JACOBS
UNIVERSITY

Faceted Search for Mathematics

by

Radu Hambasan

Bachelor Thesis in Computer Science

Prof. Michael Kohlhase
Supervisor

Date of Submission: April 6, 2015

Jacobs University — School of Engineering and Science

With my signature, I certify that this thesis has been written by me using only the indicated resources and materials. Where I have presented data and results, the data and results are complete, genuine, and have been obtained by me unless otherwise acknowledged; where my results derive from computer programs, these computer programs have been written by me unless otherwise acknowledged. I further confirm that this thesis has not been submitted, either in part or as a whole, for any other academic degree at this or another institution.

Radu Hambasan Bremen, April 6, 2015

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 **1**

2 Preliminaries **3**
- 2.1 Project Goals . . . 3
- 2.2 MathWebSearch . . . 4
- 2.3 Elasticsearch . . . 5
- 2.4 arXiv . . . 7
- 2.5 LaTeXML . . . 7

3 Implementation **8**
- 3.1 Formalizing the problem . . . 8
- 3.2 An Algorithm for FSG . . . 8
- 3.3 Finding a cutoff heuristic . . . 9
- 3.4 Design Overview . . . 10
- 3.5 The Formula Schematizer . . . 10
- 3.6 The Elasticsearch proxy . . . 11
- 3.7 The Front-End . . . 12
- 3.8 ES aggregations . . . 12
- 3.9 Making the schemata recognizable . . . 13
- 3.10 Naming query variables . . . 14

4 Evaluation **17**
- 4.1 Front-End results displaying . . . 17
- 4.2 Performance of the Schematizer . . . 19
- 4.3 The API . . . 20

5 Conclusion **20**

6 Applications and future work **21**

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 [4] or Bing [2]) seem to be successful for the average user, they are limited when it comes to finding scientific content. This is because STEM[1] documents are mostly relevant for the mathematical formulae they contain and math cannot be properly indexed by a textual search engine, because the hierarchical structure of the content is also important.

A good math search engine is therefore needed in several applications. For example, a large airline 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. 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 need first 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]. Having almost a million documents, with possibly more than a billion 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 these is the Zentralblatt Math service [14].

Zentralblatt Math now employs formula search for access to mathematical reviews [6]. 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

[1] Science, Technology, Engineering and Mathematics

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, but not for formulae. In this way, the user is given the ability to further explore 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.

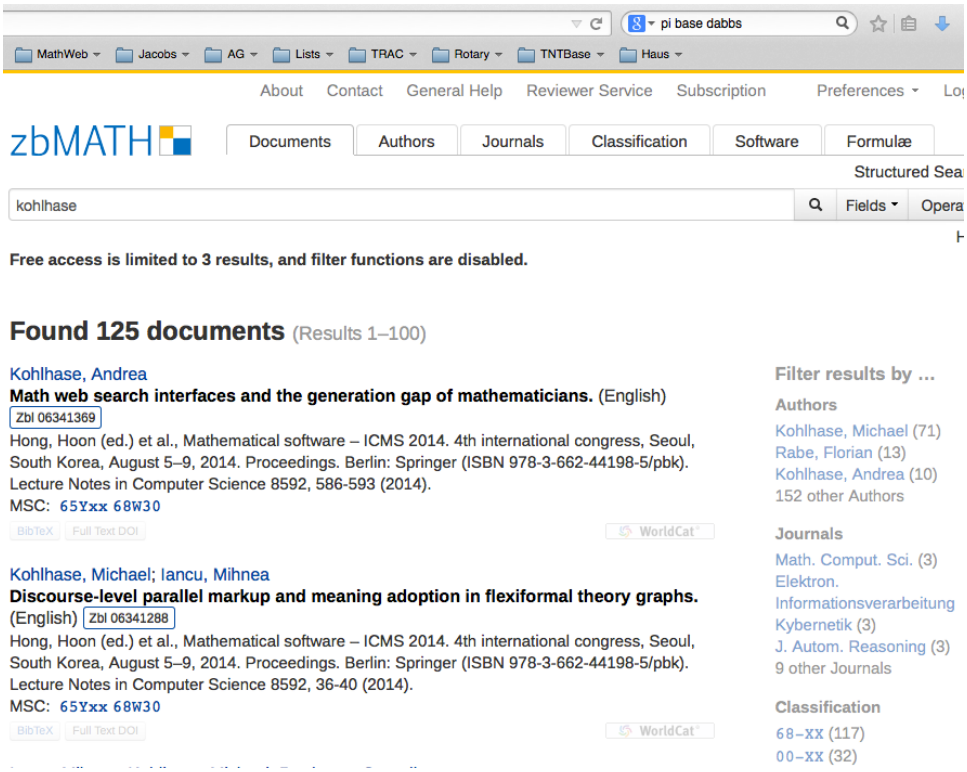

Figure 1: Faceted Search in ZBMath

There are multiple ways in which we could understand a “math facet”. One way would be through the MSC classification [9]. However, this would be rather vague because it will only provide info 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**.

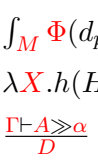

Figure 2: formula facets

These formula schemata were manually created, 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** which provides the necessary index structure for schema search.
- **Elasticsearch** which provides hits in response to text query, as well as run aggregations on the hits. These hits represent formulae to be schematized.
- **arXiv** which provides a large corpus of mathematical documents that we can index and run our system on.
- **LaTeXML** which converts LaTeXexpressions 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 requirement for our system:

1. 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.
2. it should be able to classify the given set of formulae according to the generated schemata.
3. 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 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. Recently, it was augmented with full-text search capabilities, combining keywords query 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.

The overall workflow of TeMaSearch is the following:

1. HTML5 documents representing mathematical articles are crawled to generate MWS harvests [11]. .harvest 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 [12] 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.

For fast retrieval, in the leaves of the substitution tree, MWS stores FormulaIDs. These are numbers uniquely associated with formulae, and they are also used to store context and occurrences about the respective formula. They are stored in a separate LevelDB [7] database.

A simplified sketch of the index is shown in Figure 3.

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 MathWebSearch and Elasticsearch to answer queries. In order to achieve cooperation between the two systems, 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 [3] 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[2] 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

[2] Faceted search as such is now deprecated in ES and was replaced by the more powerful “aggregations”.

on the number of the buckets (items) for which we want to receive the count.

An ES query which would return the most frequently used formulae (and subformulae) for "Pierre Fermat", is presented in Listing 1. 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*.

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

Listing 1: Elastic Search Term Aggregation Query

A possible response to the above query can be found in Listing 2. 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 2: 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 [5], 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 LaTeXor 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 LaTeXexpressions to more organized formats such as MathML.

An open source example of such a conversion engine is LaTeXML [10]. The MathWebSearch project relies heavily on it, to convert ArXiv documents from LaTeXto XHTML which is later indexed by MWS. It exposes a powerful API, allowing definition files to relate TeXelements to corresponding XML fragments that should be generated. For the scope of this project, we will be more interested in another feature of LaTeXML: cross-referencing between Presentation MathML and Content MathML. While converting TeXentities to a Presentation MathML tree, each element receives 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.

3 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.

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 $\mathcal{D}$.*

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 FSG algorithm we implemented requires a MWS index of the 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:

- Parse the given CMML expressions similarly to MWS queries, to obtain their encoded DFS representations.
- Choose a reasonable cutoff heuristic, see 3.3.
- Unify each expression with the index, up to a given threshold (given by the above heuristic).
- 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.

- We sort these path-counter pairs by counter in descending order and take the first n (n being the width required by the FSG).
- 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 3 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.

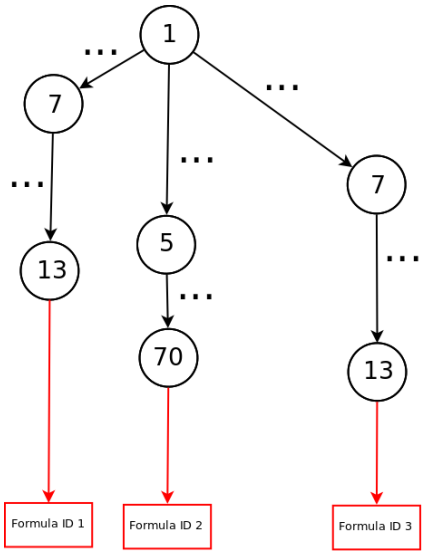

Figure 3: Simplified index at depth 1

3.3 Finding a cutoff heuristic

In order to generate formula schemata, we must define a "cutoff heuristic", which tells the program when two formulae belong to the same schemata class. If there was 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.

One reasonable cutoff heuristic would be a certain expression depth, given as a parameter to the schema-engine. In this way, a depth of 0 would always return $?x$ which corresponds to the 0-unification. The algorithm above is based on this heuristic. Another possible choice for the cutoff would be expression length. This would output formulae which

begin the same way.

3.4 Design Overview

The full faceted search system comprises of the following components: the Formula Schematizer 3.5, 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 4.

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 (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.

3.5 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.

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. [1] [2] EdN:1 EdN:2

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 a 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, in order 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.

3.6 The Elasticsearch proxy

In order to obtain formulae to feed as input to the Schematizer, we will 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), a NodeJS proxy was developed. 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 Elas-

[1] EdNote: Explain encoding of tokens
[2] EdNote: Add signature examples

ticsearch results to the Schematizer. To ensure a fast response, only a minimum of 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.

3.7 The Front-End

To demonstrate the capabilities of the Schematizer, as well as provide an access point for the user, a simple front-end was developed. As shown in figure 5, 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.

3.8 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 $\tilde{1}0$s, 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 to the approach using aggregations.

3.9 Making the schemata recognizable

As explained in Section 3.5, the initial Schematizer returned formula schemata as Content MathML. Naturally, we would need to transform it to Presentation MathML in order to be able to display it to the user. For this purpose we used and XSL stylesheet written by David Carlisle [13]. Figure 6 shows the initial results of the conversion.

It is clear that the conversion leads to somewhat disappointing results. Firstly, they are not recognizable and secondly, they contain left-over text such as “subscript”, “superscript”, “normal”, etc. On a deeper analysis, we can point out two causes for this strange behaviour.

The reason why schemata are hard to recognize is the inherent ambiguity of CMML. For instance, a csymbol element can be represented in at least three different ways depending on the notation being used. [3] EdN:3
Also, the two applications `apply(f, a, b)` and `apply(+, a, b)` are described similarly by CMML, but represented in a different manner by PMML. If the Schematizer replaces the tokens `f`, `a` and `b` with query variables, both the of applications will be displayed as `?x1(?x2, ?x3)`, even though we would expect `?x1` to be infix when it represents the plus operator.

The reason for the left-over text is even simpler: the stylesheet does not implement enough conversion rules, so it is not able to convert `subscript`, `superscript` and other elements to PMML.

Although this was a significant hindrance for the faceted search engine, we managed to overcome it by making 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”[4] effectively return- EdN:4
ing 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.

[3] EdNote: @Michael: Deyan told me that there are >3 ways of representing a csymbol. Can you give me some real examples?

[4] EdNote: @Michael: too informal?

The procedure above will solve the left-over text issues, using the template's initial layout which is valid Presentation MathML. However, we still need to display the schemata in a way which provides intuition about its class. Heuristically, we can just keep the first child of apply (the operator) unsubstituted and replace only the other children (the operands) with query variables. The results obtained using this heuristic and the cross-reference system are further discussed in Section 4.

3.10 Naming query variables

As discussed in the section before, 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.

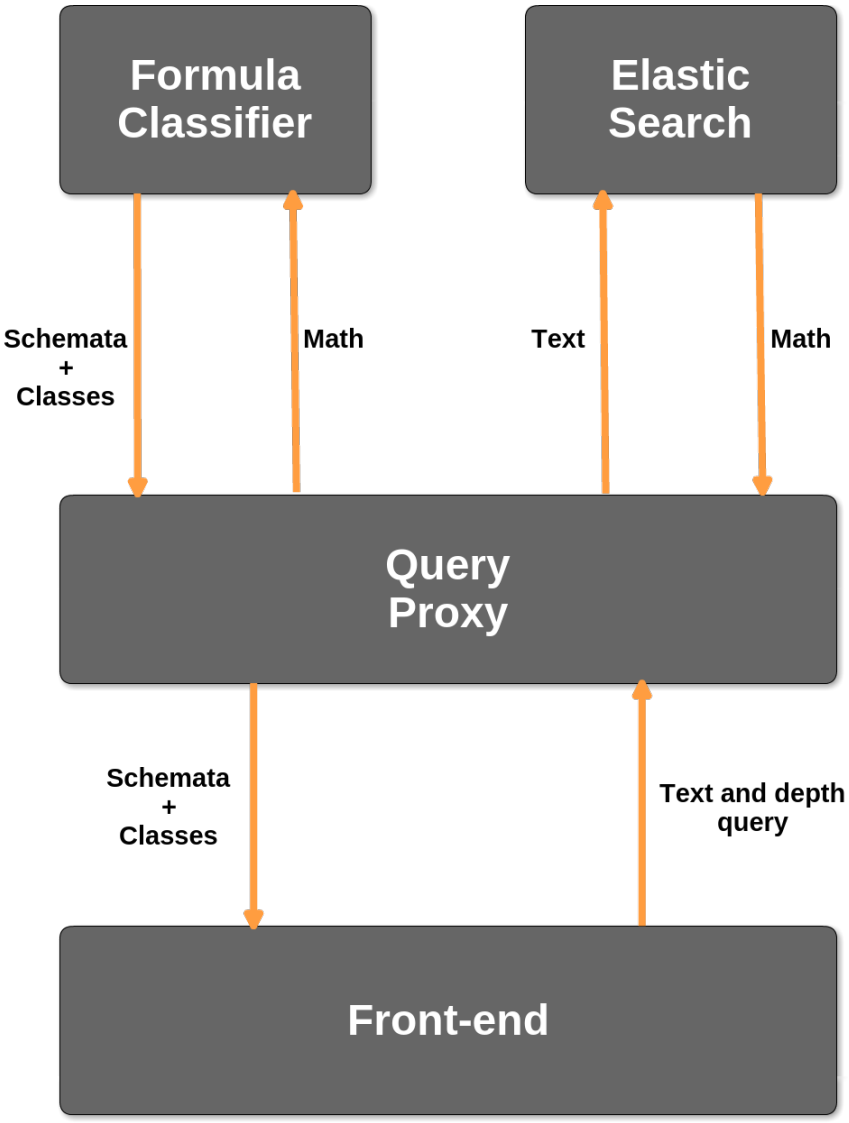

Figure 4: Faceted search engine architecture

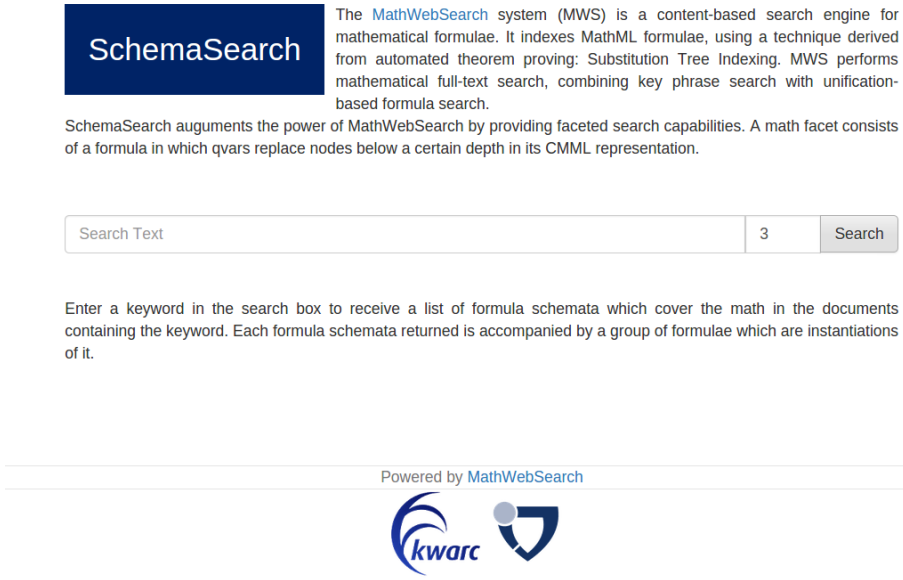

Figure 5: Faceted search front-end

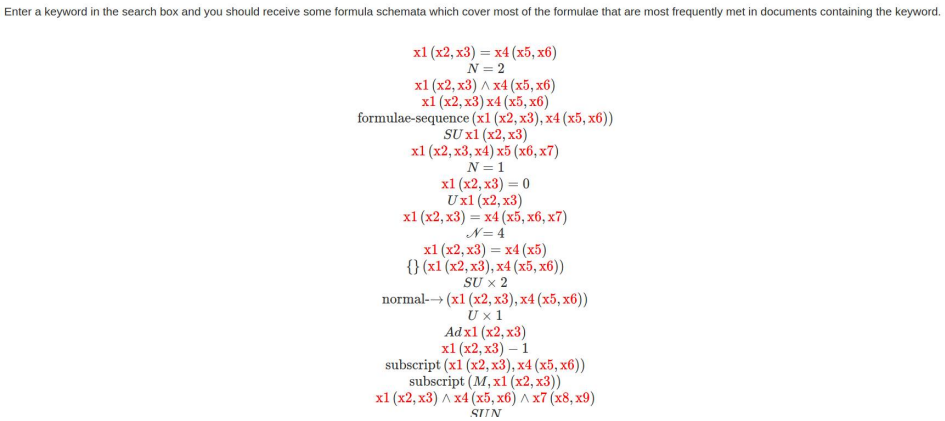

Figure 6: Displaying Content MathML

4 Evaluation

In this section, we will present an evaluation of our faceted search engine. Firstly, we will examine the front-end (4.1), 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 (4.2). In the end, we will look at the API that the Schematizer exposes and elaborate on its potential (4.3).

4.1 Front-End results displaying

Figure 7 shows the formula schemata at depth 3 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 red are query variables (qvars).

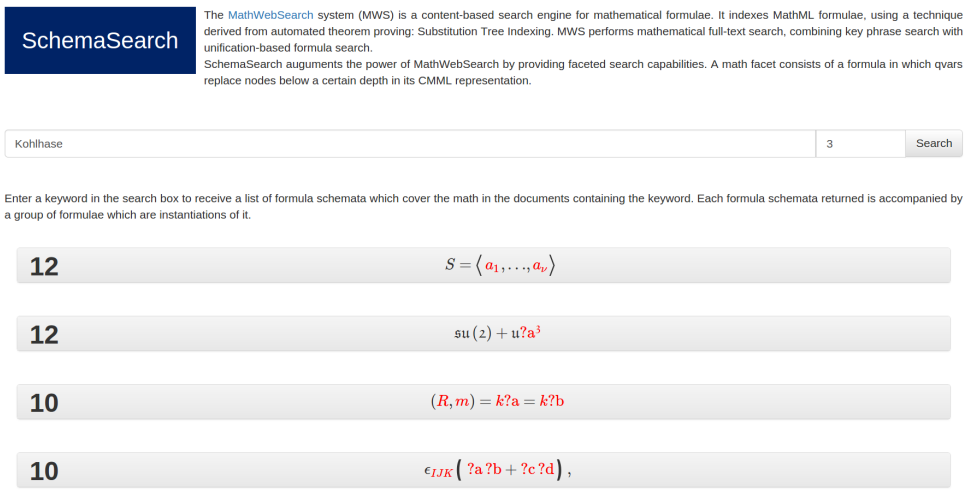

Figure 7: Faceted results at depth 3

Figure 8 shows the expansion of a formula class. There are three formulae in the class given by this particular math schema, as indicated by the count on the left upper side. We have one named query variable, corresponding to a Content MathML leaf and 7 anonymous ones marked with letters from a to g.

As explained in Section 3.9, the first formula from the class was also used as a template to solve CMML ambiguities. This fact is easily noticeable by comparing the first formula with the math schema. It is very intuitive that the former is an instantiation of the latter. We can see that the named qvar Z instantiates to Z, as expected since it is a leaf. Other elements from the formula are at a higher depth, so entire nodes are replaced by the qvars, for example $?b$ replaces m_R^2.

Despite the close relation between the first member of the class and the schema, it is slightly less intuitive to see why the other formulae from the class are instantiations. Between Z and $?a$ there is an invisible times operator. Given the presence of this operator, the left hand-side of the equality can be expressed in CMML as `apply(times, Z, ?a)`, which is equivalent to `apply(f, k, 0)` because all arguments of apply are below the cut-off depth. Since we decided to keep the first child of apply (the operator) unsubstituted, the left hand-side might not initially appear as a function application.

A similar issue arises with $?b + ?c$ and $\frac{1}{\sqrt{W_k}}$. The user could be under the impression that the former is an addition between two qvars, when it is actually a function application to two qvars. The operator of this function application is division for the second formula. Since both the schema operator (addition) and the formula's operator (division) are below the cut-off depth, the CMML structure is essentially the same (up to the cut-off depth).

As we have seen, there is an important trade-off to be made. We can either show a very cryptic schema which instantiates somewhat more intuitively, or show an easily recognizable schema with less intuitive instantiations. We have opted for the second choice, because it seems that the user would be more interested in receiving relevant schemata as titles for the classes.

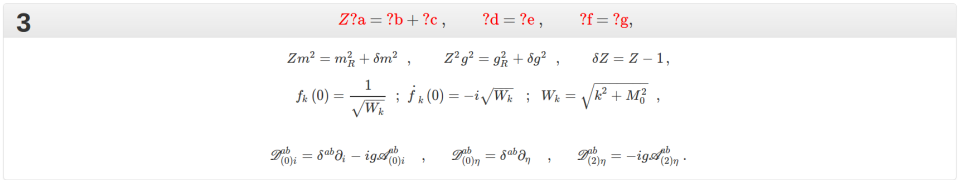

Figure 8: Expansion of a formula class

4.2 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 by 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 3.5.

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 [8], a heap profiler from Valgrind's tool suite. Figure 9 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 initial steep increase moments after the program started. This can be attributed to the dynamic linker. Then, the consumption keeps increasing because we are receiving the formulae (storing them in memory) and generating the schemata. The peak is reached at 20 MiB, which is impressively 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.

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:

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

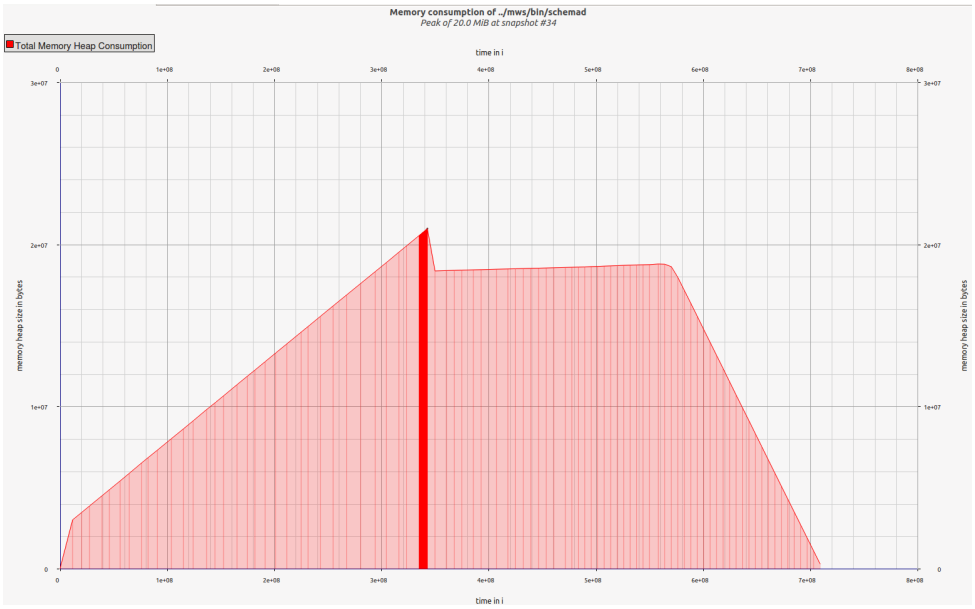

Figure 9: Schematizer heap usage for a query

4.3 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, which `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) and depth (the depth for cut-off). 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.

5 Conclusion

We have presented the design and implementation of a system capable of mathematical faceted search. Moreover, we have described a general-purpose Schematizer which can generate 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 easily recognizable formulae, some instantiations might be counterintuitive, which suggests a better cut-off heuristic is needed. Also, some queries (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 Schematizer.

6 Applications and future work

5 6

EdN:5
EdN:6

References

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

[2] *Bing Website*. Dec. 21, 2014. url: http://bing.com/ (visited on 12/21/2014).

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

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

[5] 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.

[6] 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.

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

[5] EdNote: mention NNexus

[6] EdNote: next steps? improve style sheet?

[8] *Massif: a heap profiler*. Apr. 6, 2015. url: http://valgrind.org/docs/manual/ms-manual.html (visited on 04/06/2015).
[9] *Mathematics Subject Classification (MSC) SKOS*. 2012. url: http://msc2010.org/resources/MSC/2010/info/ (visited on 08/31/2012).
[10] Bruce Miller. *LaTeXML: A LaTeX to XML Converter*. url: http://dlmf.nist.gov/LaTeXML/ (visited on 03/12/2013).
[11] *MwsHarvest*. Dec. 21, 2014. url: https://trac.mathweb.org/MWS/wiki/MwsHarvest (visited on 12/21/2014).
[12] *Node.js*. Dec. 21, 2014. url: http://nodejs.org/ (visited on 12/21/2014).
[13] *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).
[14] *Zentralblatt Math Website*. Dec. 7, 2014. url: http://zbmath.org/ (visited on 12/07/2014).