A Search Engine for Mathematical Formulae

Michael Kohlhase and Ioan A. Şucan

Computer Science, International University Bremen
m.kohlhase@iu-bremen.de i.sucan@iu-bremen.de

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.

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

Generally, searching for mathematical formulae is a non-trivial problem — especially if we want to be able to search occurrences of the query term as sub-formulae:

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^n_k, and C^k_n all mean the same thing:[1]

[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.

$\frac{n!}{k!(n-k)!}$. 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. 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 we would like to find both, even if we only query for one of them.

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 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 system is released under the Gnu General Public License [FSF91] (see [Mat06] for details). 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] and ActiveMath system [LM06]: mathematical formulae are converted to text and indexed. The search string is similar to LaTeX 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 the formulae is lost, semantic equivalences like α-equivalence cannot be taken into account.

Another system that takes this second approach is described in [TSP06]. It uses term indexing for interfacing with Computer Algebra Systems while determining applicable algorithms in an automatically carried proof. This is closely related to what we present in this paper, the main difference being that we provide search for any formula in a predefined index, while in [TSP06] a predefined set of formulae characterizing an algorithm is automatically searched for in a changing index.

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][2] 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.

[2] There are various other formats that are proprietary or based on specific mathematical software packages like Wolfram Research's MATHEMATICA [Wol02]. We currently support them if there is a converter to OPENMATH or MATHML.

In Content MathML, the formula $\int_0^a \sin(x)dx$ would be represented as the following expression:

Listing 1.1. 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 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 tree[3] (see `http://functions.wolfram.com` for a widely known web-site that uses parallel markup).

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

1.4 A Running Example: The Power of a Signal

A standard use case[4] 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 a 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 1.3 in Section 3). MathWebSearch finds a document about Parseval's Theorem, more specifically $\frac{1}{T}\int_0^T s^2(t)dt = \Sigma_{k=-\infty}^{\infty}|c_k|^2$ where c_k are the Fourier coefficients of the 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 again to find out how to compute the Fourier transform of the concrete signal $s(t)$, eventually solving the problem completely.

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). 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 (the **index**, normally stored in memory) where common parts of the terms are potentially shared, so as to minimize access time and storage. The indexing technique we work with is a form of tree-based indexing called *substitution-tree indexing*. A substitution tree, as the name suggests, is simply a tree where substitutions are the nodes. A term is constructed by successively applying substitutions along a path in the tree, the leaves represent the terms stored in the index. Internal nodes of the tree are **generic terms** and represent similarities between terms.

[4] We use this simple example mainly for expository purposes here. Other applications include the retrieval of equations that allow to transform a formula, of Lemmata to simplify a proof goal, or to find mathematical theories that can be re-used in a given context (see [Nor06a] for a discussion of the latter).

The main advantage of substitution tree indexing is that we only store substitutions, not the actual terms, and this leads to a small memory footprint. Figure 1 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.

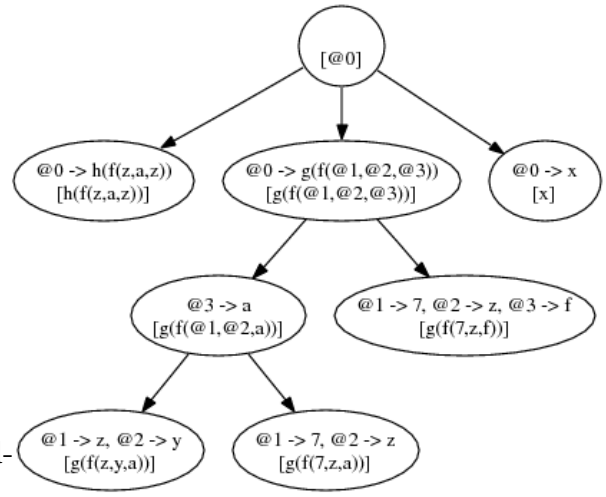

Fig. 1. An Index with Five Terms

Adding data to an existing index is simple and fast, querying the data structure is reduced to performing a walk down the tree. In contrast to automated reasoning our application does not need tree merging. Therefore we use substitutions only when building the index. Index building is done based on Algorithm 1. 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. Structure sharing methods conserve memory and make this tractable. To each of the indexed terms, some data is attached — an identifier that relates the term to its exact location. The identifier, location and other relevant data are stored in a database external to the search engine. We use XPointer [GMMW03] references to specify term locations (see Subsection 4.3 for more details).

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 simple trick works well: 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 and familiar, 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 (referred to as base format) rather than a genuine query language itself.

The extension mechanism is represented by 4 tags and 4 attributes. The extension tags are `mq:query`, `mq:and`, `mq:or`, `mq:not`. The `mq:query` tag is used if one or more of the other extension tags are to be used and encloses the whole search expression. The tags `mq:and`, `mq:or`, `mq:not` may be nested and can contain tags from the base XML format, which may carry extension attributes (will be explained later). The `mq:and`, `mq:or`, `mq:not` tags are logical operators and may carry the `mq:target` attribute (the default value is `term`; only one other value allowed: `document`) which specifies the scope of the logical operator. Scope `term` is used to find *formulae* that contain the query terms as subformulae, while scope `document` does not restrict the occurrences of query terms.

There are 3 other attributes that may be used for any of the base format tags: `mq:generic`, `mq:anyorder` and `mq:anycount`. The first is used to specify that a term matches any subterm in the index; we call it a **generic term**. Note that generic terms with the same `mq:generic` value must be matched against identical target subterms. The `mq:anyorder` is used to specify that the order of the children can be disregarded. The `mq:anycount` attribute defines any number of occurrences of a certain base tag (if that base tag is known to be allowed multiple times). This is useful e.g. to define a variable number of bound variables (`bvar` MathML).

Listing 1.2 shows a (somewhat contrived but illustrative) example query that searches for documents that contain at least one mathematical formula matching each of the `math` tags in the query. The

first `math` tag will match any application of function f to three arguments, where at least two of the arguments are the same. The second `math` tag will match any formula containing at least two consecutive applications of the same function to some argument.

Listing 1.2. Example MathML^Q Query

```
<mq:query xmlns:mq="http://mathweb.org/MathQuery">
  <mq:and mq:target="document">
    <math xmlns="http://www.w3.org/1998/Math/MathML">
      <apply><ci mq:anyorder="yes">f</ci>
        <ci mq:generic="same"/>
        <ci mq:generic="same"/>
        <ci mq:generic="other"/>
      </apply>
    </math>
    <math xmlns="http://www.w3.org/1998/Math/MathML">
      <apply><ci mq:generic="fun"/>
        <apply><ci mq:generic="fun"/><ci mq:generic="rest"/></apply>
      </apply>
    </math>
  </mq:and>
</mq:query>
```

Given the above, the MathML^Q query of our running example has the form presented in Listing 1.3. Note that we do not know the integration limits or whether the formula is complete or not. Expressing this in MathML^Q[5]

Listing 1.3. 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>
```

4 The MathWebSearch Application

We have built a web search engine around the indexing technique explained above. Like commercial systems, MathWebSearch con-

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

sists of three system components: a set of web crawlers[6] 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 2, where individual search servers are replicated via a search meta-server that acts as a front-end.

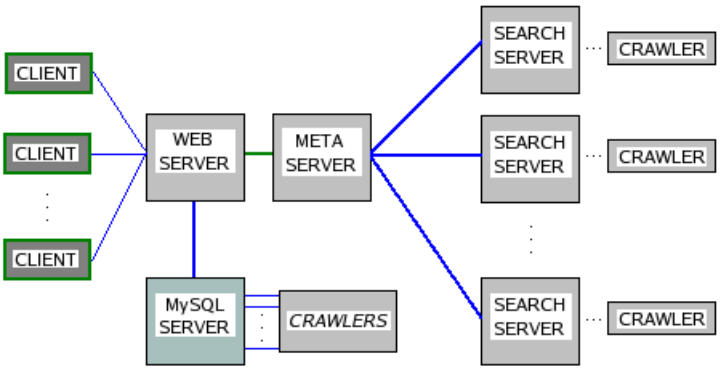

Fig. 2. The Architecture of the MathWebSearch Application

4.1 Input Processing

MathWebSearch can process any Xml-based content mathematics. Currently, the system supports MathML and OpenMath (and Mathematica notebooks via the system's MathML converter). We will discuss input processing for the first here.

Given an Xml document, we create an index term for each of its `math` (this is the case for MathML) elements. Consider the example on the right: We have the standard mathematical notation of an 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.

1) Mathematical expression: $f(x) = y$

3) Term representation: $eq(f(x), y)$

2) Content MathML:

```
<apply><eq/>
  <apply>
    <ci>f</ci>
    <ci>x</ci>
  </apply>
  <ci>y</ci>
</apply>
```

[6] 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.

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 1.1) using a query integral specified with the `interval` element[7], since we standardize argument order and integration domain representation for integrals.

Search modulo α-renaming becomes available via a very simple input processing trick: during input processing, we add a `mq:generic` attribute to every bound variable (but with distinct strings for different variables). Therefore in our running example the query variable t (`@time` in Listing 1.3) in the query $\int_?^? s^2(t)dt$ is made generic, therefore the query would also find the variant $\frac{1}{T}\int_0^T s^2(x)dx = \Sigma_{k=-\infty}^{\infty}|c_k|^2$: 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).

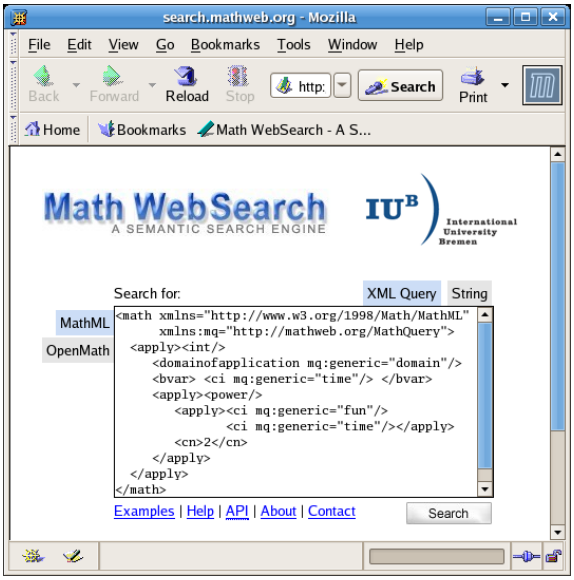

Fig. 3. Searching for Signal Power

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

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

Presentation MathML. For this reason, we index it as well. The little 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 number of internal nodes is not significant and the extra computation time is large.

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)
```

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 from 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.

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 where possible. Concretely, we group all occurrences into one page item that can be expanded on demand and within this we order the groups by number of contained references. See Figure 4 for an example.

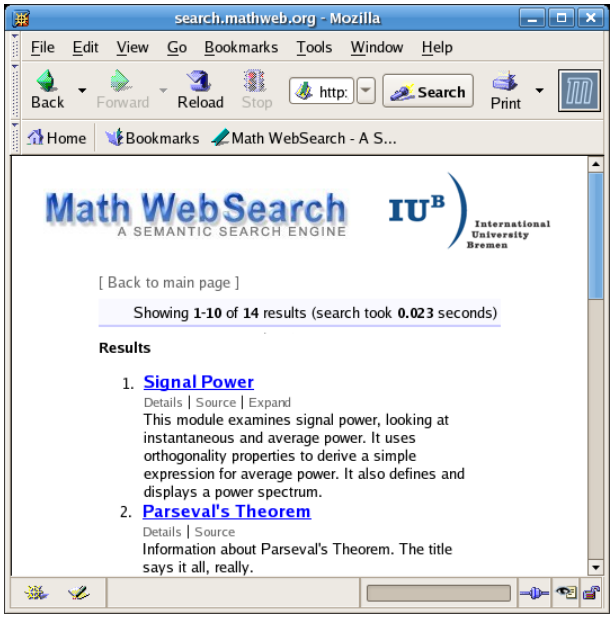

Fig. 4. Results for the Search in Fig. 3

For any given result, a detailed view is available. This view shows the exact term that was matched and the used substitution (a mapping from the query variables specified by the `mq:generic` attributes to certain subterms) to match that specific term. 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.

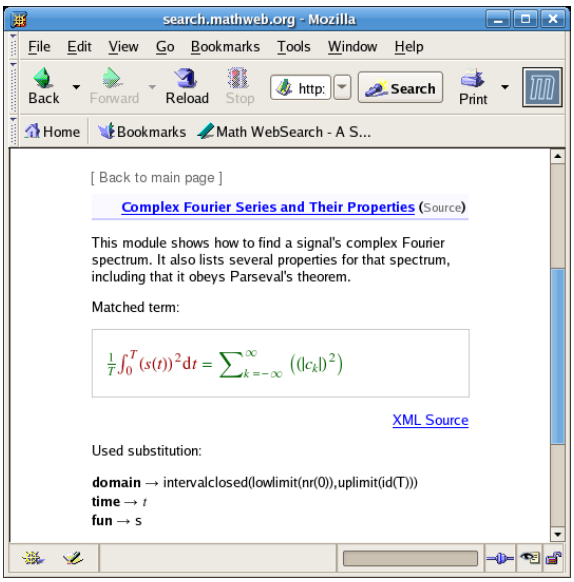

Fig. 5. Detailed Search Results

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 3,400 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 23 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 also built an index of the 87,000 Content MathML formulae from `http://functions.wolfram.com`. Here, term depths are much larger (average term depth 8.9, maximally 53) resulting in a much larger index: 1.6 million formulae; total number of nodes in the index is 2.9 million, resulting in a memory footprint of 770MB. First experiments indicate that search times are largely unchanged by the increase in index size (for reasonably simple searches).

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.

A current weakness of the system is that it can only search for formulae that match the query terms up to α-equivalence. Many applications would benefit from similarity-based searches or stronger equalities. For instance, our search in Listing 1.3 might be used to find a useful identity for $\int_{\infty}^{0} f(x) \cdot g(x)dx$, if we know that $s(x) \cdot s(x) = s^2(x)$. MathWebSearch can be extended to a *E-Retrieval* engine (see [Nor06b]) without compromising efficiency by simply E-standardizing index and query terms.

We plan to index more content, particularly more 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 LaTeX: 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, et al. Mathematical Markup Language (MathML) version 2.0 (second edition). W3C recommendation, World Wide Web Consortium, 2003. Available at http://www.w3.org/TR/MathML2.

[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 LNCS, 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. `http://www.openmath.org/standard/om20`.

[Cre] Creative Commons. Web page at `http://creativecommons.org`.

[FSF91] Free Software Foundation FSF. Gnu general public license. Software License available at `http://www.gnu.org/copyleft/gpl.html`, 1991.

[GMMW03] Paul Grosso, Eve Maler, Jonathan Marsh, and Norman Walsh. Xpointer framework. W3c recommendation, World Wide Web Constortium 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. `http://www.mathmlconference.org/2002/presentations/huerter/`.

[ICW06] Tetsuo Ida, Jacques Calmet, and Dongming Wang, editors. *Proceedings of Artificial Intelligence and Symbolic Computation, AISC'2006*, number 4120 in LNAI. Springer Verlag, 2006.

[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, September 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. Springer Verlag, 2006.

[Koh06] Michael Kohlhase. OMDoc *An open markup format for mathematical documents (Version 1.2)*. Number 4180 in LNAI. Springer Verlag, 2006. in press `http://www.mathweb.org/omdoc/pubs/omdoc1.2.pdf`.

[LM06] Paul Libbrecht and Erica Melis. Methods for access and retrieval of mathematical content in ActiveMath. In N. Takayama and A. Iglesias, editors, *Proceedings of ICMS-2006*, number 4151 in LNAI. Springer Verlag, 2006. forthcoming.

[Mat06] Math web search. Web page at `http://kwarc.eecs.iu-bremen.de/projects/mws/`, seen July 2006.

[MBG+03] Erica Melis, Jochen Büdenbender, George Goguadze, Paul Libbrecht, and Carsten Ullrich. Knowledge representation and management in ActiveMath. *Annals of Mathematics and Artificial Intelligence*, 38:47–64, 2003. see `http://www.activemath.org`.

[MY03] B. Miller and A. Youssef. Technical aspects of the digital library of mathematical functions. *Annals of Mathematics and Artificial Intelligence*, 38(1-3):121–136, 2003.

[Nor06a] Immanuel Normann. Enhanced theorem reuse by partial theory inclusionss. In Ida et al. [ICW06].

[Nor06b] Immanuel Normann. Extended normalization for e-retrieval of formulae. to appear in the proceedings of Communicating Mathematics in the Digital Era, 2006.

[OAI02] The open archives initiative protocol for metadata harvesting, June 2002. Available at `http://www.openarchives.org/OAI/openarchivesprotocol.html`.

[TSP06] Frank Theiß, Volker Sorge, and Martin Pollet. Interfacing to computer algebra via term indexing. In Silvio Ranise and Roberto Sebastiani, editors, Calculemus-2006, 2006.

[Wol02] Stephen Wolfram. *The Mathematica Book*. Cambridge University Press, 2002.