Faceted Search for Mathematics

by

Radu Hambasan

Bachelor Thesis in Computer Science

Prof. Michael Kohlhase

Supervisor

Date of Submission: May 9, 2015

Jacobs University — School of Engineering and Science

Abstract

Faceted search represents one of the most practical ways to browse a large corpus of information. Information is categorized automatically for a given query and the user is given the opportunity to further refine his/her query. Many search engines offer a powerful faceted search engine, but only on the textual level. Faceted Search in the context of Math Search is still unexplored territory.

In this thesis, I describe one way of solving the faceted search problem in math: by extracting recognizable formula schemata from a given set of formulae and using these schemata to divide the initial set into formula classes. Also, I provide a direct application by integrating this solution with existing services.

Contents

1 Introduction

The size of digital data has been growing tremendously since the invention of the Internet. Today, the ability to quickly search for relevant information in the vast amount of knowledge available is essential in all domains. As a consequence, search engines have become the prevalent tool for exploring digital data.

Although text search engines (e.g. Google [6] or DuckDuckGo [3]) seem to be sufficient for the average user, they are limited when it comes to finding scientific content. The limitation arises because STEM[1] documents are also relevant for the mathematical formulae they contain and math cannot be properly indexed by a textual search engine. Math comprises of tokens that are expressed as structural markup (fractions, square-roots, subscripts and superscripts), which are not captured by simply indexing the text content of a page.

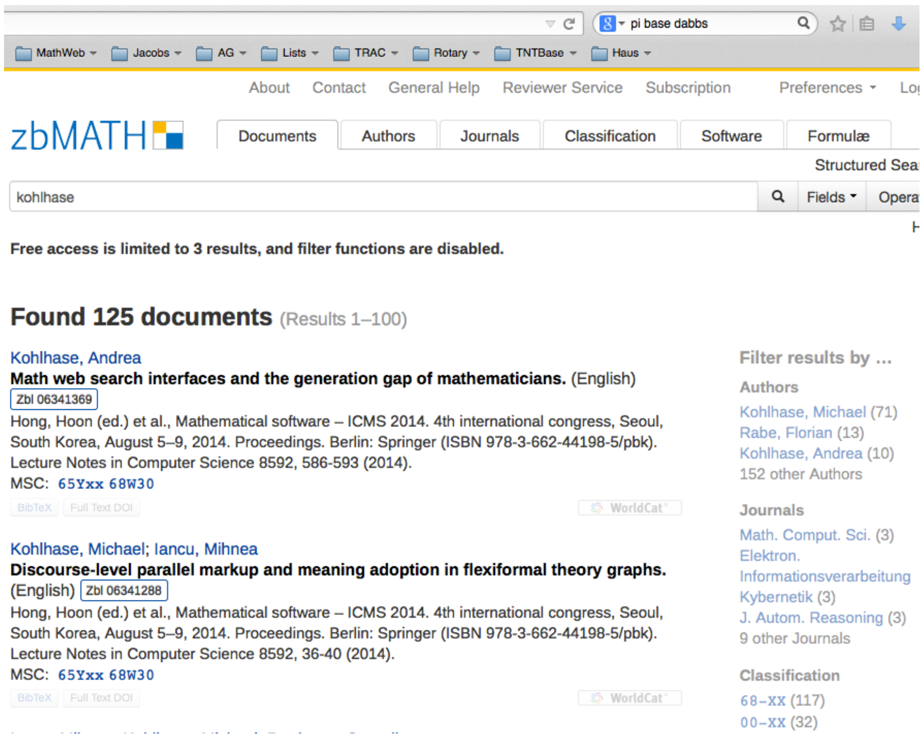
A good math search engine is therefore needed in several applications. For example, a large airline manufacturer may have many ongoing research projects and could significantly improve efficiency if they had a way of searching for formulae in a corpus containing all their previous work in the fields of physics and mathematics. The same holds for all large physics-oriented research centers, such as CERN. Valuable time would be saved if scientists would have a fast, reliable and powerful math search engine to analyse previous related work. As a third application, university students should be mentioned. Their homework, research and overall study process would be facilitated once they are provided with more than textual search. For all these applications, we first need a strong math search engine and second, a large corpus of math to index.

The Cornell e-Print Archive, arXiv, is an example of such a corpus, containing over a million STEM documents from various scientific fields (Physics, Mathematics, Computer Science, Quantitative Biology, Quantitative Finance and Statistics) [1]. Given such a high number of documents, with several million formulae, the search engine must provide an expressive query language and query-refining options to be able to retrieve useful information. One service that provides both of these is the Zentralblatt Math service [19].

Zentralblatt Math now employs formula search for access to mathematical reviews [9]. Their database contains over 3 million abstract reviews spanning all areas of mathematics. To explore this database they provide a powerful search engine called “structured search”. This engine is also capable of faceted search. Figure 1 shows a typical situation: a user searched for a keyword (here an author name) and the faceted search generated links for search refinements (the **facets**) on the right. Currently, facets for the primary search dimensions are generated – authors, journals, MSC[2], but not for formulae. In this way, the user is given the ability to further explore

[1] Science, Technology, Engineering and Mathematics
[2] Mathematics Subject Classification

the result space, without knowing in advance the specifics of what he/she is looking for. Recently, formula search has been added as a component to the structured search facility. However, there is still no possibility of faceted search on the math content of the documents.



Free access is limited to 3 results, and filter functions are disabled.

Found 125 documents (Results 1–100)

Kohlhase, Andrea
Math web search interfaces and the generation gap of mathematicians. (English) Zbl 06341369
Hong, Hoon (ed.) et al., Mathematical software – ICMS 2014. 4th international congress, Seoul, South Korea, August 5–9, 2014. Proceedings. Berlin: Springer (ISBN 978-3-662-44198-5/pbk). Lecture Notes in Computer Science 8592, 586-593 (2014).
MSC: 65Yxx 68W30

Kohlhase, Michael; Iancu, Mihnea
Discourse-level parallel markup and meaning adoption in flexiformal theory graphs. (English) Zbl 06341288
Hong, Hoon (ed.) et al., Mathematical software – ICMS 2014. 4th international congress, Seoul, South Korea, August 5–9, 2014. Proceedings. Berlin: Springer (ISBN 978-3-662-44198-5/pbk). Lecture Notes in Computer Science 8592, 36-40 (2014).
MSC: 65Yxx 68W30

Filter results by ...

Authors
Kohlhase, Michael (71)
Rabe, Florian (13)
Kohlhase, Andrea (10)
152 other Authors

Journals
Math. Comput. Sci. (3)
Elektron. Informationsverarbeitung Kybernetik (3)
J. Autom. Reasoning (3)
9 other Journals

Classification
68-XX (117)
00-XX (32)

Figure 1: Faceted Search in ZBMath

There are multiple ways in which we could understand a “math facet”. One way would be through the MSC classification [13]. However, this would be rather vague because it will only provide information about the field of mathematics to which an article belongs. If the authors use formulae from another field in their paper, the results will suffer a drop in relevance.

I am attempting to solve this problem by extracting formula schemata from the query hits, as formula facets. A math facet consists of a set of formula schemata generated to further disambiguate the query by refining it in a new dimension. For instance, for the query above we could have the formulae in Figure 2, which allows the user to drill in on *i*) variation theory and minimal surfaces, *ii*) higher-order unification, and *iii*) type theory. Following the MWS (see 2.2) tradition, the red identifiers stand for query variables, their presence making the results **formula schemata**.

$$\int_M \Phi(d_p f) dvol$$

$$\lambda X.h(H^1 X)\cdots H^n X$$

$$\frac{\Gamma \vdash A \gg \alpha}{D}$$

Figure 2: formula facets

These formula schemata were manually created to judge the feasibility of using schemata as recognizable user interface entities, but for an application we need to generate them automatically from the query. Moreover, each schema should further expand to show the formula class it represents. Formula classes would consist of all formulae sharing the same schema. This is the algorithmic problem I explore in the thesis.

2 Preliminaries

In this section we describe the existent systems on which our work will be based, with the intention of making this thesis self-contained. We will present these systems in detail in the rest of this section, but below is a summary of the role they play in our work:

- **MathWebSearch** provides the necessary index structure for schema search.
- **Elasticsearch** provides hits in response to text queries, as well as run aggregations on the hits. These hits represent formulae to be schematized.
- **arXiv** provides a large corpus of mathematical documents that we can index and run our system on.
- **LaTeXML** converts LaTeX expressions to MathML.

2.1 Project Goals

As discussed in Section 1, the goal of this project is to develop a scalable formula schematization engine, capable of dividing a set of query hits into classes, according to the generated formula schemata.

We have set the following end-user requirements for our system:

R1. it should be able to generate formula schemata from a given set of formulae and the resulting schemata should be easily recognizable by the user.

R2. it should be able to classify the given set of formulae according to the generated schemata.

R3. the system should be massively scalable, i.e. capable of answering queries with hundreds of thousands of formulae in a matter of seconds.

2.2 MathWebSearch

At its core, the MathWebSearch [17] system (MWS) is a content-based search engine for mathematical formulae. It indexes MathML [12] formulae, using a technique derived from automated theorem proving: Substitution Tree Indexing [7]. Recently, it was augmented with full-text search capabilities, combining keyword queries with unification-based formula search. The engine serving text queries is Elasticsearch 2.3. From now on, in order to avoid confusion, we will refer to the core system (providing just formula query capability) as MWS and to the complete service (MWS + Elasticsearch) as TeMaSearch (Text + Math Search).

The overall workflow of TeMaSearch is the following:

1. HTML5 documents representing mathematical articles are crawled to generate MWS harvests [15]. The harvest format is an extension of MathML which MWS can index. Its role is to separate the math from the text in a given document.

2. MWS indexes the harvests.

3. a second pass is made over the harvests to generate annotated documents (see below).

4. Elasticsearch indexes the annotated documents.

5. Everything is now ready for answering queries. When a query is issued, MWS will answer the mathematical part and Elasticsearch will answer the text part. The results are combined through a NodeJS [16] proxy to send a final result set.

Each mathematical expression is encoded as a set of substitutions based on a depth-first traversal of its Content MathML tree. Furthermore, each tag from the Content MathML tree is encoded as a TokenID, to lower the size of the resulting index. The (bijective) mapping is also stored together with the index and is needed to reconstruct the original formula. The index itself is an in-memory trie of substitution paths.

To facilitate fast retrieval, MWS stores FormulaIDs in the leaves of the substitution tree. These are integers uniquely associated with formulae, and they are used to store the context in which the respective expressions occurred. These identifiers are stored in a separate LevelDB [10] database.

Figure 3 shows the core components of MWS and the relationship between them. The most important component is the index which stores the encoded DFS traversal of the formulae from the corpus. This encoding (from Content MathML elements to TokenIDs) is handled by the Meaning Dictionary. As mentioned, each leaf of the index-trie has an associated FormulaID, which points to the FormulaDB. FormulaDB includes information about the XPath and the source documents of every FormulaID. Finally, there is a CrawlDB database which stores crawled opaque data, which usually represents the textual part of the source documents.

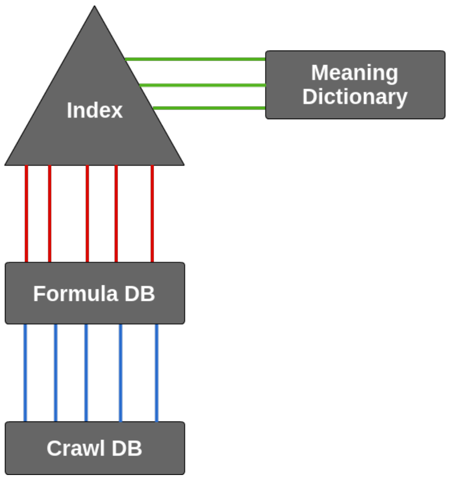


Figure 3: MWS Core Components

We can exemplify the role played by the MWS components using Listing 1, which shows a simple harvest containing some opaque data ("*Addition of variables.*") and the expression $a + b$. The data element is referenced by the expression using the data_id attribute (which has the value 0 in this example). If MWS indexes this harvest, its content will be split among the components in the following manner:

- The Meaning Dictionary will contain the encoding $\{(0, \text{apply}), (1, \text{plus}), (2, \text{ci\#a}), (3, \text{ci\#b})\}$. This means that the TokenID 0 will be decoded as an "apply" node, the TokenID 2 will be decoded as a "ci" node with "a" as content, etc.
- The Index will be a tree with one branch containing the encoded DFS traversal of the expression, which is "0 -> 1 -> 2 -> 3". The leaf of this branch will have a LeafID of 1.
- The Formula DB will contain a mapping from 1 (the LeafID of the indexed expression) to $[\text{"\#m1.1"}, 0]$. The first element of the pair represents the URI of the formula. The second element represents the data_id of the opaque data which was referenced by the formula.
- The Crawl DB will link the key 0 to the opaque data that was associated with the data_id 0. In this case, the Crawl database would contain the key-value pair $(0, \text{"Addition of variables"})$.

Listing 1: Harvest Example

```
<mws:harvest>
    <mws:data mws:data_id="0">
        Addition of variables.
    </mws:data>
    <mws:expr url="#m1.1" mws:data_id="0">
        <apply>
            <plus>
            <ci>a</ci>
            <ci>b</ci>
        </apply>
    </mws:expr>
</mws:harvest>
```

MathWebSearch exposes a RESTful HTTP API which accepts XML queries. A valid query must obey the Content MathML format, potentially augmented with *qvar* variables which match any subterms. A *qvar* variable acts as a wildcard in a query, with the restriction that if two *qvar*s have the same name, they must be substituted in the same way.

TeMaSearch is using both systems (MathWebSearch and Elasticsearch) to answer queries. To achieve cooperation between the two components, annotated documents are used. These annotated documents contain metadata from the original document (e.g. URI, title, author, etc.) and a list of FormulaIDs that can be found in that document.

2.3 Elasticsearch

Elasticsearch [4] is a powerful and efficient full text search and analytics engine, built on top of Lucene. It can scale massively, because it partitions data in shards and is also fault tolerant, because it replicates data. It indexes schema-free JSON documents and the search engine exposes a RESTful web interface. The query is also structured as JSON and supports a multitude of features via its domain specific language: nested queries, filters, ranking, scoring, searching using wildcards/ranges and faceted search.

The faceted search feature[3] is of particular interest to us. One way to use this feature is the terms aggregation: a multi-bucket aggregation, with dynamically built buckets. We can specify an array field from a document and ask ES to count how many unique items from the array are there in the whole index. This list can also be sorted, e.g. most frequently occurring items first. Additionally, we can also impose a limit on the number of the buckets (items) for which we want to receive the count.

[3]Faceted search as such is now deprecated in ES and was replaced by the more powerful "aggregations" feature, which also allows facets of facets, i.e. nested facets.

```
{
  "query" : {
      "match" : {
          "body" : {
              "query" : "Pierre Fermat",
              "operator" : "and"
          }
      }
  },
  "aggs" : {
      "formulae" : {
          "terms" : { "field" : "ids" }
      }
  }
}
```

Listing 2: Elastic Search Term Aggregation Query

An ES query which would return the most frequently used formulae (and subformulae) for "Pierre Fermat", is presented in Listing 2. The key part is the *aggs* fields. We are specifying that we want an aggregation called *Formulae* on "terms" (i.e. we want bucket counting) and the target of the aggregation is the fields *ids*.

A possible response to the above query can be found in Listing 3. In the response we can see the returned aggregations. In our example there is only one and it is called *formulae*. We can find the actual result in the *buckets* field. The key field in the bucket corresponds to a FormulaID. Here, the most frequent formulae were the one with ID 230 and the one with ID 93. The former appeared in 10 documents and the latter appeared in 9 documents.

```
{
    ...
    "aggregations" : {
        "formulae" : {
            "buckets" : [
                {
                    "key" : "230",
                    "doc_count" : 10
                },
                {
                    "key" : "93",
                    "doc_count" : 9
                },
                ...
            ]
        }
    }
}
```

Listing 3: Elastic Search Term Aggregation Response

2.4 arXiv

arXiv is a repository of over one million publicly accessible scientific papers in STEM fields. For the NTCIR-11 challenge [8], MWS indexed over 8.3 million paragraphs (totaling 176 GB) from arXiv. We will base our queries on this large index, because it provides a rich database of highly relevant formulae. Moreover, Elasticsearch will have more formulae on which it can run aggregations, also leading to more relevant results.

2.5 LaTeXML

An overwhelming majority of the digital scientific content is written using LaTeX or TeX, due to its usability and popularity among STEM researchers. However, formulae in these formats are not good candidates for searching because they do not display the mathematical structure of the underlying idea. For this purpose, conversion engines have been developed to convert LaTeX expressions to more organized formats such as MathML.

An open source example of such a conversion engine is LaTeXML [14]. The MathWebSearch project relies heavily on it, to convert arXiv documents from LaTeX to XHTML which is later indexed by MWS. It exposes a powerful API, accepting custom definition files which relate TeX elements to corresponding XML fragments that should be generated. For the scope of this project, we are more interested in another feature of LaTeXML: cross-referencing between Presentation MathML and Content MathML. While converting TeX entities to Presentation MathML trees, LaTeXML assigns each PMML element a unique identifier which is later referenced from the corresponding Content MathML element. In this manner, we can modify the Content MathML tree and reflect the changes in the Presentation MathML tree which can be displayed to the user.

Figure 4 illustrates the parallel markup for $\frac{2}{x+3}$. On the left side we have Presentation MathML and on the right side, Content MathML. As we can see, every Content element has a Presentation correspondent, except the divide operator, whose meaning is reflected in the structure of the displayed formula.

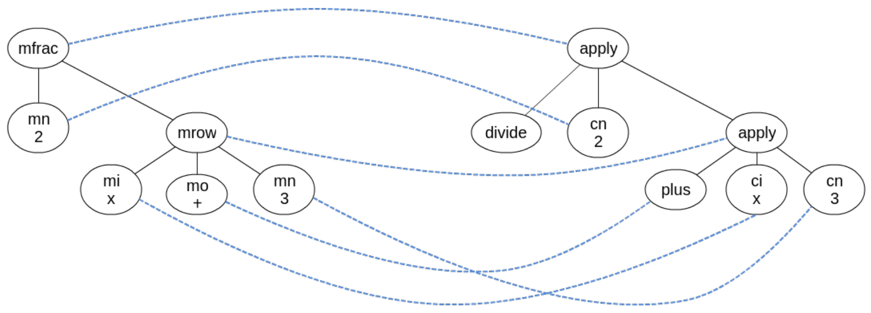


Figure 4: The CMML/PMML Parallel Markup

3 Schematization of Formula Sets

In this section, I provide a theoretical approach to the problem of generating formula schemata, by formalizing the problem and describing an efficient algorithm to solve it.

3.1 Formalizing the Problem

Let us now formulate the problem at hand more carefully.

Definition 1 *Given a set $\mathcal{D}$ of documents (fragments) – e.g. generated by a search query, a* ***coverage*** $0 < r \leq 1$*, and a* ***width*** n*, the* ***Formula Schemata Generation*** *(FSG) problem requires generating a set $\mathcal{F}$ of at most n formula schemata (content MathML expressions with* qvar *elements for query variables), such that $\mathcal{F}$ covers $\mathcal{D}$ with coverage r.*

Definition 2 *We say that a set $\mathcal{F}$ of formula schemata* ***covers*** *a set $\mathcal{D}$ of document fragments, with* ***coverage*** *r, iff at least $r \cdot |\mathcal{D}|$ formulae from $\mathcal{D}$ are an instance $\sigma(f)$ of some $f \in \mathcal{F}$ for a substitution σ.*

3.2 An Algorithm for FSG

The algorithm that I present requires a MWS index of a corpus. Given such an index, and a set $\mathcal{D}$ of formulae (as CMML expressions), we can find the set $\mathcal{F}$ in the following way:

1. Parse the given CMML expressions similarly to MWS queries, to obtain their encoded DFS representations.
2. Choose a reasonable cutoff heuristic, see 3.3.
3. Unify each expression with the index, up to a given threshold (given by the above heuristic).
4. Keep a counter for every index path associated with the unifications. Since we only match up to a threshold, some formulae will be associated with the same path (excluding the leaves). We increase the counter each time we find a path already associated with a counter.
5. Sort these path-counter pairs by counter in descending order and take the first n (n being the width required by the FSG).
6. If the threshold depth was smaller than a formula's expression depth, the path associated with it will have missing components. We replace the missing components with qvars to generate the schema and return the result set.

Figure 5 shows a simplified MWS index at depth 1. The formulae's paths represent their depth-first traversal. Every formula can be reconstructed given its path in the index. The circles represent index nodes and the number inside represents the token's ID. When we reach a leaf node, we completely described a formula. This is encoded in the leaf node by an ID, which can be used to retrieve the formula from the database. The length of the arrows symbolizes the depth of the omitted subterms (for higher depths, we have longer arrows). Notice how both formula with ID 1 and formula with ID 3 show the same "path" when ignoring subterms below a cutoff depth.

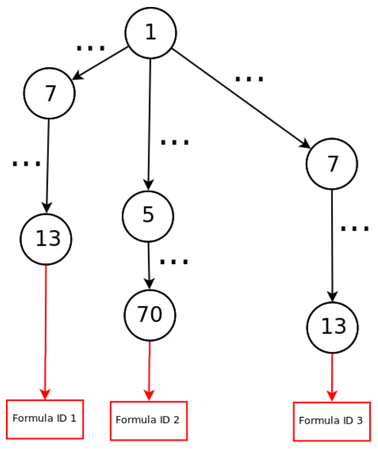


Figure 5: Simplified Index at depth 1

3.3 Finding a Cutoff Heuristic

To generate formula schemata, we must define a "cutoff heuristic", which tells the program when two formulae belong to the same schema class. If there is no heuristic, two formulae would belong to the same class, only if they were identical. However, we want formulae that have something in common to be grouped together, even if they are not perfectly identical.

3.3.1 Absolute Cutoff

One reasonable cutoff heuristic would be a fixed expression depth, given as a parameter to the Schematizer. In this way, a depth of 0 would always return $?x$ which corresponds to the 0-unification. Figure 6 illustrates the absolute cutoff heuristic. On the left side, we have the initial Content MathML representation for the formula $\frac{2}{x+3}$. Using an absolute cutoff depth of 1, we get the CMML representation of the schema shown on the right side.

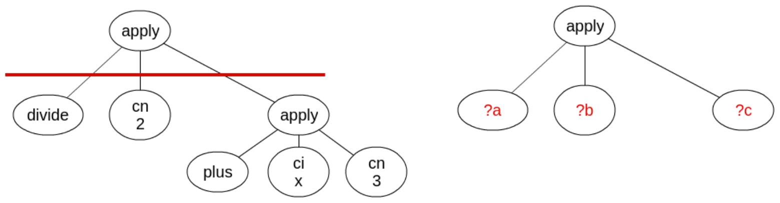


Figure 6: Cutoff at a Fixed Depth

Using this heuristic, we are able to group together formulae which share a certain portion of their Content MathML representation and return this shared portion as the group's schema. However, in practice this approach poses several disadvantages. By using a uniform cutoff for a given CMML tree, we might also remove the operator of an expression (first child of the apply node). This leads to unintuitive instantiations from the user point of view. For example, the schema $?x + ?y$ will also include $\frac{a}{b}$ in its class, which is not desired since we only want to abstract out the operands of an expression.

3.3.2 Relative Cutoff

A slight improvement over the absolute cutoff heuristic would be to choose the cutoff depth depending on the depth of the CMML tree. For this, the user would specify a relative cutoff, e.g. 50%, and each tree would be truncated at that relative depth.

On an individual basis, we are still using a uniform cutoff, so this approach also poses all disadvantages outlined in the previous section. The main advantage of this method stems from the ability to differentiate between complex and trivial formulae. If the cutoff were absolute, for low depths, complex formulae would be truncated into simple schemata. This would mean that the user has the choice to either see only the simple schemata, or only the complex ones. By using the relative heuristic, both complex and simple schemata would share the result set.

3.3.3 Dynamic Cutoff

As we have already seen, a uniform cutoff heuristic is generally not the optimal choice as it also removes operators. We would like to "schematize" an expression by abstracting out only the operands, while leaving the operators untouched. To do this, we can completely expand the first child of an apply tag and apply the cutoff depth (either absolute or relative) only to the other nodes. Figure 7 illustrates this heuristic at depth 1. The divide element was kept, because it was the first child of apply, while the other children were removed. If we were to use a depth of 2, the plus element would also be included in the schema.

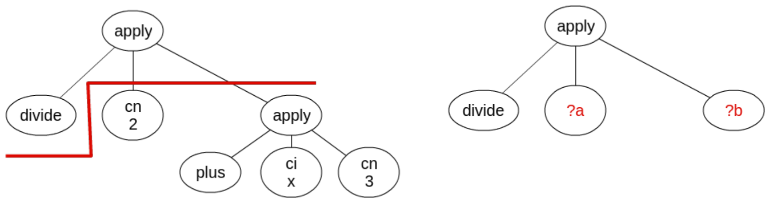


Figure 7: Dynamic Cutoff

Although this method seems to yield recognizable schemata and classes with intuitive instantiations, it is relatively difficult to implement because the operator might occupy several levels (not just one, as shown in the figure) and we must do the expansion of the first child of apply directly using the DFS traversal of the CMML tree.

3.3.4 Coverage aware Cutoff

To report optimal results to the user, we could try a "brute-force" approach, in which we compute the schematization for a large range of fixed depths, and then select schemata with the *best* coverage from all the results. The best coverage should not be too large (because this could mean it is a trivial schema) and also not too small (because this could mean the schema is not important). Consequently, we will choose several cutoffs for a given query. However, this method is very computationally expensive and defeats the R3 goal of this project.

3.3.5 Length based Cutoff

Another possibility is to place the cutoff on length, instead of depth. This method would group together formulae which begin the same way. To implement a length based cutoff, we generate the DFS encoding of a formula and then take the first n tokens, where n is a given cutoff. Despite being relatively easy to implement and fast to compute, this heuristic would generate results which are not as interesting as those generated by a depth based heuristic (because the former will only show the first few elements in a formula, whereas the latter will also show the last tokens).

3.3.6 Combining Heuristics

As we have seen, there are several advantages and drawbacks associated with each heuristic. Since the coverage-aware cutoff is too expensive to compute and the length-based cutoff does not generate interesting enough results, we will ignore them when proposing the following combined heuristic. In our project we use the dynamic cutoff (thus preserving operators) and offer the user a choice on whether the operands cutoff should be absolute or relative.

4 Implementation

In this section, I explain the key details of the formula classifier's implementation, the overall system architecture, as well as the challenges and trade-offs associated with the taken design decisions.

4.1 Design Overview

The full faceted search system comprises of the following components: the Formula Schematizer 4.2, Elasticsearch, a proxy to mediate communication between the Schematizer and Elasticsearch and a Web front-end. The architecture of the system is shown in Figure 8.

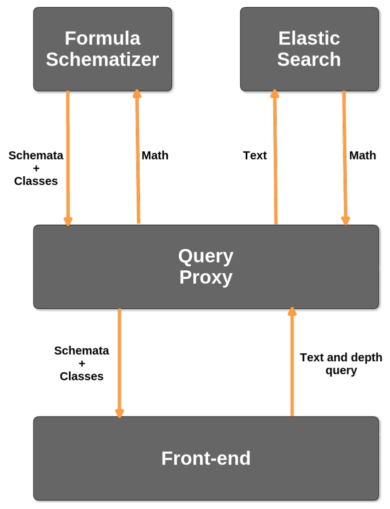


Figure 8: FS Engine Architecture

Once the user enters a query (which consists of keywords and a depth), the front-end forwards the request to a back-end proxy. The proxy sends the text component of the query to Elasticsearch and receives back math contained in matching documents. Afterwards, it sends the retrieved math and the depth parameter (from the original query) to the Schematizer. The Schematizer will respond with a classification of the math in formula classes, as well as the corresponding schema for each class. Finally, the proxy forwards the result to the front-end which displays it to the user.

In the following sections, I will explain the core components of the system in detail and describe the challenges faced during implementation.

4.2 The Formula Schematizer

The Schematizer is the central part of our system. It receives a set of formulae in their Content MathML representation, generates corresponding formula schemata and classifies the formulae according to the generated schemata. It provides an HTTP endpoint and is therefore self-contained, i.e. it can be queried independently, not only as part of the faceted search system. As a consequence, the Schematizer displays a high degree of versatility, and can be integrated seamlessly with other applications.

Although the algorithm described in Section 3.2 works well in theory, we needed to adapt it considering various MathWebSearch implementation details, e.g. the index is read-only (therefore we cannot store extra data into the index nodes). Therefore, the overall idea/theory is the same, but now we take the following shortcut: instead of unifying every formula with the index, we just pretend we do and instead generate a “signature” for each formula. This signature is the path shown in Section 3.2. We use the MathWebSearch encoding for MathML nodes, where each node is assigned an integer ID based on its tag and text content. If the node is not a leaf, then only the tag is considered. The signature will be a vector of integer IDs, corresponding to the pre-order traversal of the Content MathML tree.

Naturally, the signature depends on the depth chosen for the cutoff heuristic. At depth 0, the signature consists only of the root token of the Content MathML expression. At full depth (the maximum depth of the expression), the signature is the same as the depth-first traversal of the Content MathML tree.

Based on these computed signatures, we divide the input set of formulae into formula classes, i.e. all formulae with the same signature belong to the same class. For this operation we keep an in-memory hash table, where the keys are given by the signatures and the values are sets of formulae which have the signature key. After filling the hash table, we sort it according to the number of formulae in a given class, since the signatures which cover the most formulae should come at the beginning of the reported result.

The Schematizer caller can place an optional limit on the maximum number of schemata to be returned. If such a limit was specified, we apply it to our sorted list of signatures and take only the top ones.

As a last step, we need to construct Content MathML trees from the signatures, to be able to show the schemata as formulae to the user. We are able to do this because we know the arity of each token and the depth used for cutoff. The tree obtained after the reconstruction might be incomplete, so we insert query variables in place of missing subtrees. We finally return these Content MathML trees with query variables (the formula schemata), together with the formulae which they cover.

4.3 The Elasticsearch Proxy

To obtain formulae to feed as input to the Schematizer, we make use of Elasticsearch. The corpus indexed by Elasticsearch consists of 176 GB of arXiv documents annotated using the format described in Section 2.2.

To mediate the communication between the two components (Elasticsearch and the Schematizer), I developed a NodeJS proxy. The proxy exposes an HTTP interface, which will be used by an eventual front-end for querying. There are three important query parameters: the keywords, the depth and the maximum schemata count.

The main task of the proxy is to assemble the Elasticsearch JSON query after receiving the parameters from the front-end and forward the Elasticsearch results to the Schematizer. To ensure a fast response, only a minimal preprocessing is done between the stages.

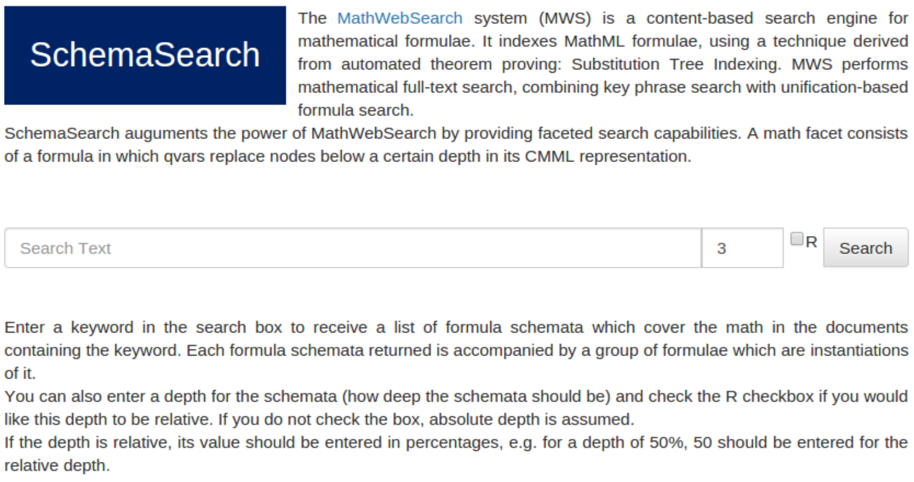
Once the Schematizer has generated schemata and has classified the formulae, the proxy assembles the results into a JSON object which it sends to the front-end.

4.4 The Front-End

To show the capabilities of the Schematizer we have prepared two demos. The first one is a text-only search engine which returns the math from the matching documents, after running it through the Schematizer. This is the demo for showcasing the schematization process. The second one is a direct application of the Schematizer into a Math Search Engine which is capable of mathematical faceted search.

4.4.1 SchemaSearch

The SchemaSearch front-end provides just a textual search input field. It is intended for users who want an overview of the formulae contained in a corpus. As shown in figure 9, the user can enter a set of keywords for the query, as well as a schema depth, which defaults to 3. The maximum result size is not accessible to the user, to prevent abuses and reduce server load. There is also an “R” checkbox which specifies if the cutoff depth should be absolute or relative. If relative, the depth should be given in percentages.


SchemaSearch

The MathWebSearch system (MWS) is a content-based search engine for mathematical formulae. It indexes MathML formulae, using a technique derived from automated theorem proving: Substitution Tree Indexing. MWS performs mathematical full-text search, combining key phrase search with unification-based formula search.

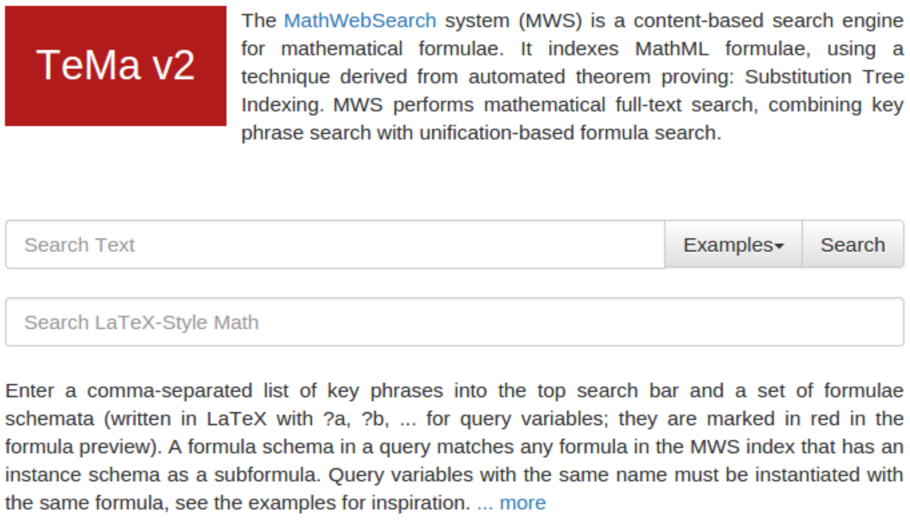
SchemaSearch auguments the power of MathWebSearch by providing faceted search capabilities. A math facet consists of a formula in which qvars replace nodes below a certain depth in its CMML representation.

Enter a keyword in the search box to receive a list of formula schemata which cover the math in the documents containing the keyword. Each formula schemata returned is accompanied by a group of formulae which are instantiations of it.
You can also enter a depth for the schemata (how deep the schemata should be) and check the R checkbox if you would like this depth to be relative. If you do not check the box, absolute depth is assumed.
If the depth is relative, its value should be entered in percentages, e.g. for a depth of 50%, 50 should be entered for the relative depth.

Figure 9: SchemaSearch Front-end

4.4.2 TemaV2

The TemaV2 front-end extends TeMaSearch to be able to perform mathematical faceted search. It is intended for users who want to filter query results based on a given facet (formula schema in this case). The look and feel is similar to the previous version of TeMaSearch, as shown in Figure 10, where the first input field is used to specify keywords and the second one is used to specify LaTeX-style formulae for the query. When returning results, a "Math Facets" menu will be presented to the user. We discuss this in Section 5.2.


TeMa v2

The MathWebSearch system (MWS) is a content-based search engine for mathematical formulae. It indexes MathML formulae, using a technique derived from automated theorem proving: Substitution Tree Indexing. MWS performs mathematical full-text search, combining key phrase search with unification-based formula search.

Enter a comma-separated list of key phrases into the top search bar and a set of formulae schemata (written in LaTeX with ?a, ?b, ... for query variables; they are marked in red in the formula preview). A formula schema in a query matches any formula in the MWS index that has an instance schema as a subformula. Query variables with the same name must be instantiated with the same formula, see the examples for inspiration. ... more

Figure 10: TemaV2 Front-end

4.5 ES Aggregations

Initially, the “aggregations” feature of Elasticsearch seemed to be a suitable improvement for the Schematizer and the ES script was originally designed to request math in a term-based aggregation format, as described in Section 2.3. However, on a closer look we can see that not only are aggregations not needed, but they influence the results in a negative way.

On a regular query, the top formulae reported using aggregations were trivial formulae, i.e. consisting of only one or two symbols. This is because authors frequently use short inline math to refer to their results. As a consequence, the top returned was completely unusable, because the first hits were irrelevant. Moreover, it was impossible to distinguish between long irrelevant expressions and infrequently used important expressions since both of them ranked the same.

As an added disadvantage of using aggregations, we must mention the time overhead. For obtaining accurate results over the entire index, several minutes were needed. In order to reach our target time of $\approx 10s$, we had to drastically shrink down the number of considered aggregations (down to 100). As mentioned before, all these 100 expressions were trivial.

Given the drawbacks mentioned above, we decided against using aggregations in the Schematizer pipeline. As an alternative, we will retrieve all math content from the documents which match the keywords and discard the trivial formulae using a configurable length heuristic. In practice, this change allowed us to process more than ten thousand formulae in less than five seconds, which is a drastic improvement from the approach using aggregations.

4.6 Presentation by Replacement

After obtaining the schemata and formula classes, we need to be able to display the result to the user. One possibility would be to have the Schematizer return Content MathML expressions for the schemata and use an XSL stylesheet [18] to convert them to Presentation MathML. This approach would unfortunately generate unrecognizable schemata due to the inherent ambiguity of CMML. For instance, a csymbol element can be represented in several different ways depending on the notation being used. Additionally, we cannot reliably foresee all possible rules that should be implemented in the stylesheet and as a consequence some formulae will be wrongly converted.

Since the XSL conversion is unreliable, we will make use of the cross-reference system provided by LaTeXML, as discussed in Section 2.5. Instead of returning Content MathML expressions, the Schematizer will use the first formula in each class as a template and “punch holes into it”, effectively returning the ID of the nodes that are to be substituted with query variables.

We will use this IDs to replace the referenced PMML nodes with `<mi>` nodes representing the qvars.

Figure 11 shows the presentation by replacement technique for a given schema. The Schematizer returned a schema which was checked against the first formula in its class ($\frac{2}{x+3}$) to generate two substitutions, marked with red on the left side. Due to the cross-reference system provided by LaTeXML, we are able to find the corresponding PMML elements and substitute them with `<mi>` tokens. The result will be displayed to the user as $\frac{?x}{?y}$.

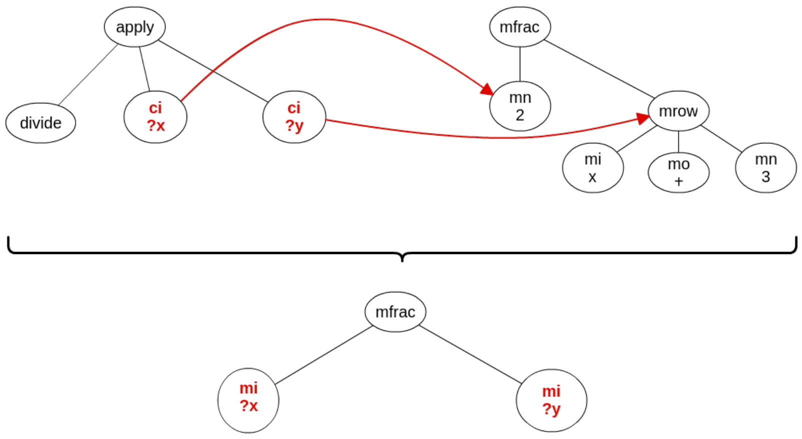


Figure 11: Presentation by Replacement

4.7 Naming Query Variables

As discussed in the section above, we will need a template to modify in order to display the schema of a class. We are using the first formula as a template, but any formula in a given class would work as a template for that class. Since we will process an existing expression, it would be appropriate to name the query variables in a way which allows the user to perceive some meaning behind them. For this reason, we conceived a naming convention with the following rules:

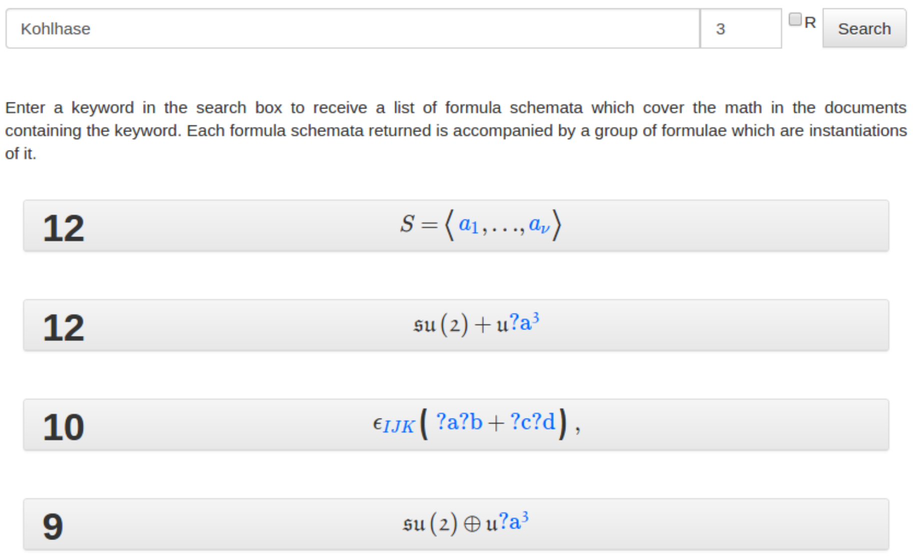
- If the node to be replaced is a leaf, its name will be used for the qvar.
- If the node to be replaced is not a leaf (thus having no name), we will use lowercase alphabetical letters from `a` to `z` preceded by a question mark (according to the MWS tradition).
- If there are more than 26 qvars (thus exceeding the alphabet), we name them $x_1, x_2, \ldots, x_{max\_count}$, also preceded by a question mark. This case should be extremely rare.

5 Evaluation

In this section, we will present an evaluation of our faceted search engine. Firstly, we will examine the 2 front-ends, obtained after applying the heuristics and improvements described in the Implementation section. Next, we will analyze the performance of the Schematizer deployed on an arXiv index (5.3). In the end, we will look at the API that the Schematizer exposes and elaborate on its potential (5.4).

5.1 SchemaSearch Front-end

Figure 12 shows the formula schemata at depth 3, over the arXiv corpus, for a query containing the keyword "Kohlhase". By default, the top 40 schemata are shown, but the results are truncated for brevity. The bold number on the left side of each result item indicates how many formulae are present in each formula class. For instance, the third schema represents a formula class containing 10 formulae. The entities marked in blue are query variables (qvars).



Enter a keyword in the search box to receive a list of formula schemata which cover the math in the documents containing the keyword. Each formula schemata returned is accompanied by a group of formulae which are instantiations of it.

12 $S = \left\langle a_1, \ldots, a_\nu \right\rangle$

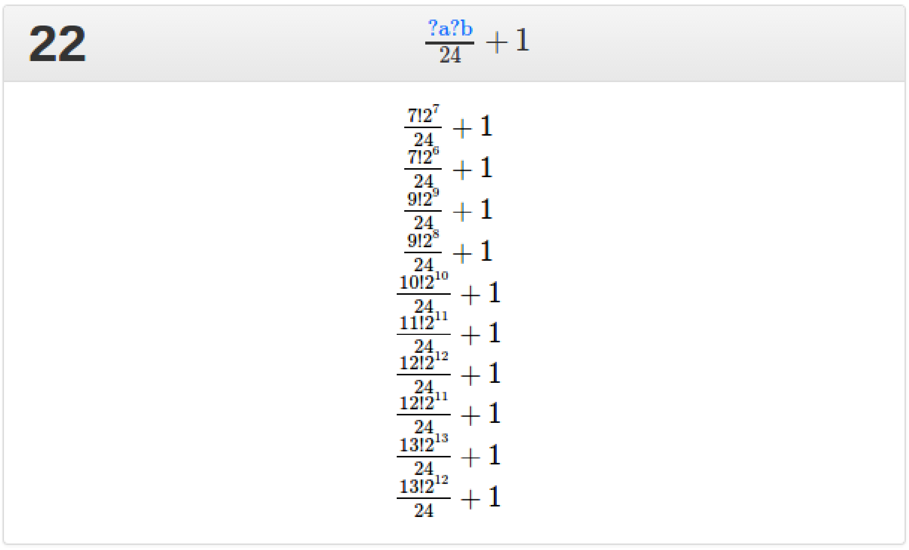
12 $\mathfrak{su}(2) + \mathfrak{u}?a^3$

10 $\epsilon_{IJK}\left(?a?b + ?c?d \right),$

9 $\mathfrak{su}(2) \oplus \mathfrak{u}?a^3$

Figure 12: Faceted Results at depth 3

Figure 13 shows the expansion of a formula class. There are 22 formulae in the class given by this particular math schema, as indicated by the count on the left upper side, out of which only ten are shown. We can see 2 unnamed query variables marked with blue as $?a$ and $?b$. By seeing the schema, the user can form an impression about the general structure of the formulae from that class. After expanding the class, the listing of concrete formulae appears. If the user clicks on one of them, he is redirected to the source document from which that expression was extracted.

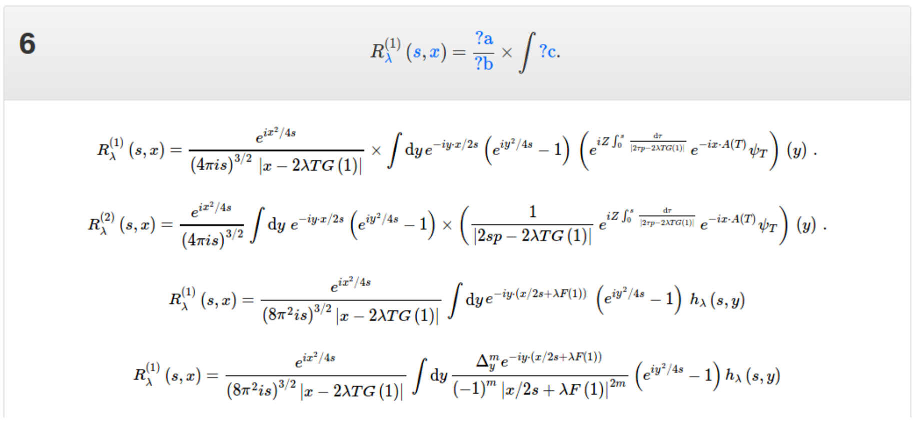

22

$$\frac{?a?b}{24}+1$$

$$\frac{7!2^7}{24}+1$$
$$\frac{7!2^6}{24}+1$$
$$\frac{9!2^9}{24}+1$$
$$\frac{9!2^8}{24}+1$$
$$\frac{10!2^{10}}{24}+1$$
$$\frac{11!2^{11}}{24}+1$$
$$\frac{12!2^{12}}{24}+1$$
$$\frac{12!2^{11}}{24}+1$$
$$\frac{13!2^{13}}{24}+1$$
$$\frac{13!2^{12}}{24}+1$$

Figure 13: Expansion of a Formula Class 1

Another class expansion which showcases the schematization can be seen in Figure 14. By seeing this schema, the user can abstract away the complexity of the formulae and obtain a "summary" of the meaning behind it. Also, by expanding the class he can explore several related formulae easily, because they are grouped together.


6

$$R_\lambda^{(1)}\left(s,x\right)=\frac{?a}{?b}\times\int ?c.$$

$$R_\lambda^{(1)}\left(s,x\right)=\frac{e^{ix^2/4s}}{\left(4\pi is\right)^{3/2}\left|x-2\lambda TG\left(1\right)\right|}\times\int \mathrm{d}y\,e^{-iy\cdot x/2s}\left(e^{iy^2/4s}-1\right)\left(e^{iZ\int_0^s\frac{\mathrm{d}\tau}{|2\tau p-2\lambda TG(1)|}}e^{-ix\cdot A(T)}\psi_T\right)(y)\;.$$

$$R_\lambda^{(2)}\left(s,x\right)=\frac{e^{ix^2/4s}}{\left(4\pi is\right)^{3/2}}\int \mathrm{d}y\;e^{-iy\cdot x/2s}\left(e^{iy^2/4s}-1\right)\times\left(\frac{1}{\left|2sp-2\lambda TG\left(1\right)\right|}e^{iZ\int_0^s\frac{\mathrm{d}\tau}{|2\tau p-2\lambda TG(1)|}}e^{-ix\cdot A(T)}\psi_T\right)(y)\;.$$

$$R_\lambda^{(1)}\left(s,x\right)=\frac{e^{ix^2/4s}}{\left(8\pi^2 is\right)^{3/2}\left|x-2\lambda TG\left(1\right)\right|}\int \mathrm{d}y\,e^{-iy\cdot(x/2s+\lambda F(1))}\left(e^{iy^2/4s}-1\right)h_\lambda\left(s,y\right)$$

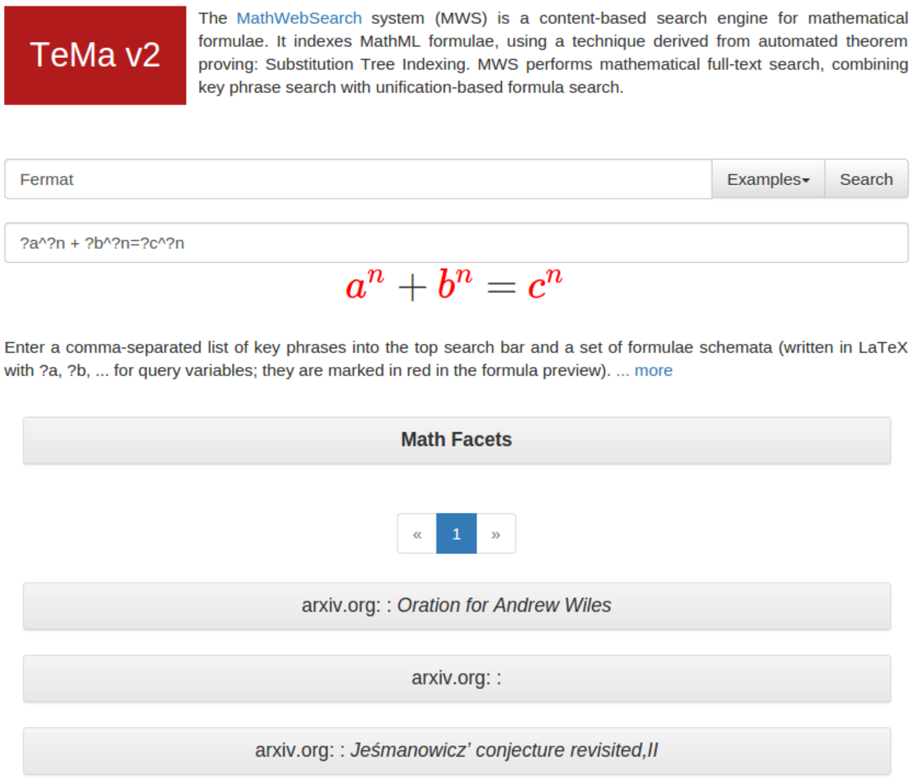
$$R_\lambda^{(1)}\left(s,x\right)=\frac{e^{ix^2/4s}}{\left(8\pi^2 is\right)^{3/2}\left|x-2\lambda TG\left(1\right)\right|}\int \mathrm{d}y\,\frac{\Delta_y^m e^{-iy\cdot(x/2s+\lambda F(1))}}{(-1)^m\left|x/2s+\lambda F\left(1\right)\right|^{2m}}\left(e^{iy^2/4s}-1\right)h_\lambda\left(s,y\right)$$

Figure 14: Expansion of a Formula Class 2

5.2 TemaV2 Front-end

Figure 15 shows the results of a query for "Fermat" and ${?a}^{?n}+{?b}^{?n}={?c}^{?n}$. Besides the regular TeMaSearch results, the user is also presented with a

“Math Facets” section.


TeMa v2

The MathWebSearch system (MWS) is a content-based search engine for mathematical formulae. It indexes MathML formulae, using a technique derived from automated theorem proving: Substitution Tree Indexing. MWS performs mathematical full-text search, combining key phrase search with unification-based formula search.

Fermat

?a^?n + ?b^?n=?c^?n

$$a^n + b^n = c^n$$

Enter a comma-separated list of key phrases into the top search bar and a set of formulae schemata (written in LaTeX with ?a, ?b, ... for query variables; they are marked in red in the formula preview). ... more

Math Facets

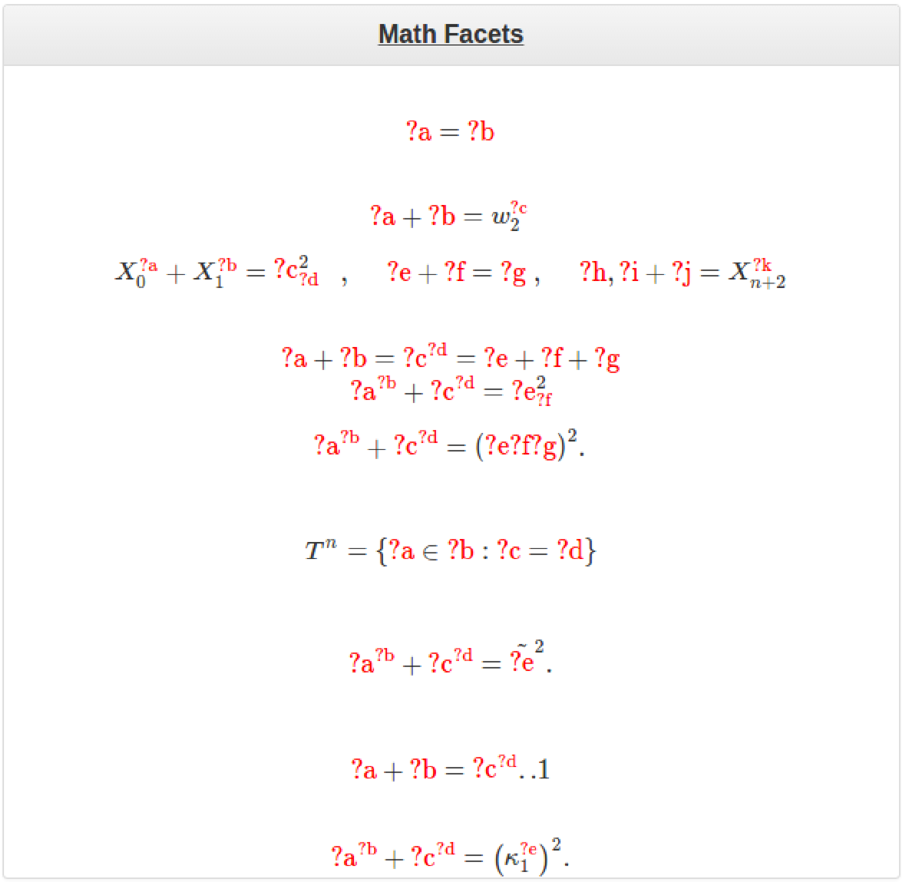
arxiv.org: : *Oration for Andrew Wiles*

arxiv.org: :

arxiv.org: : *Jeśmanowicz’ conjecture revisited,II*

Figure 15: TeMa v2 Query Results

When the “Math Facets” section is expanded the user can see the top 10 schemata (ranked with respect to their coverage), as shown in Figure 16. We have also implemented a “search-on-click” functionality that allows the user the do a fresh search using the clicked schema and the initial keyword, which effectively filters the current results.



$$?a = ?b$$

$$?a + ?b = w_2^{?c}$$

$$X_0^{?a} + X_1^{?b} = ?c^2_{?d} \quad , \quad ?e + ?f = ?g \, , \quad ?h, ?i + ?j = X^{?k}_{n+2}$$

$$?a + ?b = ?c^{?d} = ?e + ?f + ?g$$

$$?a^{?b} + ?c^{?d} = ?e^2_{?f}$$

$$?a^{?b} + ?c^{?d} = (?e?f?g)^2.$$

$$T^n = \{?a \in ?b : ?c = ?d\}$$

$$?a^{?b} + ?c^{?d} = \tilde{?e}^2.$$

$$?a + ?b = ?c^{?d}..1$$

$$?a^{?b} + ?c^{?d} = (\kappa_1^{?e})^2.$$

Figure 16: Math Facets in TeMa v2

5.3 Performance of the Schematizer

We designed the Schematizer to be a very lightweight daemon, both as memory requirements and as CPU usage. To test if we achieved this goal, we benchmarked it on a server running Linux 3.2.0, with 10 cores (Intel Xeon CPU E5-2650 2.00GHz) and 80 GB of RAM.

We obtained the 1123 expressions to be schematized by querying Elasticsearch with the keyword "Fermat". While the overall time taken by the faceted search engine was around 5 seconds, less than a second was spent in the Schematizer. Also, the CPU utilized by the Schematizer never rose higher than 15% (as indicated by the top utility). Asymptotically, the algorithm would run in $O(N)$ time, where N is the number of input formulae. We are able to reach linear time performance, because each formula is processed exactly once and the signature is stored in a hash table, as discussed in Section 4.2.

The space complexity is also linear in the number of formulae, because in the worst case scenario (large cut-off depth) we will have a schema for every formula. To analyze the memory footprint of the Schematizer we used Massif [11], a heap profiler from Valgrind's tool suite. Figure 17 shows the total heap memory consumption for a single query (containing 1123 expressions). Massif reports time using the number of executed instructions as unit of measurement. The heap memory size is given in bytes.

There is a short, steep, increase in heap usage after the program started. This can be attributed to the query set being received (since we are storing the formulae in memory first). Then, the consumption keeps increasing because we are generating the schemata signatures. The peak is reached at 20 MiB, which is relatively low. Afterwards, there is a sudden decrease in heap size when we drop the schemata which are not needed (because a max_size parameter was specified by the caller). For most of the next part, the heap size stays constant because we are only processing the signatures and finding the PMML substitution IDs. In the end, the memory footprint decreases to (almost) zero, because after answering the query, all formulae and associated objects are being discarded.

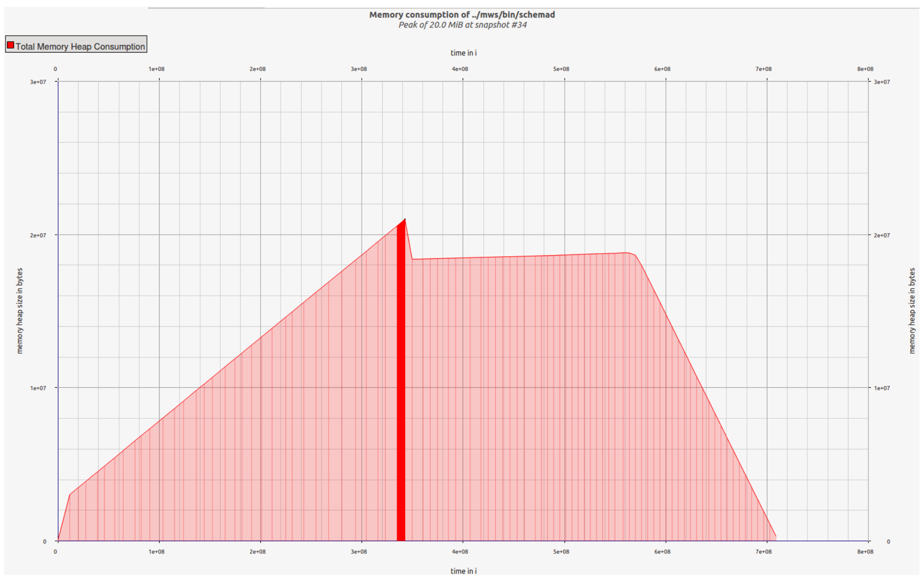


Figure 17: Schematizer Heap Usage for a Query

All things considered, it is somewhat remarkable that the peak memory usage is at only 20 MB and only 15% of one processor is in use for 1123 formulae. Considering the linear complexity, we can accommodate several millions of formulae on the mentioned server:

$$80GB \cdot \frac{1123}{20MB} \approx 4.5 \cdot 10^6 \text{formulae}$$

Due to its implementation, the Schematizer is indefinitely scalable, because it does not require shared state between formulae and can therefore be implemented as a MapReduce [2] job, where mappers compute the signature of assigned formulae and reducers assemble the signature hash table.

5.4 The API

Another relevant point of interest in the Schematizer is its simple, but sufficient API. By providing an HTTP endpoint, any application which wishes to use the schematization service can do so, simply by issuing a GET request. The content of the GET request must be an XML document, with `mws:query` as the root element. The children of the `mws:query` are the expressions to be schematized in Content MathML format. The attributes of the query are answsize (the maximum number of schemata), output (JSON or XML), depth (the depth for cutoff) and cutoff_mode (specifying whether the cutoff should be absolute or relative). If JSON is used for the output, the IDs of PMML substitutions will be returned. If XML is used, formula schemata in CMML format will be returned.

6 Applications and Future Work

One improvement angle that can be worked on is the ranking of the schemata. We have used a simple method, ranking them in decreasing order of coverage, thus having the schema with most formulae in its class on the first place. However, this is not always a good approach. When users look at the facets, it is usually because they were not able to find what they were looking for (because the result set is too large). The first schemata cover most of the formulae users have already looked at, so they are not of interest. However, the last schemata are not of interest either, because they typically only cover very rare formulae (1-2 occurrences). A good ranking approach would place the *medium-coverage* schemata first, then the top-coverage and then the low-coverage. In order to define precisely what is the range for medium-coverage, further research is required.

One other application of the faceted search engine can be providing mathematical definitions with the help of NNexus [5]. NNexus is an auto-linker for mathematical concepts from several encyclopedias, e.g. PlanetMath, Wikipedia. Assuming we are able to generate relevant schemata in response to keyword queries, we can target the faceted search engine with all the concepts stored by NNexus and store a schema for each such concept. Afterwards, for a given query, we can obtain the schema and check it against our stored set of schemata. If we find it, we can link the given expression to its mathematical definition. Given a large number of stored concepts and a high schemata relevance, the user should be able to see the definition of any encountered formulae on the Web. For example, hovering over $a^2 + b^2 = c^2$ will show the definition of the Pythagorean theorem.

Another, more direct, application of the Schematizer would be *Similarity Search*. One could create a MathWebSearch based search engine, which accepts an input formula and a similarity degree (between 0% and 100%). The engine would then create a formula schema at a relative depth corresponding to the similarity degree and use this schema to search the corpus. This approach defines the similarity between two formulae as the percentage of the CMML tree depth that they share.

7 Conclusion

We have presented the design and implementation of a system capable of mathematical faceted search. Moreover, we have described a general-purpose scalable Schematizer which can generate intuitive and recognizable formula schemata and divide expressions into formula classes according to said schemata. Consequently, we have successfully addressed all challenges outlined in Section 2.1.

Although the Schematizer provides recognizable formulae, some queries to SchemaSearch (e.g. using an author as keyword) provide hits with a very low relevance. This is because we cannot distinguish between the work of the author and work where the author is cited at the textual level. As a consequence, searching for "Fermat" would also show formulae from papers where Fermat was cited and if these papers are numerous, as it happens with known authors, would provide the user with misleading results. This suggests that a better source of mathematical expressions might be required for the SchemaSearch demo.

References

[1] *ArXiv Online*. Dec. 21, 2014. URL: http://arxiv.org/ (visited on 12/21/2014).

[2] Jeffrey Dean and Sanjay Ghemawat. *MapReduce: Simplified Data Processing on Large Clusters*. 2004.

[3] *DuckDuckGo Website*. May 8, 2015. URL: https://duckduckgo.com (visited on 05/08/2015).

[4] *Elastic Search*. Dec. 7, 2014. URL: http://www.elasticsearch.org/ (visited on 12/07/2014).

[5] Deyan Ginev and Joseph Corneli. "NNexus Reloaded". In: *Intelligent Computer Mathematics*. Conferences on Intelligent Computer Mathematics (Coimbra, Portugal, July 7–11, 2014). Ed. by Stephan Watt et al. Lecture Notes in Computer Science. Springer, 2014, pp. 423–426. URL: http://arxiv.org/abs/1404.6548.

[6] *Google Website*. Dec. 21, 2014. URL: http://google.com/ (visited on 12/21/2014).

[7] Peter Graf. *Substitution Tree Indexing*. 1994.

[8] Radu Hambasan, Michael Kohlhase, and Corneliu Prodescu. “MathWebSearch at NTCIR-11”. In: pp. 114–119. URL: http://research.nii.ac.jp/ntcir/workshop/OnlineProceedings11/pdf/NTCIR/Math-2/05-NTCIR11-MATH-HambasanR.pdf.

[9] Michael Kohlhase et al. “Zentralblatt Column: Mathematical Formula Search”. In: *EMS Newsletter* (Sept. 2013), pp. 56–57. URL: http://www.ems-ph.org/journals/newsletter/pdf/2013-09-89.pdf.

[10] *LevelDB*. Dec. 21, 2014. URL: http://leveldb.org/ (visited on 12/21/2014).

[11] *Massif: a heap profiler*. Apr. 6, 2015. URL: http://valgrind.org/docs/manual/ms-manual.html (visited on 04/06/2015).

[12] *Mathematical Markup Language*. URL: http://www.w3.org/TR/MathML3/.

[13] *Mathematics Subject Classification (MSC) SKOS*. 2012. URL: http://msc2010.org/resources/MSC/2010/info/ (visited on 08/31/2012).

[14] Bruce Miller. *LaTeXML: A LaTeX to XML Converter*. URL: http://dlmf.nist.gov/LaTeXML/ (visited on 03/12/2013).

[15] *MwsHarvest*. Dec. 21, 2014. URL: https://trac.mathweb.org/MWS/wiki/MwsHarvest (visited on 12/21/2014).

[16] *Node.js*. Dec. 21, 2014. URL: http://nodejs.org/ (visited on 12/21/2014).

[17] Corneliu C. Prodescu and Michael Kohlhase. “MathWebSearch 0.5 - Open Formula Search Engine”. In: *Wissens- und Erfahrungsmanagement LWA (Lernen, Wissensentdeckung und Adaptivität) Conference Proceedings*. Sept. 2011. URL: https://svn.mathweb.org/repos/mws/doc/2011/newmws/main.pdf.

[18] *XSLT for Presentation MathML in a Browser*. Dec. 20, 2000. URL: http://dpcarlisle.blogspot.de/2009/12/xslt-for-presentation-mathml-in-browser.html#uds-search-results (visited on 04/04/2015).

[19] *Zentralblatt Math Website*. Dec. 7, 2014. URL: http://zbmath.org/ (visited on 12/07/2014).