MathWebSearch 0.5
Open Formula Search Engine

Corneliu C. Prodescu and **Michael Kohlhase**
Computer Science, Jacobs University Bremen
http://kwarc.info

Abstract

We present an open-source, open-format, content-oriented search engine for Mathematical formulae. MATHWEBSEARCH is a complete system capable of crawling, indexing and querying expressions based on their functional structure (operator tree) rather than their presentation. MATHWEBSEARCH indexes content-rich representation formats (like MATHML) using left-most substitution trees, a technique inspired from Automated Theorem Proving which shows great advantages in terms of space and time complexity. In version 0.5 we re-implemented the index and made all the APIs web standards conformant. Our 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. We present the MATHWEBSEARCH system; a search engine that addresses the problem of searching mathematical formulae from a semantic point of view i.e. finds formulae by their structure and meaning not via their presentation (as a standard engine like GOOGLE does).

In [KŞ06] we have presented the motivation, query language, and web front end of MATHWEBSEARCH 0.2. In [KK07] we have re-examined the value proposition of semantic search for mathematical knowledge homing in on the benefits and sacrifices induced by the various search approaches [You06b; MM06; LM06], from an user's perspective. The result of this analysis is MATHWEBSEARCH 0.5, which we describe in this paper. The new version features significant efficiency gains, new management features, advanced searching capabilities, and a new user interface. The MATHWEBSEARCH system (see [Mat] for details) is released under the Gnu General Public License.

Note that throughout the paper, we assume the following namespace declarations to be in effect in all the examples.

```
xmlns:m="http://www.w3.org/1998/Math/MathML"
xmlns:mws="http://search.mathweb.org/ns"
```

Before we present the MATHWEBSEARCH system in Section 5, we will recap the foundations, in particular, the approach of using unification for querying, MATHML as a representation format (Section 3) for mathematical formulae and the indexing technique used (Section 4) in the implementation. In Section 6, we evaluate the system on a corpus, and finally Section 7 concludes the paper and discusses future work.

2 Querying Mathematics by Unification

Very often one finds himself in the position that he remembers parts of a formula but does not remember details like coefficients or arguments. This is the typical use case where **instantiation queries** might come of use as they find indexed terms which are obtained after replacing the parts marked by the user as unknown with some terms i.e. they become instantiated. For example, consider 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

$$\int_{\boxed{min}}^{\boxed{max}} \boxed{f}(x)^2 dx \tag{1}$$

Throughout the paper, we mark query variables as named boxes[1]. One of the search results will be the page containing the term (Parseval's theorem)

$$\frac{1}{T}\int_0^T s^2(t)dt = \sum_{k=-\infty}^{\infty} \|c_k\|^2 \tag{2}$$

about the energy of a signal s. This not only confirms the definition of the energy (in the surrounding text), but also gives a way to compute it via the Fourier coefficients c_k of s. Such a hit is reported as, by default, the search engine is instructed to also report subterms of indexed terms which match the given search expression.

Let us now consider a student who encounters $\int_{\mathbb{R}^2} |\sin(t)\cos(t)|dt$ and wishes to know if there are any mathematical statements (like theorems, identities, inequalities) that can be applied to it. Indeed, there are many such statements (for example Hölder's inequality) and they can be found using **generalization queries**. The idea behind answering generalization queries is that the index marks universal[2] variables in subterms as generalization targets. Hence, the search engine looks for terms in

[1] Note that this already gives us a more expressive query language than e.g. regular expressions supported by some text-based search engine, since we can use variable co-occurrences to query for co-occurring subterms.

[2] We consider an identifier as universal, if it can be instantiated without changing the truth value of the containing expression. In formal representations like first-order logic, such variable

$$\int_{\boxed{D}} \left|\boxed{f}(\boxed{x})\boxed{g}(\boxed{x})\right| d\boxed{x} \leq \left(\int_{\boxed{D}} \left|\boxed{f}(\boxed{x})\right|^p d\boxed{x}\right)^{\frac{1}{p}} \left(\int_{\boxed{D}} \left|\boxed{g}(\boxed{x})\right|^q d\boxed{x}\right)^{\frac{1}{q}}$$

Figure 1: A formula with Universal Variables in the Index

the index which after instantiating the universal identifiers become equal to the query. For our example, we have in the index the term (we reuse the box notation for generalization targets in the index) in Figure 1 which the search engine instantiates $\boxed{x} \mapsto t, \boxed{f} \mapsto sin, \boxed{g} \mapsto cos, \boxed{D} \mapsto \mathbb{R}^2$ in order to find the generalization query. Note that the variant query $\int_{\mathbb{R}^2} |\sin(t)\cos(2t)|dt$ will not find Hölder's inequality since that would introduce inconsistent substitutions $\boxed{x} \mapsto t$ and $\boxed{x} \mapsto 2t$.

A very similar idea is used in **variation queries** where the indexed terms are searched to match the search expression but only renamings of generic terms are allowed. This type of queries prove to be helpful when the structure of the term needs to be maintained.

Sometimes, however, one is in the position that the searching criteria is somewhere between instantiation queries (i.e. parts are unknown) and generalization queries (parts are probably instantiated already). In this case we give the possibility to pose **unification queries**. As the name suggests, the query just finds terms which are unifiable with the search expression. A query like $g^2\cos(\boxed{x}) + b\sin(\sqrt{y})$ would match the term $\boxed{a}\cos(\boxed{t}) + \boxed{b}\sin(\boxed{t})$ as we can substitute $\boxed{x} \mapsto \sqrt{y}, \boxed{t} \mapsto \sqrt{y}, \boxed{a} \mapsto g^2, \boxed{b} \mapsto b$ to get the term $g^2\cos(\sqrt{y}) + b\sin(\sqrt{y})$.

3 Representation of Formulae

The Mathematical Markup Language (MATHML; see [ABC+10]) is an XML [BPSM97]-based representation format for mathematical formulae. MATHML provides a basis for machine to machine communication of mathematics (content MATHML) and allows the inclusion of mathematical expressions in web pages (presentation MATHML).

To emphasize the critical differences between *Presentation* and *Content* MATHML, let us consider a simple example in the form of the following expression: $f(a+b)$. This formulae has an ambiguous meaning; it is either a multiplication between the terms f and $a+b$, or an application of f to $a+b$. *Presentation* MATHML, having a display-oriented syntax, does not distinguish between the two cases:

Listing 1: Presentation MathML for $f(a+b)$

```
<m:mrow>
  <m:mi>f</m:mi>
  <m:mo>&InvisibleTimes;</m:mo>
  <m:mfenced open="(" close=")">
    <m:mrow>
      <m:mi>a</m:mi>
      <m:mo>+</m:mo>
      <m:mi>b</m:mi>
    </m:mrow>
  </m:mfenced>
</m:mrow>
```

occurrences can be effectively computed, but in semi-formal settings like mathematical textbooks, they have to be approximated by heuristic methods; see the discussion in the conclusion for details

However, *Content* MATHML, with its machine-oriented syntax, dispels any ambiguity. For our example, depending on the context, we would get one of the representations in Figure 2. We observe that, when reading mathematical

multiplication	application

`<m:apply>`
`  <m:times/>`
`  <m:ci>f</m:ci>`
`  <m:apply>`
`  <m:plus/>`
`  <m:ci>a</m:ci>`
`  <m:ci>b</m:ci>`
`  </m:apply>`
`</m:apply>`
	`<m:apply>`
	`  <m:ci>f</m:ci>`
	`  <m:apply>`
	`  <m:plus/>`
	`  <m:ci>a</m:ci>`
	`  <m:ci>b</m:ci>`
	`  </m:apply>`
	`</m:apply>`

Figure 2: Content MathML for $f(a+b)$

formulae, humans interpret the presentation, resolve ambiguities from the context and obtain a cognitive representation that is close to the content form. This is the desired form which is manipulated when "doing mathematics". As a consequence, we will build our search engine based internally on Content MATHML and we will show the presentation form as end results to the user.

4 Left-Most Substitution Tree Indexing

As we are interested in indexing Mathematical formulae at a large scale (document archives, text corpora), repetitive content is expected. After all, theorems are built on top of other theorems and terms on top of subterms. With this in mind, we chose a space-efficient internal representation based on substitutions.

In the previous version of MATHWEBSEARCH, we used a technique borrowed from Automated Theorem Proving called Substitution Indexing [Gra96]. It involves indexing expressions in a tree based on the generality. The root is a generic variable and as we go from a node to one of its children, one or more substitutions occur.

However, the conventional substitution index is not ideal, as running a query involves, at each node, going linearly through the possible substitutions to find the right child node.

Hence, the index was slightly modified. Whenever a term is introduced, it is broken down into substitutions in a consistent way. Every node involves exactly one substitution, always applied to the left-most free variable.

For example, $f(g(a,a),b)$ would be indexed in the following way, as it goes from the generic variable $@0$ of the root down to its corresponding leaf:

$$\begin{aligned} @0 &\to @1(@2,@3) \\ &\to f(@2,@3) \\ &\to f(@4(@5,@6),@3) \\ &\to f(g(@5,@6),@3) \\ &\to f(g(a,@6),@3) \\ &\to f(g(a,a),@3) \\ &\to f(g(a,a),b) \end{aligned}$$

This is particularly interesting, as this is similar with the representation of the expression in *Content* MATHML:

Listing 2: ContentMathML for $f(g(a,a),b)$

```
<m:apply>
  <m:ci>f</m:ci>
  <m:apply>
    <m:ci>g</m:ci>
    <m:ci>a</m:ci>
    <m:ci>a</m:ci>
  </m:apply>
  <m:ci>b</m:ci>
</m:apply>
```

If we take a Depth-First-Search traversal of the generated XML tree, it is easy to notice that every `m:apply` element induces a substitution of the form

$$@a_i \rightarrow @a_{i_1}(@a_{i_2}, \ldots, @a_{i_n})$$

with the number n of introduced variables equal to the number of children of the `m:apply` node, while all the other nodes resolve the left-most free variable.

Hence, indexing with left-most substitutions can be done efficiently, using Depth-First-Search traversals of the Content MATHML, holding only the node meanings [3] and the arities of the `m:apply` nodes.

5 The MathWebSearch System

The MATHWEBSEARCH system consists of the three main components pictured in Figure 3. The *crawler subsystem* collects data from the corpora[4]. It transforms the mathematical formulae in the corpus into *MWS Harvest*s (see 5.1) and feeds them into the core system. The *core system* (the MATHWEBSEARCH daemon `mwsd`) builds the search index and processes search queries: it accepts the MATHWEBSEARCH input formats (*MWS Harvest* and *MWS Query*) and generates the MATHWEBSEARCH output format (*MWS Answer Set*). These are communicated through the *RESTful interface* `restd` which provides a public HTTP API conforming to the REST paradigm.

These components have been implemented using POSIX-compliant [POS88] C++. We use the MicroHTTPd library [Mic] API for handling HTTP, and LibXML2 [Vei] API for XML parsing. The meta-data accompanying the internal index is stored using an external database system. As we are dealing mainly with key-value pairs retrieval, the BerkeleyDB [Ber09] API was preferred.

The system supports two main workflows:

1. The crawler sends a *MWS Harvest* to the MathWebSearch Daemon. The XML is parsed and an internal representation is generated. This is used to update the Substitution Indexing Tree and consequently the database.
2. The user sends a *MWS Query* the MathWebSearch Daemon. The XML is parsed, an internal query is generated. Using an efficient traversal of the Substitution Indexing Tree, formulas matching the search term are retrieved and aggregated into a result. This is translated to a *MWS Answer Set* and sent back to the user.

[3] The meaning of a node is a combination of its name, its attributes and its text content. For example, the `m:apply` node has all the relevant information in the tag, while others like `m:ci` hold it in the text area

[4] Note that we envision essentially one crawler per corpus. The crawlers are specialized to the respective formula representation, the organization and access methods to the corpus, etc. We have only implemented a crawler for the ARXIV (see Section 6), but additional crawlers can be patterned after this (see Section 7.2).

5.1 Data Representation

The *MWS Harvest* is the format that the MATHWEBSEARCH accepts as crawled data. A harvest is represented by the `mws:harvest` which contains a set of `mws:expr` elements that contain URI/expression pairs. The expression is represented in *Content* MATHML, and its URL+UUID in the `url` attribute.

Listing 3: Example Mws Harvest

```
<mws:harvest>
  <mws:expr url="http://math.example.org/art1234#e4321">
    <m:apply>
      <m:eq/>
      <m:apply>
        <m:plus/>
        <m:ci>x</m:ci>
        <m:ci>y</m:ci>
      </m:apply>
      <m:ci>z</m:ci>
    </m:apply>
  </mws:expr>
</mws:harvest>
```

The *MWS Query* format is used to present mathematical queries to the MATHWEBSEARCH system. A query is given by a set of content MATHML expressions with named query variables represented by the `mws:qvar` element. It is a place holder for any *Content* MATHML expression, so `mws:qvar` can appear wherever *Content* MATHML sub-expressions are expected. A formula t in a harvest matches a query with expressions e_1, ..., e_n, iff there is a joint unifier for t and the e_i. The root `mws:query` element uses the following attributes to specify additional search parameters.

- `answsize`: the maximum number of solutions to be returned
- `totalreq`: boolean value showing if a total count is requested besides the specified number of solutions.
- `limitmin`: the offset within the range of solutions (this is used in follow-up queries that request additional hits).

Listing 4: Example Mws Query

```
<mws:query limitmin="0" answsize="3" totalreq="yes">
  <mws:expr>
    <m:apply>
      <m:plus/>
      <mws:qvar>q1</mws:qvar>
      <mws:qvar>q2</mws:qvar>
    </m:apply>
  </mws:expr>
</mws:query>
```

*MWS Answer Set*s are encoded in `mws:answset` elements that use the attributes

- `size`: number of solutions in the returned answer set
- `total`: the total number of solutions found in the database

Its children, `mws:answ` nodes encode individual answers through their attributes

- `uri`: the URL+ID of the `m:math` node from which the solution was indexed.
- `xpath`: the XPath of the solution relative to the `m:math` node.

The children of `mws:answ` are substitution pairs that together represents a substitution which instantiates the user query to the answer in the current *MWS Answer Set*. The `mws:substpair` is characterized by the following attributes:

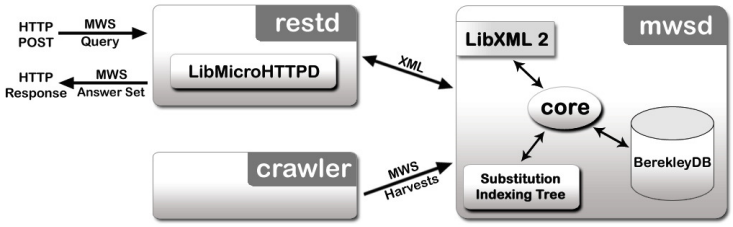

Figure 3: MWS-0.5 System Structure

- `qvar`: the name of the substituted qvar.
- `xpath`: the XPath of the qvar relative to the `m:math` node.

Listing 5: Example MATHWEBSEARCH Answer Set

```
<mws:answset size="1" total="123">
  <mws:answ xpath="1/2"
         uri="http://math.example.org/art1234#e4321">
    <mws:substpair qvar="q1" xpath="1/2/2"/>
    <mws:substpair qvar="q2" xpath="1/2/3"/>
 </mws:answ>
</mws:answset>
```

5.2 Core Algorithms

The key elements behind the MATHWEBSEARCH architecture are the indexing structure and the involved algorithms. As described in section 4, the indexing core is a substitution tree. Each node holds pointers to its children in a map indexed by token name and arity.

For example, the expression in listing 2 would be reached by starting from the root and following the path:

$$\begin{aligned} root &\rightarrow (m:apply,3) \rightarrow (f,0) \rightarrow (m:apply,3) \\ &\rightarrow (g,0) \rightarrow (a,0) \rightarrow (a,0) \rightarrow (b,0) \end{aligned}$$

In algorithms 1 and 2, we present the general ideas behind *term insertion* and *term query*. Note that the querying algorithm is simplified (the query variables have been omitted), as the extended algorithm implies a number of additional data structures and a rather lengthy approach.

Algorithm 1 INSERT_TERM$(node, term)$

```
var curr_node = node
stack dfs_container
PUSH(dfs_container, term.root)
while NOT_EMPTY(dfs_container) do
  var curr_term = POP(dfs_container)
  for all sons of curr_term in reverse order do
    PUSH(dfs_container, son)
  end for
  if curr_node.HAS_CHILD(curr_term) then
    curr_node = curr_node.GET_CHILD(curr_term)
  else
    curr_node.ADD_CHILD(curr_term)
    curr_node = curr_node.GET_CHILD(curr_term)
  end if
end while
DATABASE_INSERT(GET_ID(curr_node),term.metadata)
```

Algorithm 2 QUERY_TERM$(node, term)$

```
var curr_node = node
stack dfs_container
PUSH(dfs_container, term.root)
while NOT_EMPTY(dfs_container) do
  var curr_term = POP(dfs_container)
  for all sons of curr_term in reverse order do
    PUSH(dfs_container, son)
  end for
  curr_node = curr_node.GET_CHILD(curr_term)
  if curr_node==NULL then
    return NULL
  end if
end while
return DATABASE_QUERY(GET_ID(curr_node))
```

6 Evaluation

We evaluate the MATHWEBSEARCH implementation on a large corpus of mathematical formulae: We are currently working on translating the almost 700.000 TEX/LATEX articles on Physics, Mathematics, and Computer Science in the Cornell ePrint archive (see `http://www.arXiv.org`) to content MATHML [SK08]. We estimate that this collection contains in the order of 10^8 non-trivial formulae. To harvest these, the ARXMLIV crawler goes recursively through the pages of [arX] extracting the content MATHML[5] elements, combines them with URI references and reports them to MATHWEBSEARCH.

To be able to accurately asses the performance characteristics of the current MATHWEBSEARCH system, we record a number of parameters:

- Harvest size: The total number of harvested expressions, as well as the number of unique expressions.
- Average query times: This involves measuring several query response times. A standard query has `answsize` 30 and `limitmin` less than 9000. Response times are measured for standard queries (fresh and cached) with 0 up to 5 `qvars`. Additionally, stress queries with `answsize` 10000 are used.
- Memory usage: The memory footprint of the core process.

We'll start with the time efficiency aspect, as this is highly relevant for a search system. The average query times [6] are presented in Figure 4.

As one can see, the query response times are fairly constant as the harvest data increases. This fits with the theory, as the querying process will follow the same paths in the index (because the query data remains constant). The small increase is due to the slightly higher density of the index tree (which affects retrieval of the right path).

[5] The result of the transformation contains both content and presentation MATHML representations in parallel markup.

[6] Note that the queries were sent from the local network, to eliminate any channel delays

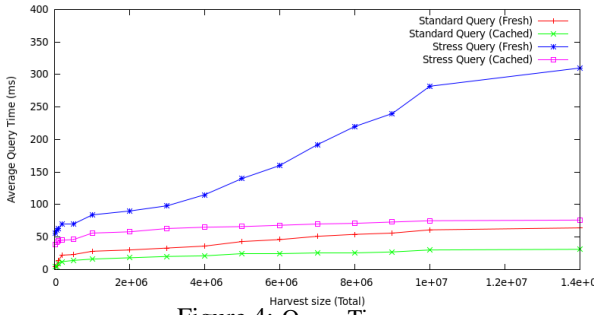

Figure 4: Query Times

The gap between the fresh and cached stress queries is expected, due to the fact that the current bottleneck is retrieving the meta-data from the external database. Hence, caching significantly improves the results.

Next, we'll focus our attention on the memory usage statistics (see Figure 5).

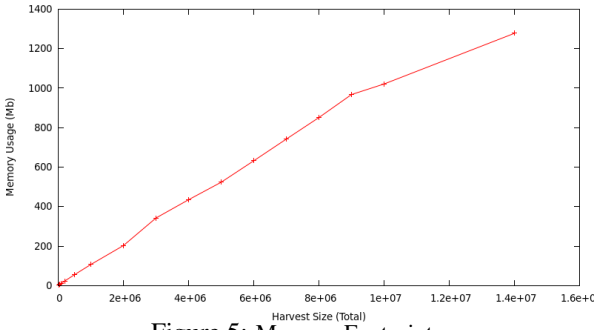

Figure 5: Memory Footprint

The graph is fairly linear, with a slight damping tendency. This is due to the fact that, as the index grows, expressions start building up on top of each other, thus saving memory. Unfortunately, this saturation effect is only noticeable for significant index sizes, which overflow an average server's memory pool in the current implementation.

Before the conclusions, we would like to note that the MATHWEBSEARCH system used for benchmarking was revision 652 and the host system had the following characteristics:

- Intel (R) T4300 2.10GHz Dual (1 MB L2 Cache)
- 3 Gbyte RAM
- Debian Sid (x86)

7 Conclusion and Future Work

We have presented a scalable search engine for mathematical formulae on the Internet (see figure 6). In contrast to other approaches, MATHWEBSEARCH uses the full content structure of formulae and is easily extensible to other content-oriented formats. Our first evaluation shows that query times are low and essentially constant in index/harvest size, so that a search engine can scale up to web proportions. Contrary to our expectations, index size is linear in harvest size for the ARXIV corpus, which transcends the main memory limits of standard servers. Therefore, developing parallelization strategies is a priority. Besides that, we will work on improving the features and extending the functionalities of the system, as we will discuss in this section.

7.1 System Parallelization

To be able to handle large[7] corpora, we need to come up with techniques to distribute the computational workload, as well as the raw data, between multiple nodes (machines). One of the possible approaches is to split the harvests based on a hash function. The same hash function could be used to distribute queries and an additional interface would aggregate the results. While being a simple task in theory, the implementation needs to be executed carefully, in order to overcome the typical issues of distributive systems (synchronization, robustness against node failure, etc). Besides this approach, we are also investigating local distribution techniques like storing the index in a hybrid data structure; a dictionary encodes the meanings (see section 4) into optimized forms which are used in the left-most substitution index.

7.2 Additional Corpora

The ARXIV corpus we are currently using for benchmarking is paradigmatic for the "informal but rigorous mathematics" that dominates mathematical communication today. Here, the Content MATHML has to be heuristically reconstructed from the presentation in the sources. This is unnecessary for corpora of formalized mathematics, e.g. the Mizar Mathematical Library [Miz] with over 50 000 formal theorems. Here the problems of obtaining the content MATHML are different: Even though the representations are formal in principle, the surface languages are human-oriented, and fully explicit representations need reconstruction processes (e.g. for reconstructing elided types and arguments, resolution of operator overloading, etc.) We are currently working on content MATHML exporters for the Mizar Library and the TPTP (Thousands of Problems for Theorem Proving) library [SS]. Other future targets could be the input files of mathematical software systems e.g. computer algebra systems like Mathematica, numerical computation systems like MatLab, or statistics programs like the R system.

7.3 Extending the Indexing and Interfaces

A current weakness of the system is the fact that it can only search for formulae that match the query terms up to α-equivalence. Many applications would benefit from stronger equalities for instance. Our search in the running example might be used to find a useful identity for $\int_0^\infty f(x) \cdot g(x)dx$, if we know that $s(x) \cdot s(x) = s^2(x)$. MATHWEBSEARCH can be extended to a *E-Retrieval* engine (i.e. search modulo an equational theory E or logical equivalence) without compromising efficiency by simply E-normalizing index and query terms (see [NK07] for a first implementation).

In the long run, we plan to extend MATHWEBSEARCH, so that it can take more document context information into account, i.e. not just keywords from the text around the formulae but e.g. the topology of theories in the OMDOC format [Koh06]: It would be very useful, if we could restrict searches to formulae that are consistent with current (mathematical) assumptions.

Another important area of future research is result ranking in MATHWEBSEARCH; indeed advances in ranking have made word-based search engines scalable from a user point of view. This is an open research question, there

[7] We deem a corpus as *large* if it has more than 20 Million expressions

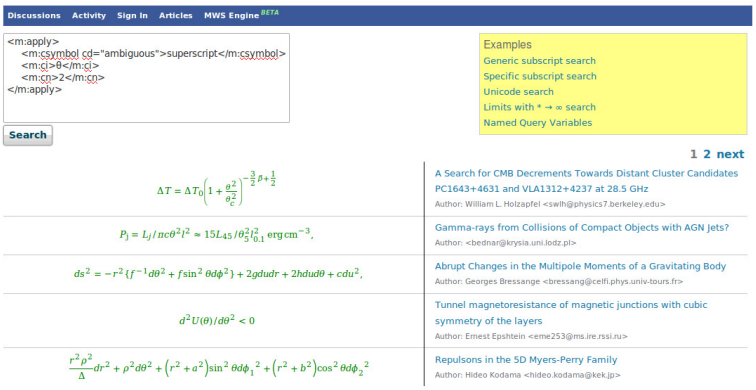

Figure 6: MWS-0.5 arXivDemo Search interface

is only one paper that covers this in a more presentation-search oriented setting [You06a]. To solve the problem, we have to consider the following aspects:

- what is a good measure for relevance in theory? (pagerank only applies to pages).
- how can be compute this efficiently
- can we organize the index, so that it finds the most relevant hits (as estimated by this measure) first
- is a single measure enough?

We plan to look at the following simple measures as starting points:

- pagerank by citations over the papers
- the size (whatever that means) of the substitutions (small might be beautiful)
- similarly, the size of the formulae
- popularity of the papers (by download)

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 can be reached by applying an extension (by query variables) of the LaTeX to XML transformation used on the arXiv to process queries. The new LaTeXML daemon [GSK11] allows to integrate this efficiently.

7.4 Advanced Services

Another important application of the unification search in MathWebSearch is applicable theorem search (see Figure 7 for a future use case and [KK07] for an analysis for other applications). The MathWebSearch system already supports the necessary queries (unification), but the arXiv corpus we are currently using does not have the necessary degree of formalization (explicitly marked up universal quantifications). We plan to utilize (possibly shallow) linguistic technologies to reliably analyze phrases like "*Let f and g be functions from $\mathbb{N}$ to $\mathbb{R}$...*" that mark the identifiers f and g as universal and to retrieve the associated sortal restrictions. Note that the linguistic capabilities of the variable spotter have to be considerable to detect the difference between "*... where c is a natural number*" and "*... where x is the number between 1 and n, such that...*" (only c universal) or to detect that universals in a negative scope are indeed existential.

References

[ABC+10] Ron Ausbrooks, Stephen Buswell, David Carlisle, Giorgi Chavchanidze, Stéphane Dalmas, Stan Devitt, Angel Diaz, Sam Dooley, Roger Hunter, Patrick Ion, Michael Kohlhase, Azzeddine Lazrek, Paul Libbrecht, Bruce Miller, Robert Miner, Murray Sargent, Bruce Smith, Neil Soiffer, Robert Sutor, and Stephen Watt. Mathematical Markup Language (MathML) version 3.0. W3C Recommendation, World Wide Web Consortium (W3C), 2010.

[arX] arXMLiv build system. `http://arxmliv.kwarc.info`. seen May 2010.

[Ber09] Berkeley DB. available at `http://www.oracle.com/technology/products/berkeley-db/`, 2009. seen January.

[BF06] Jon Borwein and William M. Farmer, editors. *Mathematical Knowledge Management (MKM)*, number 4108 in LNAI. Springer Verlag, 2006.

[BPSM97] Tim Bray, Jean Paoli, and C. M. Sperberg-McQueen. Extensible Markup Language (XML). W3C recommendation, World Wide Web Consortium (W3C), December 1997.

[Gra96] Peter Graf. *Term Indexing*. Number 1053 in LNCS. Springer Verlag, 1996.

[GSK11] Deyan Ginev, Heinrich Stamerjohanns, and Michael Kohlhase. The LaTeXML daemon: Editable math on the collaborative web. In James Davenport, William Farmer, Florian Rabe, and Josef Urban, editors, *Intelligent Computer Mathematics*, number 6824 in LNAI. Springer Verlag, 2011. in press.

[KK07] Andrea Kohlhase and Michael Kohlhase. *Re*examining the MKM Value Proposition: From Math Web Search to Math Web *Re*Search. In Kauers et al. [KKMW07], pages 266–279.

[KKMW07] Manuel Kauers, Manfred Kerber, Robert Miner, and Wolfgang Windsteiger, editors. *MKM/Calculemus*, number 4573 in LNAI. Springer Verlag, 2007.

John, a trainee electronics engineer, has developed an n-bit adder circuit[a] $\mathrm{CSA}(n)$ and has determined the following recursive equations for the cost $\mu(\mathrm{CSA}(n))$ of the circuit:

$$\mu(\mathrm{CSA}(n)) = 3\mu(\mathrm{CSA}(n/2)) + \mu(\mathrm{MUX}(n/2+1)) \qquad \mu(\mathrm{CSA}(1)) = 5 \tag{3}$$

For solving this equation one can either guess a closed formula for the solution and prove it by induction, or use a general tool for solving recursive equations. As our engineer does not know the literature on recurrence equations, he uses the APPLICABLE THEOREMS SEARCH service (ATS) available on the MATHWEBSEARCH platform he is using to author the semantically enhanced specification and documentation of the circuit. As the recursive equations in (3) are already in the system, they can be directly referenced to effortlessly build the search sequent

$$\langle\!\langle 3 \rangle\!\rangle \vdash \mu(\mathrm{CSA}(n)) = \boxed{\mathrm{R}}(n)$$

where the $\boxed{\mathrm{R}}$ is a search variable, which is used for returning the desired result. The ATS service returns a hit from the community portal `planetmath.org`:

Theorem (Master's Theorem for Recursive Equations)
Given the recursively defined function $f : \mathbb{N} \to \mathbb{N}$, such that $f(1) = c \in \mathbb{N}$ and $f(b^k) = af(b^{k-1}) + g(b^k)$ for some $a \in \mathbb{N}$, $1 \leq a$, $k \in \mathbb{N}$, and $g : \mathbb{N} \to \mathbb{N}$, then $f(b^k) = ca^k + \Sigma_{i=0}^{k-1} a^i g(b^{k-i})$.

together with the substitution

$$\sigma = \left\{ \begin{array}{ll} a \mapsto 3, c \mapsto 5, f \mapsto \lambda x.\mu(\mathrm{CSA}(x)), g \mapsto \lambda x.\mu(\mathrm{MUX}(x)) + 1, & \text{instantiates the theorem} \\ \boxed{\mathrm{R}} \mapsto \lambda k.5 \cdot 3^k + \Sigma_{i=0}^{k-1} 3^i \cdot \mu(\mathrm{MUX}(b^{k-i})) & \text{returns the result} \end{array} \right.$$

which John does not quite understand. So he decides to read up on this and invokes the GUIDED TOURS service which (based on the semantic dependency relation embedded in the new, MATHWEBSEARCH-powered `planetmath.org`) generates a personalized course leading up to Master's theorem. Based on this information John decides that it will solve his problem, and the ATSAPPLY service gives him the desired solution

$$\mu(\mathrm{CSA}(n)) = 5 \cdot 3^n + \Sigma_{i=0}^{n-1} 3^i \cdot \mu(\mathrm{MUX}(b^{n-i}))$$

So he accepts the ATS option to apply the substitution to his sentence "the cost of manufacturing $\mathrm{CSA}(n)$ is \$ $\boxed{\mathrm{R}}(n)$". From here on the solution only requires arithmetic simplifications, using that $\mu(\mathrm{MUX}(n)) = \$3n + 1$. Our engineer gladly leaves those to the CASE SIMPLIFICATION service, which the MATHWEBSEARCH platform offers via the icon menu associated with the formula. Similarly, he can convert the currency from \$ to € by a separate UNIT CONVERSION service.
To determine $\mu(\mathrm{CSA}(n))$ the community-supplied information in `planetmath.org` was sufficient; otherwise John would have used the "premium semantic edition" of the Elsevier Electronic Collection of Journals, to which his company subscribes.

[a] For concreteness we take the well-known conditional sum adder, which is defined by a recursion on the width of the argument vector from the one-bit adder and the n-bit multiplexer MUX.

Figure 7: Future Use Case: John, a Trainee Electronics Engineer

[Koh06] Michael Kohlhase. OMDOC – *An open markup format for mathematical documents [Version 1.2]*. Number 4180 in LNAI. Springer Verlag, August 2006.

[KŞ06] Michael Kohlhase and Ioan Şucan. A search engine for mathematical formulae. In Tetsuo Ida, Jacques Calmet, and Dongming Wang, editors, *Proceedings of Artificial Intelligence and Symbolic Computation, AISC'2006*, number 4120 in LNAI, pages 241–253. Springer Verlag, 2006.

[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.

[Mat] Math Web Search. `https://trac.mathweb.org/MWS/`. seen Jan. 2011.

[Mic] GNU MicroHTTPd library. `http://www.gnu.org/software/libmicrohttpd/`. seen Jul 2011.

[Miz] Mizar mathematical library. Web Page at `http://www.mizar.org/library`. seen May 2008.

[MM06] Rajesh Munavalli and Robert Miner. Mathfind: a math-aware search engine. In *SIGIR '06: Proceedings of the 29th annual international ACM SIGIR conference on Research and development in information retrieval*, pages 735–735, New York, NY, USA, 2006. ACM Press.

[NK07] Immanuel Normann and Michael Kohlhase. Extended formula normalization for ϵ-retrieval and sharing of mathematical knowledge. In Kauers et al. [KKMW07], pages 266–279.

[POS88] IEEE POSIX, 1988. ISO/IEC 9945.

[SK08] Heinrich Stamerjohanns and Michael Kohlhase. Transforming the arxiv to XML. In Serge Autexier, John Campbell, Julio Rubio, Volker Sorge, Masakazu Suzuki, and Freek Wiedijk, editors, *Intelligent Computer Mathematics*, number 5144 in LNAI, pages 574–582. Springer Verlag, 2008.

[SS] Geoff Sutcliffe and Christian Sutner. The TPTP problem library for automated theorem proving. web page at `http://www.tptp.org`. seen February 2008.

[Vei] Daniel Veillard. The XML c parser and toolkit of gnome; libxml. System Home page at http://xmlsoft.org.

[You06a] Abdou Youssef. Methods of relevance ranking and hit-content generation in math search. In Borwein and Farmer [BF06], pages 393–406.

[You06b] Abdou Youssef. Roles of math search in mathematics. In Borwein and Farmer [BF06], pages 2–16.