

A Search Engine for Mathematical Formulae

Ioan Alexandru Şucan

*Computer Science
International University Bremen
Campus Ring 1
28759 Bremen
Germany*

*Type: Bachelor Thesis
Date: May 7, 2006
Supervisor: Prof. Michael Kohlhase*

Abstract. We present a search engine for mathematical formulae. The MATHWEBSEARCH system harvests the web for content representations (currently MATHML and OPENMATH) of formulae and indexes them with substitution tree indexing, a technique originally developed for accessing intermediate results in automated theorem provers. For querying, we present a generic language extension approach that allows constructing queries by minimally annotating existing representations. First experiments show that this architecture results in a scalable application that can easily be connected to a standard search engine thus creating an even more powerful application. As the world of information technology grows, the amount of mathematical information available on the web increases, and thus specialized searching becomes a more important need.

Table of Contents

1	Introduction	3
1.1	Semantic Search for Mathematical Formulae	3
1.2	State of the Art in Math Search	4
1.3	Content Representation for Mathematical Formulae ..	5
1.4	A Running Example: The Power of a Signal	6
2	Indexing Mathematical Formulae	6
3	A Query Language for Content Mathematics	8
4	The MATHWEBSearch Application	11
4.1	Input Processing	11
4.2	Term Indexing	13
4.3	Result reporting	14
4.4	Case Studies and Results	15
5	Conclusions and Future Work	16

1 Introduction

As the world of information technology grows, being able to quickly search data of interest becomes one of the most important tasks in any kind of environment, be it academic or not. This paper addresses the problem of searching mathematical formulae from a semantic point of view, i.e. to search for mathematical formulae not via their presentation but their structure and meaning.

1.1 Semantic Search for Mathematical Formulae

Searching for mathematical formulae is a non-trivial problem.

1. *Mathematical notation is context-dependent.* For instance, binomial coefficients can come in a variety of notations depending on the context: $\binom{n}{k}$, ${}_nC^k$, C_k^n , and C_n^k all mean the same thing:¹ $\frac{n!}{k!(n-k)!}$. In a formula search we would like to retrieve all forms irrespective of the notations.
2. *Identical presentations can stand for multiple distinct mathematical objects*, e.g. an integral expression of the form $\int f(x)dx$ can mean a Riemann Integral, a Lebesgue Integral, or any other of the 10 to 15 known anti-derivative operators. In an envisioned formula search, we would like to be able to restrict the search to the particular integral type we are interested in at the moment.
3. *Certain variations of notations are widely considered irrelevant*, for instance $\int f(x)dx$ means the same as $\int f(y)dy$ (modulo α -equivalence), so in a formula search we would like to find both, even if we only query for one of them.
4. We would like to be able to search occurrences of the query term as sub-formulae.

To solve this formula search problem, we concentrate on *content representations of mathematical formulae* (which solves the first two problems; see Section 1.3), since they are presentation-independent and disambiguate mathematical notions. Furthermore, we adapt term

¹ The third notation is the French standard, whereas the last one is the Russian one (see [KK06] for a discussion of social context in mathematics). This poses a very difficult problem for searching, since these two look the same, but mean different things.

indexing techniques known from automatic theorem provers to obtain the necessary *efficiency and expressivity in query processing* (see Section 1.2) and to build in common equalities like α -equivalence.

Concretely, we present the web application MATHWEBSEARCH that is similar to a standard search engine like GOOGLE, except that it can retrieve content representations of mathematical formulae not just raw text. The system² will be released under the Gnu General Public License [Fre91]. A running prototype is available for testing at <http://search.mathweb.org>.

1.2 State of the Art in Math Search

There seem to be two general approaches to searching mathematical formulae. One generates string representations of mathematical formulae and uses conventional information retrieval methods, and the other leverages the structure inherent in content representations.

The first approach is utilized for the Digital Library of Mathematical Functions [MY03]: T_EX/L^AT_EX formulae are converted to text and indexed. The search string is similar to L^AT_EX commands and is converted to string before performing the search. This allows searching for normal text as well as mathematical content simultaneously but it cannot provide powerful mathematical search — for example searching for something like $a^2 + c = 2a$, where a must be the same expression both times, cannot be performed. An analogous idea to this would be to rely on an XML-based XQuery search engine. Both these methods have the important advantage that they rely on already existing technologies but they do not fully provide a mathematical formulae oriented search method.

The second approach is taken by the MBASE system [KF01], which applies the pattern matching of the underlying programming language to search for OMDOC-encoded [Koh06] mathematical documents in the knowledge base. The search engine for the HELM project indexes structural meta-data gleaned from Content MATHML representations for efficient retrieval [AS04]. The idea is that this metadata approximates the formula structure and can serve as a filter for very large term data bases. However, since the full structure of

² To obtain pre-release code, please contact the authors.

the formulae is lost, semantic equivalences like α -equivalence cannot be taken into account.

1.3 Content Representation for Mathematical Formulae

The two best-known open markup formats for representing mathematical formulae for the Web are MATHML [ABC⁺03] and OPENMATH [BCC⁺04]. There are various other formats that are proprietary or based on specific mathematical software packages like Wolfram Research’s MATHEMATICA [Wol02]. We will not concern ourselves with them, since we are only interested in open formats.

MATHML offers two sub-languages: Presentation MATHML for marking up the two-dimensional, visual appearance of mathematical formulae, and Content MATHML as a markup infrastructure for the functional structure of mathematical formulae. In Content MATHML, the formula $\int_0^a \sin(x)dx$ would be represented as the following expression:

Listing 1. Content Representation of an Integral

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

The outer `apply` tags characterize this as an application of an integral to the `sin` function, where x is the bound variable. The format differentiates numbers (`cn`) from identifiers (`ci`) and objects with a meaning fixed by the specification (represented by about 80 MATHML token elements like `int`, or `plus`). The OPENMATH format follows a similar approach, but replaces the fixed set of token elements for known concepts by an open-ended set of concepts that are defined in “content dictionaries”: XML documents that specify their meaning in machine-readable form (see [BCC⁺04,Koh06] for details).

As content markup for mathematical formulae is rather tedious to read for humans, it is mainly used as a source to generate Presentation MATHML representations. Therefore content representations are often hidden in repositories, only their presentations are available on the web. In these cases, the content representations have to be harvested from the repositories themselves. For instance, we

harvest the CONNEXIONS corpus, which is available under a Creative Commons License [Cre] for MATHWEBSEARCH. As we will see, this poses some problems in associating presentation (for the human reader) with the content representation. Other repositories include the ACTIVEMATH repository [MBG⁺03], or the MBASE system [KF01].

Fortunately, MATHML provides the possibility of “parallel markup”, i.e. representations where content and presentation are combined in one tree³ (see <http://functions.wolfram.com> for a widely known web-site that uses parallel markup).

1.4 A Running Example: The Power of a Signal

A standard use case for MATHWEBSEARCH is that of an engineer trying to solve a mathematical problem such as finding the power of a given signal $s(t)$. Of course our engineer is well-versed in signal processing and remembers that signal’s power has something to do with integrating its square, but has forgotten the details of how to compute the necessary integrals. He will therefore call up MATHWEBSEARCH to search for something that looks like $\int_?^? s^2(t)dt$ (for the concrete syntax of the query see Listing 5 in Section 3). Searching for this, directly finds a document about Parseval’s theorem, and more specifically $\frac{1}{T} \int_0^T s^2(t)dt = \sum_{k=-\infty}^{\infty} |c_k|^2$ where c_k are the Fourier coefficients of our signal. In short, our engineer found the exact formula he was looking for (he had missed the factor in front and the integration limits) and a theorem he may be able to use. So he would use MATHWEBSEARCH to find out how to compute the Fourier transform of the concrete signal $s(t)$, eventually solving the problem totally.

2 Indexing Mathematical Formulae

For indexing mathematical formulae on the web, we will interpret them as first-order terms (see Subsection 4.1 for details). Let V be the set of all mathematical symbols with 0 arity. Let F_n be the set of n -ary function symbols. Then T is the set of terms with

³ Modern presentation mechanisms will generate parallel markup, since that e.g. allows copy-and-paste into mathematical software systems [HRW02].

$V \subseteq T$ and $t_f = f(t_1, \dots, t_n) \in T$ if $f \in F_n$ and $t_i \in T$; t_i are called subterms of t_f ; subterms of t_i are subterms of t_f as well.

This allows us to use a technique from automated reasoning called *term indexing* [Gra96]. This is the process by which a set of terms is stored in a special purpose data structure where common parts of the terms are potentially shared, so as to minimize access time and storage. The indexing technique we will be working with is a form of tree-based indexing called *substitution-tree indexing*. To explain the idea behind substitution-tree indexing, we will first define substitutions. A *substitution* σ is a mapping from mathematical symbols to terms and is generally represented as $\{v_1 \rightsquigarrow t_1, \dots, v_n \rightsquigarrow t_n\}$ with $v_i \in V$ and $t_i \in T$. A substitution tree, as the name suggests, is then simply a tree where substitutions are the nodes. A term is constructed by successively applying substitutions along a path in the tree. The terms we want to index will be reproduced at the leafs of the built tree. The internal nodes of the tree define the so-called *generic terms*, that represent similarities between terms to be indexed. Generic terms always contain one or more placeholder variables and potentially placeholder strings. These play an important role in the searching process (see Section 3). Placeholder variables are in fact terms of the form *@string*. They can be replaced by any other term. Placeholder strings are strings of the form *@string* and can be replaced by any string. The sets of placeholder variables and placeholder strings are disjoint.

The main advantage of substitution tree indexing is that we only store substitutions, not the actual terms, and this leads to low amounts of data. Figure 2 shows a typical index for the terms $h(f(z, a, z))$, x , $g(f(z, y, a))$, $g(f(7, z, a))$, and $g(f(7, z, f))$. For clarity we present not only the substitutions in the node, but the term produced up to that node as well (between square brackets). The variables *@integer* are used to denote placeholder variables for parts that differ between terms. All placeholder variables are substituted before a leaf is reached.

Adding data to an existing index is simple and fast, querying the data structure is reduced to performing a walk down the tree. This technique is particularly well suited for theorem provers, where combining already generated indexes — an operation called tree merging — is often used. In our case however, we want the searching to be

as fast as possible and we don't need tree merging at all. Therefore we use substitutions only when building the index. Index building is done in similar fashion to [Gra96]. Once the index is built, we keep the actual term instead of the substitution at each node, so we do not have to recompute it with every search. At first glance this may seem to be against the idea of indexing, as we would store all the terms again, not only the substitutions. However, due to the tree-like structure of the terms, we can in fact store only a pool of (sub)terms and define the terms in our index using pointers to elements of the pool (which are simply other terms). To each of the indexed terms, a data string is attached — a string that represents the exact location of the term. We use XPointer [GMMW03] to specify this (see Subsection 4.3 for more details).

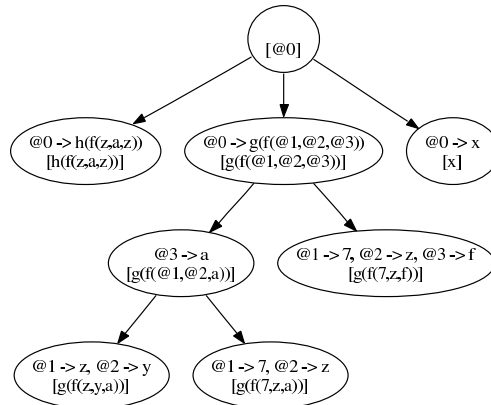


Fig. 2. An Index with Five Terms

Unfortunately, substitution tree indexing does not support subterm search in an elegant fashion, so when adding a term to the index, we add all its subterms as well. This may not seem to be the ideal solution, but it turns out that the increase in index size remains manageable (see Section 4.4) and it greatly simplifies the implementation. The rather small increase is caused by the fact that many of the subterms are shared among larger terms and they are only added once.

3 A Query Language for Content Mathematics

When designing a query language for mathematical formulae, we have to satisfy a couple of conflicting constraints. The language should be content-oriented (obviously), and familiar to the human, but it should not be specialized to a given content representation format. Our approach to this problem is to use a simple, generic extension mechanism for XML-based representation formats rather than

a genuine query language itself. The query extension is very simple, it adds two new attributes to the respective languages: `mq:generic` and `mq:anyorder`, where the prefix `mq:` abbreviates the namespace URI `http://mathweb.org/MathQuery/`.

In this way, the user need not master a new representation language, and we can generate queries by copy and paste and then make parts of the formulae generic by simply adding suitable attributes. Moreover, for any representation format that allows extensions by attributes in other namespaces (e.g. MATHML and OPENMATH), we can directly reuse all their XML tools. We will use Content MATHML [ABC⁺03] as an example here (we call the query language MATHML^Q), but MATHWEBSEARCH also supports OPENMATH and a shorthand notation (we simply call it STRING) that resembles the internal representation we are using for terms (prefix notation). The support for placeholder variables and placeholder strings is implicit — we can simply specify them with `@string`. There is no equivalent for the `mq:anyorder` attribute.

The `mq:generic` attribute takes string values and can be specified for any tag in the query. When such an attribute is encountered, the contents of the tag that has it is ignored and it matches any term in the search process.⁴ For example, the MATHML^Q expression⁵ in Listing 3 searches for any term that is or contains as subterm the function f applied to three arguments, where the first two arguments are the same. This would find expressions like $f(g(h), g(h), c)$ or $f(0, 0, 0)$. Note that the placeholder variable identified by `@same` may be equal to `@other` but is not required to be.

Listing 3. Example MATHML^Q Query

```
<apply><ci>f</ci>
  <ci mq:generic="same"/>
  <ci mq:generic="same"/>
  <ci mq:generic="other"/>
</apply>
```

⁴ Internally, a term consisting of a placeholder variable is produced. The placeholder variable has the same format (`@string`) as the ones of the placeholder variables used in the generic terms (terms produced at internal nodes in the tree). The value specified by the attribute is the value used to identify the placeholder variable. If at some point in the search, a placeholder variable matches a term, that term is bound to the value of the placeholder variable so that if the same placeholder variable is used multiple times in the query, the same term is required to be found.

⁵ The STRING representation of this is `f(@same,@same,@other)`.

In the case of searching expressions of the form $A = B$, we might like to find occurrences of $B = A$ as well. At this point, the `mq:anyorder` attribute comes in. Inside an `apply` tag, the first child defines the function to be applied; if this child has the attribute `mq:anyorder` defined with the value “yes”, the order of the next children is ignored (terms with all the possible orderings are generated from the MATHML^Q query). This attribute may be used multiple times⁶, at different levels or not, in the same query.

If we do not want to specify the function name, we can use the `mq:generic` attribute again, but this time for the first child of an `apply` tag. If for example we want to find terms that contain the same function applied twice to some other term, we may use something like⁷

Listing 4. Another Example MATHML^Q Query

```
<apply><ci mq:generic="fun" />
  <apply><ci mq:generic="fun" /><ci mq:generic="rest" /></apply>
</apply>
```

Given the above, the MATHML^Q query of our running example has the form presented in Listing 5. Note that we do not know the integration limits or whether the formula is complete or not. Expressing this in MATHML^Q⁸

Listing 5. Query for Signal Power

```
<math xmlns="http://www.w3.org/1998/Math/MathML"
  xmlns:mq="http://mathweb.org/MathQuery" >
  <apply><int />
    <domainofapplication mq:generic="domain" />
    <bvar> <ci mq:generic="time" /> </bvar>
    <apply><power />
      <apply><ci mq:generic="fun" /></ci><ci mq:generic="time" /></apply>
      <cn>2</cn>
    </apply>
  </apply>
</math>
```

⁶ The system however imposes a limit on how many different terms can be produced by a query so that not too many searches are performed at once by a single client.

⁷ This is equivalent to the STRING representation `@fun(@fun(@rest))`

⁸ This is equivalent to the STRING representation `int(bvarset(bvar(@time)), @domain,power(@fun(@time),nr(2)))`.

4 The MathWebSearch Application

We have built a web search engine around the indexing technique explained above. Like commercial systems, MATHWEBSEARCH consists of three system components: a set of web crawlers⁹ that periodically scan the Web, identify, and download suitable web content, a search server encapsulates the index, and a web server that communicates the results to the user. To ensure scalability, we have the system architecture in Figure 6, where individual search servers are replicated via a search meta-server that acts as a front-end.

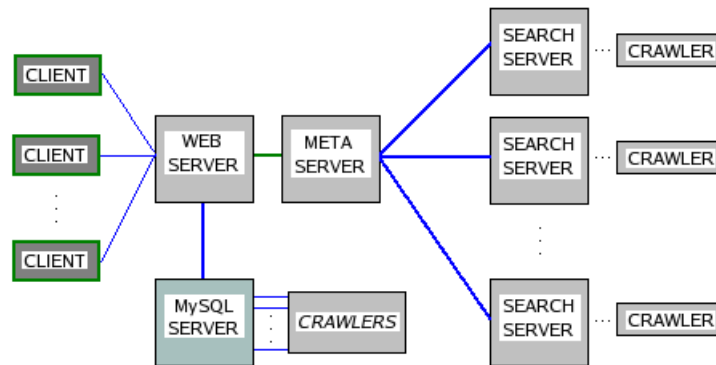


Fig. 6. The Architecture of the MATHWEBSEARCH Application

4.1 Input Processing

MATHWEBSEARCH can process any kind of XML representation for content mathematics. The system is modular and provides an interface through which PERL modules (that convert XML nodes to index terms) can simply be attached to our list of XML-processors. Currently, the system supports MATHML and OPENMATH. We will discuss input processing for the former here.

⁹ At the moment, we are employing an OAI-based [OAI02] crawler for repositories like CONNEXIONS and a standard web-crawler for finding other MATHML repositories.

Given an XML document, we create an index term for each of its `math` elements. Consider the example on the right: We have the standard mathematical notation of an

1) Mathematical expression: $f(x) = y$	2) Content MATHML: <code><apply><eq/> <apply> <ci>f</ci> <ci>x</ci> </apply> representation: $eq(f(x), y)$</code>
---	---

equation (1), its Content MATHML representation (2) and the term we extract for indexing (3). As previously stated, any mathematical construct can be represented in a similar fashion to (3).

When we process the Content MATHML formulae, we roughly create a term for every `apply` element tag, taking the first child of `apply` as the function and the rest of the children as arguments. Of course, cases like vectors or matrices have to be treated specially. In some cases — e.g. for integrals — the same content can be encoded in multiple ways. Here, a simple standardization of both the indexed formulae and the queries leads to an improved recall of the search: for instance we can find an integral specified with `lowlimit` and `uplimit` tags (see Listing 1) using a query integral specified with the `interval` element¹⁰, since we standardize argument order and integration domain representation for integrals.

Even search modulo α -renaming becomes available via a very simple input processing trick. In the input processing, we add a `mq:generic` attribute to every bound variable (but different with different strings for different variables). Therefore in our running example the query variable t (`@time` in Listing 5) in the query $\int_?^? s^2(t)dt$ is made generic, therefore the query would also find the variant $\frac{1}{T} \int_0^T s^2(x)dx = \sum_{k=-\infty}^{\infty} |c_k|^2$: as t is generic it could principally match any term in the index, but given the MATHML constraints on the occurrences of bound variables, it will in reality only match variables (thus directly implementing α -equivalence).

Presentation MATHML in itself does not offer much semantic information, so it is not particularly well suited for our purposes. However, most of the available MATHML on the World Wide Web is Presentation MATHML. For this reason, we index it as well. The lit-

¹⁰ STRING representation: `int(bvarset(bvar(id(x))), intervalclosed(lowlimit(nr(0)), uplimit(nr(a))), sin(id(x)))`.

the semantic information we are offered, like when a number (`mn`), operator (`mo`) or identifier (`mi`) are defined, we use for recovering simple mathematical expressions which we then index as if the equivalent Content MATHML were found. This offers the advantage that when using a mixed index (both Presentation and Content MATHML) we have increased chances of finding a result.

4.2 Term Indexing

As the term retrieval algorithm for substitution trees is standard, we will concentrate on term insertion and memory management here. In a nutshell: we insert a term in the first suitable place found. This will not yield minimal tree sizes, but (based on the experiments carried out in [Gra96]) the reduction in size is not significant and the extra computation time is large.

Concretely, an initial empty index contains a single node with the empty substitution. The term produced by that node is always the generic term `@0`. When a new term is to be inserted, we always try to insert it starting with the root, using the algorithm `INSERT_TERM`, where

1. `COMPLETE_MATCH` checks if the second argument is an instance of the first argument. It uses a simple rule: a term is only an instance of itself and of any placeholder variable.
2. `PARTIAL_MATCH` returns an integer that represents the number of equal subterms.
3. `INSERT_AT` adds a new leaf to *node* with a substitution from *node.term* to *term* unless that substitution is empty.
4. `INSERT_WITH_SEPARATION` creates a son of *node* named *n* with a substitution to the shared parts of *son.term* and *term*; it then adds proper substitutions to *son.term* and *term* from *n.term* as sons of *n*.

An important optimization that we wish to further explain refers to memory management. We mentioned above that we actually store the terms themselves at every node so that we do not need to recompute them with every search. To keep the memory requirements low and still be able to do this, we store the terms in a pool. Whenever we want to add a term to the index, we do not create a new instance

Algorithm 1 INSERT_TERM(*node*, *term*)

```
    found = true
2: while found do
    found = false
4:   for all sons of node do
      if COMPLETE_MATCH(son.term, term) then
6:         node = son, found = true
          break
8:       end if
    end for
10: end while
    match = PARTIAL_MATCH(node.term, term)
12: for all sons of node do
      if PARTIAL_MATCH(son.term, term) > match then
14:         return INSERT_WITH_SEPARATION(node, son, term)
          end if
16: end for
    return INSERT_AT(node, term)
```

of it but point to a term in the pool. If the term is not already in the pool, we add it. We also make sure that if a subterm of the term we want to add is already in the pool, we use that instance of the subterm and not redefine it. For this to work, we need an ordering on terms: we first order terms by number of subterms, then by name and then by recursively looking at the subterms, in increasing order of index.

4.3 Result reporting

For a search engine for mathematical formulae, we need to augment the set of result items (usually page title, description, and page link) reported to the user for each hit. As typical pages contain multiple formulae, we need to report the exact occurrence of the hit in the page, we do this by supplying an XPOINTER reference [GMMW03] where possible. Concretely, we group all occurrences into one page item that can be later expanded and we order the groups by number of contained references. See Figure 7 for an example. For any given result, a detailed view is available. This view shows the exact term that was matched (using Presentation MATHML) and the used substitution (a mapping from the query variables specified by the `mq:generic` attributes to certain subterms) to match that specific term.

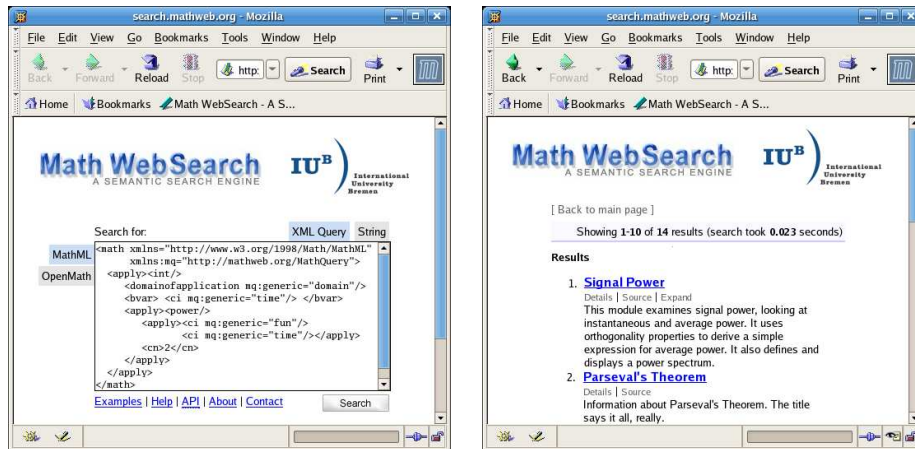


Fig. 7. A Search at <http://search.mathweb.org>.

A more serious problem comes from the fact that — as mentioned above — content representations are often the source from which presentations are generated. If MATHWEBSEARCH can find out the correspondence between content and presentation documents, it will report both to the user. For instance for CONNEXIONS we present two links as results: one is the *source link*, a link to the document we actually index, and the *default link*, a link to the more aesthetically pleasing presentation document.

4.4 Case Studies and Results

We have tested our implementation on the content repository of the CONNEXIONS Project, available via the OAI protocol [OAI02]. This gives us a set of over 3,200 articles with mathematical expressions to work on. The number of terms represented in these documents is approximately 53,000 (77,000 including subterms). The average term depth is 3.6 and the maximal one is 14. Typical query execution times on this index are in the range of milliseconds. The search in our running example takes 14 ms for instance. There are however complex searches (e.g. using the `mq:anyorder` attribute) that internally call the searching routine multiple times and take up to 200 ms but for realistic examples execution time is below 50 ms. We are currently building an index of the 86,000 Content MATHML

formulae from <http://functions.wolfram.com>. Here, term depths are much larger (average term depth 5.7, maximally around 50) resulting in a much larger index: it is just short of 2 million formulae. First experiments indicate that search times are largely unchanged by increase in index size.

5 Conclusions and Future Work

We have presented a search engine for mathematical formulae on the Internet. In contrast to other approaches, MATHWEBSEARCH uses the full content structure of formulae, and is easily extensible to other content formats. A first prototype is available for testing at <http://search.mathweb.org>. We will continue developing MATHWEBSEARCH into a production system.

We plan to index more content, particularly OPENMATH. In the long run, it would be interesting to interface MATHWEBSEARCH with a regular web search engine and create a powerful, specialized, full-feature application. This would resolve the main disadvantage our implementation has – it cannot search for simple text. Finally we would like to allow specification of content queries using more largely known formats, like L^AT_EX or even ASCII: strings like $\frac{1}{x^2}$ or $1/x^2$ could be processed as well. This would make MATHWEBSEARCH accessible for a larger group of users.

References

- [ABC⁺03] Ron Ausbrooks, Stephen Buswell, David Carlisle, Stéphane Dalmas, Stan Devitt, Angel Diaz, Max Froumentin, Roger Hunter, Patrick Ion, Michael Kohlhase, Robert Miner, Nico Poppelier, Bruce Smith, Neil Soiffer, Robert Sutor, and Stephen Watt. Mathematical Markup Language (MathML) version 2.0 (second edition). W3C recommendation, World Wide Web Consortium, 2003.
- [AS04] Andrea Asperti and Matteo Selmi. Efficient retrieval of mathematical statements. In Andrea Asperti, Grzegorz Bancerek, and Andrej Trybulec, editors, *Mathematical Knowledge Management, MKM'04*, number 3119 in LNAI, pages 1–4. Springer Verlag, 2004.
- [BCC⁺04] Stephen Buswell, Olga Caprotti, David P. Carlisle, Michael C. Dewar, Marc Gaetano, and Michael Kohlhase. The Open Math standard, version 2.0. Technical report, The Open Math Society, 2004.
- [Cre] Creative Commons. Web page at <http://creativecommons.org>. seen August 2006.
- [Fre91] Free Software Foundation. Gnu general public license, 1991.
- [GMMW03] Paul Grosso, Eve Maler, Jonathan Marsh, and Norman Walsh. Xpointer framework. W3C recommendation, World Wide Web Consortium W3C, 25 March 2003.
- [Gra96] Peter Graf. *Term Indexing*. Number 1053 in LNCS. Springer Verlag, 1996.
- [HRW02] Sandy Huerter, Igor Rodionov, and Stephen Watt. Content-faithful transformations for mathml. In *Second International Conference on MathML and Technologies for Math on the Web*, Chicago, USA, 2002.
- [KF01] Michael Kohlhase and Andreas Franke. MBase: Representing knowledge and context for the integration of mathematical software systems. *Journal of Symbolic Computation; Special Issue on the Integration of Computer algebra and Deduction Systems*, 32(4):365–402, 2001.
- [KK06] Andrea Kohlhase and Michael Kohlhase. Communities of Practice in MKM: An Extensional Model. In Jon Borwein and William M. Farmer, editors, *Mathematical Knowledge Management, MKM'06*, number 4108 in LNAI, pages 179–193. Springer Verlag, 2006.
- [Koh06] Michael Kohlhase. OMDOC – *An open markup format for mathematical documents [Version 1.2]*. Number 4180 in LNAI. Springer Verlag, 2006.
- [MBG⁺03] Erica Melis, Jochen Büdenbender, George Gogvadze, Paul Libbrecht, and Carsten Ullrich. Knowledge representation and management in active-math. *Annals of Mathematics and Artificial Intelligence*, 38:47–64, 2003. see <http://www.activemath.org>.
- [MY03] Bruce R. Miller and Abdou Youssef. Technical aspects of the digital library of mathematical functions. *Annals of Mathematics and Artificial Intelligence*, 38(1-3):121–136, 2003.
- [OAI02] The open archives initiative protocol for metadata harvesting, June 2002.
- [Wol02] Stephen Wolfram. *The Mathematica Book*. Cambridge University Press, 2002.