscript and superscript elements, fractions, roots. We also extract bi-grams and tri-grams as analogue to that is done for text indexing in current systems.

After examining a set of relevant documents from our dataset, we identified some heuristics that could be applied to improve the recall of our system:

Unicode Normalizing

The first issue we discovered is that because of the big wide of different characters to represent the possible mathematical symbols, there is need for an automatic way to match different possible variations of a single character. An example of this situation is specification of an natural number, sometimes denoted by the letter N. In this case, at least three different common

representations were identified: $N = 1$ (Using character with unicode code point 0x004e), $\mathcal{N} = 1$ (0xd835 0xdc41 or in latex expressed as \mathcal{N}) and $\mathbb{N} = 1$ (0x2115 or in latex expressed as \mathbb{N}).
For addressing this situation, we rely on the Unicode equivalence framework. In the unicode specification, sometimes the same character have two or more different code-points because of backwards compatibility with other encoding standards or because the same character has different essential meanings. Another group of equal characters with different code-points are pre-combined characters such as the Latin letter Ñ which has Unicode point 0x00D1 and also is the sequence of code-points 0x004E (Latin letter N) and 0x0303 (the combining character tilde). Taking this sample into account, Unicode defines the character composition and decomposition operations as replacing the decomposed representation with the pre-composed one and vice-versa. Finally, Unicode defines two types of equivalences between characters: Canonical and compatible. Canonical equivalent characters are assumed to have the same appearance and meaning. Compatible ones are allowed to have slightly different graphical representations but the same meaning in some contexts. One example of compatible equivalent characters is the Roman number 12 Ⅻ with code-point 0x216b and the sequence of characters XII with codepoints 0x0058 0x0049 0x0049. Taking into account the composition or decomposition of characters and the canonical or compatible equivalences, Unicode specifies 4 different forms of a given character:

- NFD: Normalization Form Canonical Decomposition
- NFC: Normalization Form Canonical Composition
- NFKD: Normalization Form Compatibility Decomposition
- NFKC: Normalization Form Compatibility Composition

Table 5.1 presents a small list of troubling characters classes that were identified in our datasets.

Character Unicode Name	Visual Rendering	NFC	NFD	NFKC	NFKD
LATIN CAPITAL LETTER A	A	0x41	0x41	0x41	0x41
LATIN CAPITAL LETTER A WITH MACRON	Ā	0x100	0x41 0x304	0x100	0x41 0x304

LATIN CAPITAL LETTER A WITH RING ABOVE	Å	0xc5	0x41 0x30a	0xc5	0x41 0x30a
ANGSTROM SIGN (0x212b)	Å	0xc5	0x41 0x30a	0xc5	0x41 0x30a
LATIN CAPITAL LIGATURE IJ	IJ	0x132	0x132	0x49(I) 0x4a(J)	0x49 0x4a
CYRILLIC CAPITAL LIGATURE A IE	Ӕ	0x4d4	0x4d4	0x4d4	0x4d4
LATIN CAPITAL LETTER AE	Æ	0xc6	0xc6	0xc6	0xc6
DOUBLE-STRUCK CAPITAL N	ℕ	0x2115	0x2115	0x2115	0x4e
LATIN CAPITAL LETTER N	N	0x4e	0x4e	0x4e	0x4e
MATHEMATICAL ITALIC CAPITAL N	$\mathcal{N}$	0xd835 0xdc41	0xd835 0xdc41	0x4e	0x4e
ROMAN NUMERAL TWELVE	Ⅻ	0x216b	0x216b	0x58 0x49 0x49	0x58 0x49 0x49
LATIN CAPITAL LETTER X - LATIN CAPITAL LETTER I - LATIN CAPITAL LETTER I	XII	0x58 0x49 0x49	0x58 0x49 0x49	0x58(X) 0x49(I) 0x49(I)	0x58 0x49 0x49
VULGAR FRACTION ONE QUARTER	¼	0xbc	0xbc	0x31(1) 0x2044(/) 0x34(4)	0x31 0x2044 0x34
DIGIT ONE - SOLIDUS - DIGIT 4	1⁄4	0x31 0x2f 0x34	0x31 0x2f 0x34	0x31 0x2f 0x34	0x31 0x2f 0x34
DIGIT ONE - SOLIDUS - DIGIT 4	1⁄4	0x31 0x2f 0x34	0x31 0x2f 0x34	0x31 0x2f 0x34	0x31 0x2f 0x34
DIGIT ONE - FRACTION SLASH - DIGIT 4	1⁄4	0x31 0x2044 0x34	0x31 0x2044 0x34	0x31 0x2044 0x34	0x31 0x2044 0x34
THERE EXISTS	∃	0x2203	0x2203	0x2203	0x2203

THERE DOES NOT EXIST	∄	0x2204	0x2203 0x338	0x2204	0x2203 0x338
LATIN CAPITAL LETTER OPEN E	Ɛ	0x190	0x190	0x190	0x190
EULER CONSTANT	ℇ	0x2107	0x2107	0x190	0x190
PLANCK CONSTANT	h	0x210e	0x210e	0x68	0x68
LATIN SMALL LETTER H	h	0x68	0x68	0x68	0x68
PLANCK CONSTANT OVER TWO PI	$\hbar$	0x210f	0x210f	0x127	0x127
LATIN SMALL LETTER H WITH STROKE	ħ	0x127	0x127	0x127	0x127
GREEK CAPITAL LETTER OMEGA	Ω	0x3a9	0x3a9	0x3a9	0x3a9
OHM SIGN	Ω	0x2126	0x2126	0x3a9	0x3a9
INTEGRAL	$\int$	0x222b	0x222b	0x222b	0x222b
DOUBLE INTEGRAL	$\iint$	0x222c	0x222c	0x222b 0x222b	0x222b 0x222b
CONTOUR INTEGRAL	$\oint$	0x222e	0x222e	0x222e	0x222e

Table 5.1: Characters with similar glyphs and/or semantics

Since our goal is to match as much as possible similar characters, our system employs the NFKD representation of a character. For a given token, we test whether if it is already equal to its NFKD. If not, for all the characters that are not combining nor control characters, we add the token into the same position as the original one. For example, given the token <mi>Å</mi>, this will lead to the sequence <mi>Å</mi><mi>A</mi> and the token <mi>Ⅻ</mi> will produce <mi>Ⅻ</mi><mi>X</mi><mi>I</mi><mi>I</mi>.

Synonym Expansion

Even though the Unicode normalization step work for a big part of the cases presented in table 5.1, there are still some groups of characters which

would not be matched (Even partially), such as some types of integrals. For addressing these groups, a precomputed table was created for some general groups of characters which have some similar meaning. For each token that belongs to some of this predefined groups, a second token is created identifying the group. For example the token <mo>∮</mo> is expanded into <mo>∮</mo><mo>INTEGRALS</mo>, similarly the token token <mo>∫</mo> is expanded into <mo>∫</mo><mo>INTEGRALS</mo> and then both will have a partial match in the INTEGRALS token. Unicode specification does not define a strict rule to group similar characters based on their semantics. For general operators, we used the grouping provided by Xah Lee in his website[9].

Numeric Approximation

Some equations relate specific numeric quantities. While most of the previously explorer systems, handled <mn> tags by only representing them as a constant identifier, this approach is not suitable. A lot of articles in CDS relate to specific experiments under certain conditions, and require a high level of precision in the measurements.

5.1.2 Structural Features

5.1.3 Development with Lucene/Solr

5.1.4 Integration with Mathematica

This section describes some aspects that should be considered while implementing the integration with the Mathematica CAS.

Recursive interpretation

Error handling

Sometimes after certain time Mathematica will crash, launch an exception and following invocations would fails as well. Also, re-instantiating a KernelLink will yield an Exception. For this scenario, killing the associated processes to Mathematica was necessary:

```
killall -s 9 Mathematica
killall -s 9 MathKernel
```

or

```
taskkill /im Mathematica.exe
taskkill /im MathKernel.exe
```

Depending on the OS the system is running on.

With this, the next creation of a new KernelLink works properly and the subsequent invocations work as expected.

Timeout Handling

Unfortunately, checking the quality of a LaTeX document is a difficult job and users have a lot of freedom in the way they write their document since the only visible result is the rendered version. One example we found during our work is the following TeX snippet:

```
"\ \varphi\ are\ shown,\ as\ well\ as\ the\ weight\ and\ tension"
```

A human can easily identify that this corresponds to text and that should be inserted into a proper "mbox" or "text" tag. In this case, the snippet produced the following MathML code

```
<math><mrow><mi> </mi><mo></mo><mrow><mi>a</mi><mo></mo><mi>r</mi>
<mo></mo><mi>e</mi></mrow><mo></mo><mrow><mi>s</mi><mo></mo>
<mi>h</mi><mo></mo><mi>o</mi><mo></mo><mi>w</mi><mo></mo><mi>n</mi>
</mrow><mo>,</mo><mrow><mi>a</mi><mo></mo>
<mi>s</mi></mrow><mo></mo><mrow><mi>w</mi><mo></mo>
<mi>e</mi><mo></mo><mi>l</mi><mo></mo><mi>l</mi>
</mrow><mo></mo><mrow><mi>a</mi><mo></mo>
<mi>s</mi></mrow><mo></mo><mrow><mi>t</mi><mo></mo>
<mi>h</mi><mo></mo><mi>e</mi></mrow><mo></mo>
<mrow><mi>w</mi><mo></mo><mi>e</mi><mo></mo>
<mi>i</mi><mo></mo><mi>g</mi><mo></mo><mi>h</mi>
<mo></mo><mi>t</mi></mrow><mo></mo><mrow>
<mi>a</mi><mo></mo><mi>n</mi><mo></mo><mi>d</mi>
</mrow><mo></mo><mrow><mi>t</mi><mo></mo><mi>e</mi>
<mo></mo><mi>n</mi><mo></mo><mi>s</mi><mo></mo>
<mi>i</mi><mo></mo><mi>o</mi><mo></mo><mi>n</mi>
</mrow></mrow></math>
```

Mathematica took around 10 minutes to interpret it. In the ideal case we would be able to identify the malformed expressions and pass to Mathematica only well-formed equations to be processed but as this simple example shows, that is a very challenging task and may require more work. As a remedial situation, we used a fixed timeout to stop the processing of a given expression.

5.1.5 Integration with CDS

Chapter 6

Evaluation

... The machine used for experiments has a Intel Core i7 processor running at 3.4Ghz with 4 cores. The available RAM is 8Gb, and the Java Virtual Machine used for running Solr was run with the $-Xms2G - Xmx5G$ parameters. The OS was Linux Mint 16 Petra, which is a fork based on Ubuntu, running on kernel 3.11.0-15.

6.1 Dataset

6.2 Indexing Performance

For this set of evaluations, we were only interested in observing how the performance of the indexing stage is affected by the different types of features that are employed. We selected the first 100 records from our dataset and measured the total time it required to index all the elements in Solr using a python script and the solrpy library to do the communication. The default similarity in Solr was used. We also measured the index size at the end of each experiment. Our four scenarios were organized as follows:

- N : Only notational features over the original expression were used
- N+S : Notational and structural features over the original expression
- N+S+NN : Notational and structural features over the original expression and notational features over the normalized string using Mathematica's Simplify function
- N+S+FN : Notational and structural features over the original expression and notational features over the normalized string using Mathematica's Full Simplify function

Table **??** summarizes the results of this set of experiments. The first result is the significant increase in processing time when using Mathematica. Going from the N scenario to N+S represent an increase of almost 15x in the indexing time. Adding standard normalization increases the indexing time by a factor near to 1.44x and full normalization by a factor of 1.55x.

With respect to storage size, we see a small footprint (Less than 5%)in the total size by going from N to N+S. Adding normalization represents an increase by a factor of 1.78x and the size of the fully normalized index represents a smaller increase (1.76x)

We can observe that there is a small trade-off between time and storage size depending on which normalization mode is employed.

Feature Set	Total time (sec)	Index Size (kb)	Avg. time per equation	Avg. size of equation (kb)
N	58	13926	0.00524	1.237
N+S	884	14438	0.0785	1.282
N+S+NN	1273	25780	0.113	2.345
N+S+FN	1377	25548	0.122	2.269

Bibliography

[1] Stephen E. Robertson and Karen Sparck Jones. Document retrieval systems. chapter Relevance Weighting of Search Terms, pages 143–160. Taylor Graham Publishing, London, UK, UK, 1988.

[2] Stephen Robertson and Hugo Zaragoza. The probabilistic relevance framework: Bm25 and beyond. *Found. Trends Inf. Retr.*, 3(4):333–389, April 2009.

[3] Kaizhong Zhang and Dennis Shasha. Simple Fast Algorithms for the Editing Distance Between Trees and Related Problems. *Siam Journal on Computing*, 18:1245–1262, 1989.

[4] Philip N. Klein. Computing the edit-distance between unrooted ordered trees. In *Proceedings of the 6th Annual European Symposium on Algorithms*, ESA '98, pages 91–102, London, UK, UK, 1998. Springer-Verlag.

[5] Erik D. Demaine, Shay Mozes, Benjamin Rossman, and Oren Weimann. An Optimal Decomposition Algorithm for Tree Edit Distance. *ACM Transactions on Algorithms*, 6:146–157, 2007.

[6] Petr Sojka and Martin Líška. Indexing and searching mathematics in digital libraries. In JamesH. Davenport, WilliamM. Farmer, Josef Urban, and Florian Rabe, editors, *Intelligent Computer Mathematics*, volume 6824 of *Lecture Notes in Computer Science*, pages 228–243. Springer Berlin Heidelberg, 2011.

[7] Jozef Mišutka and Leo Galamboš. Extending full text search engine for mathematical content. *Towards Digital Mathematics Library*, pages 55–67, 2008.

[8] Jozef Mišutka and Leo Galamboš. System description: Egomath2 as a tool for mathematical searching on wikipedia.org. In *Proceedings of the*

18th Calculemus and 10th International Conference on Intelligent Computer Mathematics, MKM'11, pages 307–309, Berlin, Heidelberg, 2011. Springer-Verlag.

[9] Xah Lee. Unicode: Math symbols, 2010. `http://web.archive.org/web/20131223003502/http://xahlee.info/comp/unicode_math_operators.html`.