Friedrich-Alexander-Universität
Erlangen-Nürnberg

Professur für Wissensrepräsentation und -verarbeitung

Diagram Chasing Done Meta

Formula Search for nLab

Bachelor Thesis in Computer Science

Author: Christoph Alt

Advisors: Prof. Dr. Michael Kohlhase, Tom Wiesing

Erlangen, December 1, 2019

Abstract

When searching for mathematical formulas we encounter the issue that classical text retrieval approaches perform poorly, since much of the meaning of such an expression lies in the structure and the context, while aspects like the variable names are mostly irrelevant. On the field of Category Theory, it becomes even worse, since a lot of contents are presented as diagrams and a textual description becomes really long. The nLab is a big resource for mathematics, especially for Category Theory and related topics, on the internet.

To provide a possibility to search through its contents, so to say to chase for diagrams, this thesis presents a formula search application for nLab, based on the MathWebSearch engine, along with a new user interface, which is based on React. To convert the formulas in the nLab, so that they can be indexed, LaTeXML with a custom plugin, is used.

Ich versichere, dass ich die Arbeit ohne fremde Hilfe und ohne Benutzung anderer als der angegebenen Quellen angefertigt habe und dass die Arbeit in gleicher oder ähnlicher Form noch keiner anderen Prüfungsbehörde vorgelegen hat und von dieser als Teil einer Prüfungsleistung angenommen wurde. Alle Ausführungen, die wörtlich oder sinngemäß übernommen wurden, sind als solche gekennzeichnet.

Erlangen, December 1, 2019 _____

Contents

1 Introduction

For common text based Information Retrieval there are quite good and successful approaches available. For instance, there are Google, DuckDuckGo or open source engines like Elasticsearch [Es] or Apache Lucene. However, they perform not that well on mathematics. This is due to the fact that mathematical contents not only consists of plain text. Mostly mathematical documents contain formulas annotated with some explaining text. Formulas are not like plain text, since they encode some specific semantic meaning, that depends on the context, the domain and a lot of other factors. A simple example for that is the name of variables. They are commonly replaceable, since $a^2 + b^2 = c^2$ encode the same as $x^2 + y^2 = z^2$.

When coming to the field of abstract algebra or category theory a lot of content is presented as diagrams to visualize the connections between different structures. For example we might come over something like this

$$\begin{array}{ccc} A & \xrightarrow{f} & B \\ \downarrow{\scriptstyle f'} & & \downarrow{\scriptstyle g} \\ C & \xrightarrow{g'} & D \end{array}$$

while studying on some topics of this field. Such a diagram visualizes that in a Category $\mathcal{C}$ some of its objects are connected by some morphisms. We say that this diagram commutes, if it is proved that $f \circ g = f' \circ g'$. Diagram chasing is a proof technique in this part of mathematics, which works by tracing various elements through a commutative diagram [Nlab].

In case we discover such a diagram in a document, the authors might have shown some property of the objects in a Category. If we want to find out more about such a diagram, because the surrounding text is not informative enough for us and we do not know the right keywords to search for, then we encounter some problems when using a normal text search application.

Since formulas are represented in the most documents as some LaTeX expressions, we note that we can hardly use the LaTeX representation as an input for the text search application. For example, if we use Google to search for the LaTeX representation of the example above, we get more results that refer to the LaTeX documentation about the used LaTeX commands than about actual diagrams.

Another problem comes with the already mentioned variable names. What if the example above is some general diagram and we are interested in some special cases of that diagram, where we have not an arbitrary object A, but some fixed object. Or if other authors wrote about the same topic but just used different variable names (for instance X instead of A), then we also want to find this. So we want to have a possibility to search for generalizations or specializations of a concept, if we have a representation of it.

Research Question In this thesis, we will discuss the development of a formula search application for the field of Category Theory. That allows us the search for (LaTeX) encoded formulas with some kind of wildcards, that are replace by arbitrary terms in the search process. We want to answer the question if we can chase diagrams instead of chasing in them.

One of the biggest resources for Category Theory and related fields of mathematics on the internet is the nLab [Nlad]. It is a community driven wiki and contains 14K articles. These articles contain a huge amount of diagrams and other formulas and it might also hold a page that may answer our question. All we need to do is to search and find these articles, but searching through them manually seems to be a never ending task.

Use Cases There are multiple use cases for such a formula search application. For example, for students that are studying on some topic by working through a text book or lecture notes and then come over some formula that is denoted with a phrase like, thats some how trivial or the proof is left to the reader as an exercise, it might be helpful to have a possibility to gather some more information about that formula and its meaning. Maybe that also happens to other consumers of mathematical contents, so not only students may use that. A further use case might be if we get over some concept that is expressed in a formula and we want to look if there are similar concepts that might be represented in a similar shape.

Running Example In the following, we use the Natural Numbers Object (NNO) [Nlae] as a running example. The NNO is a category theoretical generalization of the natural numbers. It exists in categories with a Terminal Object and has a morphism $z : 1 \to \mathbb{N}$ and a morphism $s : \mathbb{N} \to \mathbb{N}$. Especially, we will use the morphism z, to figure out what information we can gather with the help of our formula search application.

Contribution The contribution of this thesis is the deployment of a math-aware search application with the contents of the nLab, that uses MWS as search engine and LaTeXML with a custom plugin for the conversion of its contents. This thesis also presents a newly developed user interface for MWS to replace the old one. The new one is based on a modern framework, has extended functionality and is easy to deploy.

Overview This thesis is structured as follows: Section 2 gives an overview about related work and the current state-of-art in math-aware searching. In Section 3 the preliminaries, including MWS, LaTeXML and React are introduced. Section 4 is about the process of extracting the formulas from the nLab. Section 5 is about the newly developed search interface and in Section 6 follows a conclusion.

2 Related Work

Guidi and Sacerdoti Coen present a survey about the topic of Mathematical Information Retrieval (MIR) in [GSC15]. There the authors give an overview about the topic and sum up the different techniques and their implementations.

In MIR, the problem of having two different types of content that should be searched (and found) needs to be tackled. On the one hand, there is the mathematical content, that is some structured content that encodes some special semantic meaning. On the other hand, the formulas are also surrounded by some text, that is somehow unstructured and helps the reader of the documents to understand them. The typical query may contain formulas as well as some keywords. To combine these in a reasonable way is the big issue in MIR.

While the information retrieval on full text content is far advanced, it is not as far yet on mathematical content. In [GSC15] there are two main approaches to tackle this problem elucidated, the text-based and the tree-based. The text-based approach tries to reduce the problem to a full text search problem by converting the formulas to textualized items. This approach has been implemented several times, e.g. in [MY03; SL11; YM19] (for a more complete list consult [GSC15]). This approach makes it possible to use some well-developed full text search engine like Elasticsearch [Es] and the combination of the results of the text and the formulas comes there for free. But this method heavily relies on the quality of the conversion of the formulas to text and always comes with some (partial) loss of the structure of the mathematical content. This text-based approach has performed less successful in the past during the NTCIR contest[1] [Aiz+14; Aiz+16].

The tree-based approach tries to build some kind of hierarchical structure, typically a tree, out of the mathematical content. The tree-based methods separate also in two different practices. The first are the operator trees and the second the symbol layout trees. Operator trees are meant to build on the actual meaning of the formula and are typically generated out of some Content MathML encoded formulas. An example for such an engine is the MWS system that is used in this thesis. Symbol layout trees are based on the appearance of the formula and are typically based on Presentation MathML. In [Man+19] and [Dav+19] are different versions of the Tangent search engine presented, which uses a combination of operator tree and a symbol layout tree.

An example for the operator tree is the Approach0 system [ZZ19]. There the authors not only create an operator tree as index, they also use a method to measure the similarity between formulas. Currently, there is an instance of it running[2]. It allows searches on an index that is based on Mathematics StackExchange. Their user interface allows to query formulas combined with keywords. Another example that can be used and tried out is Search on Math [Oli+17][3], which has an index with about 11 Million formulas from

[1] NII Testbed and Community for Information Access Research

[2] https://approach0.xyz/search/

[3] https://www.searchonmath.com/

Mathematics StackExchange, Wikipedia, NIST DLMF[4] and some more. This user interface also allows to search for combinations of formulas and keywords. A feature that also both of these interfaces provide, is some menu to insert symbols and operators in the search query. Thus, it is possible to insert formulas without the use of the respective TEX/LATEX commands.

Some studies about search behaviour of mathematicians can be found in [Koh16] and [Koh14]. There the authors used interviews with mathematicians to conduct different design implications for mathematical search interfaces. They identified fact-finding as the primary mathematical search task. Another interesting study about user behaviour while searching mathematical content can be found in [MZO19]. The authors came to the conclusion that searching for math related information is often unsatisfying and that the users needs need to be considered already during designing the digital library that will be browsed later.

Most of the digital mathematical contents is encoded as LATEX expressions. Thus, often comes the task of converting these formulas to another format, commonly to MathML. In [Sch+18] is benchmark framework for the mathematical format conversion presented, along with an evaluation of the state-of-the-art methods for formula conversion. One of thier results was that LATEXML performed best on the LATEX-to-MathML conversion, even though it comes with a slow conversion speed.

[4]National Institute of Standards and Technology Digital Library of Mathematical Functions

3 Preliminaries

The following section introduces the diffrent systems our search application uses. First, we will introduce the contents, so the nLab, we want to search through. Second, we we will discuss the engine that converts the mathematical snippets of our corpus in the right format, so that our search engine can handle it. Third, it follows a subsection about the React framework that is used to build the user interface and then comes an overview about MWS engine and its ecosystem.

3.1 nLab — A Wiki for Category Theory

The nLab [Nlad] is a community driven Wiki on Mathematics, Physics and Philosophy. It originates from the blog n-Category Cafè[5] and was initiated to have an archive for the ideas and concepts that were developed in that blog and its comments. The nLab could be called the Wikipedia for category theory and claims to have a n-categorical point of view, instead of having a neutral point of view [Nlaf]. So all the contents of nLab are written from the perspective of category theory or higher category theory. It is meant to be a public group lab book and contains research-level notes, collaborative work, summaries of known work and complete lecture notes, to have a place to collect and connect mathematical information.

Furthermore, similar to Wikipedia, everybody is welcome to contribute to the nLab by editing, replenishing or creating articles about related topics. For every page there exists an overview about the of its history, so every change can be inspected and reviewed by the community. For discussions about contributions and the related topics, the community uses the nForum[6].

Currently it contains roughly 14K pages and articles. Multiple times a day these articles are changed or updated or new articles are added. These 14K pages contain at the moment 2.8 million (sub-) formulas that can be indexed by the MWS search engine.

3.2 LaTeXML — A LaTeX to MathML Translator

TeX/LaTeX is the common way to write down formulas and mathematical content because it is usable, portable and popular among mathematicians and related research communities. But MWS uses Content MathML snippets to build its search index, since this is a more structured format for mathematical fragments. That means we need a way to convert LaTeX to MathML before we can use it with our search engine. LaTeXML [LTX] is an open source tool to fulfil such a task and is used here to convert LaTeX fragments in MathML fragments. It provides a feature on which MWS relies on, the so-called Parallel Markup. LaTeXML

[5] https://golem.ph.utexas.edu/category/

[6] https://nforum.ncatlab.org/

converts a LaTeX formula to a Presentation MathML and a Content MathML representation and then adds references from a Presentation MathML element to the corresponding Content MathML node and vice versa.

Even though LaTeXML provides already a lot of functionality and supports a huge amount of LaTeX/TeX features, it can be easily extended. This is done with plugins, some community extensions to LaTeXML, that may provide new features, like support for otherwise unsupported LaTeX packages or custom setting profiles. Examples for this are the plugin for MWS [Mwsc] or the iTex plugin, both of them are mentioned later in more detail.

3.3 React — A JavaScript Library for Building User Interfaces

React [Reaa] is an open-source framework to create user interfaces. It is developed and maintained by Facebook and used for the user interfaces of Facebook and Instagram. Beyond that, it is used by other big companies, like Netflix or AirBnB, for their web applications.

There are several other popular frameworks and libraries for the development of user interfaces for web applications, like Vue.js, Angular or Polymer. But for the user interface of MWS, however, React was chosen. The reason is, that the user interface for MathHub.info is already based on React and should be kept easy to integrate into that [See18].

One of the main concepts of React is the so-called Virtual Document Object Model (DOM). This is an internal representation of the browser's DOM, that is kept in memory. Changes in components of the interface are first applied to that Virtual DOM. The difference to the actual DOM is calculated and only the components that are affected by the change are updated in the browser's DOM. With that strategy React aims to reduce the changes of the actual browser DOM, since they are resource intensive. So, the goal is to let the browser render only the updated parts of the DOM new instead of the complete DOM.

React provides a component interface, that allows to split the user interface into different parts. So we can encapsulate the different functionalities in different components in order to make them easily reusable and replaceable. To do so, React defers between class components and function components. Class components are special JavaScript classes that provide at least a render method, that essentially returns a description of how the component should look and behave. The most convenient way to do that is to use JSX [Jsx], which looks almost like normal HTML and can be embedded in the JavaScript Code. Since such a component is a class, it can encapsulate some state full logic. A function component is, as the name says, just a normal JavaScript function that returns such a JSX snippet, so in the past there were no possibilities to use some state in a function component.

React Hooks A new feature that was introduced with the React version 16.8.0 is the Hooks API. With this the developers of React wanted to establish the possibility to create state full logic that is reusable in different components in the application. Before that,

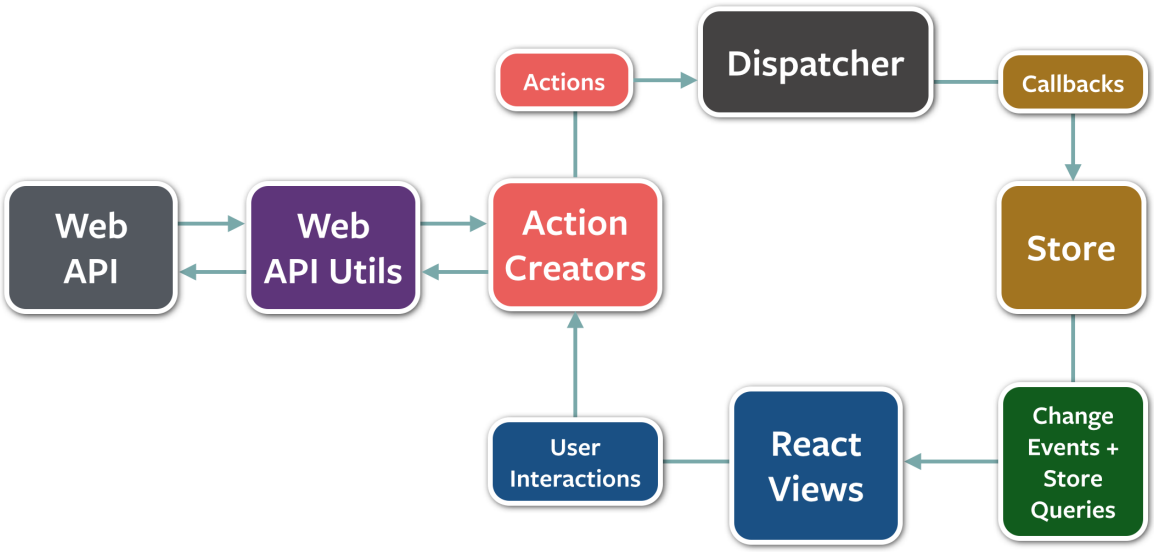

Figure 1: Diagram that visualizes the Flux architecture [Fbfa]

only React class components could use states and function components were stateless. So all state full logic was encapsulated in classes and it was barely possible to reuse this in other class components. With the use of these hooks it is possible to access the state of React outside of React class components. The most prominent Hook is the useState Hook which allows the use of a state and its modification. With the useEffect Hook it is possible to create side effects. With the creation of a React context it is possible to pass through the application, without the need to pass it through every single layer as an argument. So it is possible to model a global state conveniently. With the call of useContext this context can be accessed in a function component. [Reab]

Flux Since a React component combines logic for presentation and interaction in one object, Facebook also introduced a design pattern, called Flux, for such web applications [Fbfb]. It deviates from the Model View Controller architectures that are widely used for user interfaces. Figure 1 is a diagrammanticall visualization of the Flux architecture. The main idea of Flux is to have an unidirectional data flow in the application. This is achieved by having a single spot where the global data is stored. In most cases this is called the store. It keeps track of the state that is used in and by several different components. To influence the global state there are some objects, mostly called actions, that can be used by a component to sent new data, e.g. some user input, to the store. The actions are created with the help of a set of functions that are called action creators. Typically for every action there is at least one dedicated action creator. During the call of such action creator, it is possible that there are some asynchronous calls to the backend. The actions are sent to the store via a callback function that is called the dispatcher. This

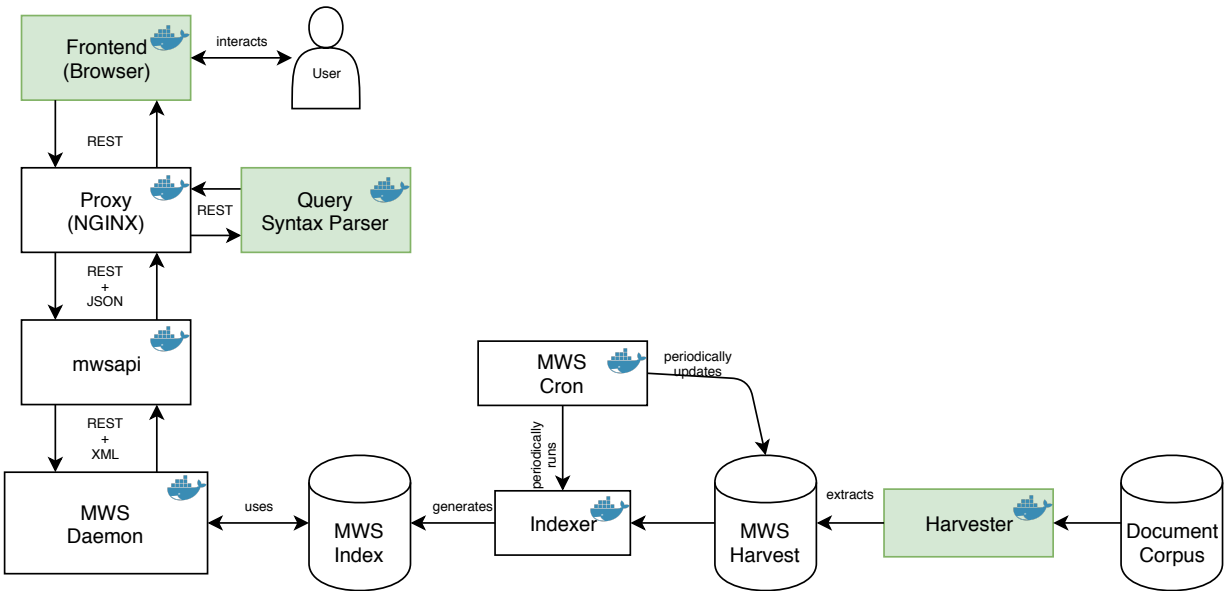

Figure 2: Layout of MWS deployment, components that are available as docker container are marked with a docker symbol. The colored components are described in more detail in Section 4 and Section 5. [Ber+19]

is essentially the entry point for the data to the store and the place where the actions are unpacked and the new data is added to the global state. Basically the dispatcher is a function that takes the current state and an action as arguments and returns a new state. This new state is then provided to all the other view components by the store. These view components are then showing the data to the user and provide some possibilities for interaction [Fbfb; Fbfa].

This architecture provides the advantage that it makes the application easily extensible, since adding new items to the data model just means to add new items to the store and maybe add some new actions. Furthermore, it is easy to add new view components, since they just have to include the store and therefore have access to the global state.

3.4 Overview of MathWebSearch Ecosystem

This section serves as a brief overview of the MWS infrastructure and its deployment, of which a visualization can be seen in Figure 2. Some of the components were newly introduced in [Ber+19]. For convenience, all the different components are deployed in docker containers [7]. This makes the deployment really comfortable, since every software part is running in its own environment and so unpleasant interferences, like confliction dependencies, are reduced.

[7] https://docs.docker.com/

The central part of the system is an instance of the MWS daemon. It does the main part of searching and provides an API for these queries. It can be found in the bottom left corner of Figure 2.

MathWebSearch [KŞ06; KMP12] is a search engine for mathematical content and is an example for the Operator Tree approach, which was mentioned in Section 2. It uses substitution tree indexing [Gra95] to create an index from Content MathML formulas. In such a substitution tree the expressions are stored as substitutions in the nodes and can be constructed by applying gradually the substitutions while the depth-first tree-traversal. The inner nodes of the index tree store different substitutions and only the leaves contain the actual formulas. To lower the memory usage, the formulas are not directly stored there, but some pointers to them. To allow the hits also on subterms, MWS adds every subterm of a formula to the index. This does not affect the memory footprint too much, because duplicate formulas or subterms are not stored twice, but the pointer in the leaves refer to the same formula.

MWS provides a REST API to process the search queries. Such a query is essentially a Content MathML expression that may be annotated with query variables (qvars). These query variables are wildcards in queries and can be substituted with every other possible subterm during the search process.

mwsapi As seen in Figure 2, the MWS instance is not directly queried, but there is some additional API layer in between. That is called mwsapi, [Mwsb] and was newly introduced in [Ber+19]. This daemon passes the queries to the MWS daemon and modifies the response before it is sent on. These modifications include the extraction of the hit subterm and the substitutions, so it can be presented in the frontend directly, without additional XPath querying.

Frontend The interface for the user is provided by the frontend, it runs in the browser and essentially gives the user the possibility to enter some search queries and displays the list of search results. The old interface had several issues owing to the fact that it was developed before the mwsapi layer was introduced. Thus, it could not communicate with it. Another one was that it was not flexible enough to present the contents of a new corpus properly. To fix that, it was necessary to modify the source code. So for every new deployment of a MWS instance the frontend needed to be adjusted and since it was not based on the state-of-the-art of the current web development technologies, we decided to develop a new one, based on modern framework for building user interfaces. How this is done, is elucidated in Section 5.

Query Syntax Parser A LaTeXML daemon parses the user's queries, by converting the LaTeX based queries with some query variables to MathML snippets. So it can be used to query the MWS daemon. This MWS specific extension to LaTeXML is brought by the

```
<?xml version="1.0"?>
<mws:harvest xmlns:mws="http://search.mathweb.org/ns"
    xmlns:m="http://www.w3.org/1998/Math/MathML">
  <mws:data data_id="{uuid}">
    <id>{id}</id>
    <text>{text}</text>
    <metadata></metadata>
    {% for f in formulae %}
    <math local_id="{f.id}">{f.math}</math>
    {% endfor %}
  </mws:data>
  {% for f in formulae %}
  <mws:expr url="{f.id}" mws:data_id="{uuid}">{f.cmml}</mws:expr>
  {% endfor %}
</mws:harvest>
```

Listing 1: pseudocode of harvest [Mwse]

MWS plugin for LaTeXML [Mwsc]. This daemon can also be extended by other plugins to provide some corpus specific bindings, for example. This was also done during this project and this plugin will be explained in Section 4.

Since MWS daemon provides the possibility to search for numeric ranges, we have added a binding to the LaTeXML plugin for MWS to enter such ranges so the user of the frontend has the possibility to use this feature.

Index The index that MWS uses to search through is created by the mws-indexer [Mwsf]. This takes a set of harvest files to create the index. A rebuild of the index is periodically triggered by mws-cron [Mwsa], which is for example relevant for the nLab, since it is community driven and regularly updated or extended by its users.

Harvest The MWS harvests (bottom right of Figure 2) are specific to the used corpus and are created by a specific Harvester. The implementation of the Harvester for the nLab is described in Section 4.

Harvest files are basically XML files [Bra+08] with some extension to MathML [ISO]. A pseudo code description of a Harvest file can be seen in Listing 1. Such a file contains a mws:harvest element as root element. The children of that are mws:data and mws:expr elements. The mws:expr element contains the actual Content MathML, that was extracted from a single formula which was in the source [KP]. The mws:data is there to store some metadata, represented as arbitrary XML, about the source files. Each mws:data node is identified by the `data_id` attribute, which should be unique, and corresponds with a single file in the source-corpus. Each mws:expr node refers with that `data_id` attribute to the mws:data element it belongs to.

To fit the demands of mwsapi [Mwsb], the mws:data should be structured like the following scheme, which can be seen also in Listing 1. The id element provides an identifier of the source file. The text element contains the actual text of the source file, where all extracted formulas are replaced with the string **"math+id"**, where the id is an identifier for the formula. This id is also used as the local_id attribute. As Listing 1 shows, the mws:data element also contains all formulas of the source file with Presentation MathML and Content MathML annotations as parallel markup. The mws:expr element that corresponds to a formula only contains the Content MathML part and the url attribute stores the value of the local_id. In the metadata there is some space for arbitrary information of the document. Furthermore, there are some math elements to store the Parallel Markup, which could be the output of the LaTeXML conversion for every single formula [Mwse]. If the MWS daemon finds a formula that matches the query it returns the hole mws:data, the formula belongs to. So the response of the MWS daemon contains the complete MathML snippet of every formula.

4 Harvesting the nLab

This section is about the implementation of the nLab Harvester, the lower right part in Figure 2. Hence, the way from the document corpus, in this case it is the nLab, to a MWS Harvest is described. To create harvests based on the contents of the nLab we used the files from this GitHub repository [Nlaa]. It contains a HTML file for every single page in the nLab. Several times a day the changes that were made, are committed to the repository to syncronize it with the state of the actual nLab. Since MWS can not directly index these HTML files because the formulas are encoded in a specific markdown language, they have to be extracted from the HTML and converted to a Parallel Markup MathML presenation.

4.1 Implementation

The source code that is described in the following can be found here [Nlac]. It is a Python script that uses the Beautiful Soup library [Bs4] for parsing the HTML files. Both of them are certainly not the best performers for that kind of task, regarding the speed, but they are easy to use and the bottle neck of the task is still the LaTeX-to-MathML conversion.

To make update processes less expensive, up to ten files source files are used to create one harvest file. So if there are changes in one source file, just the corresponding harvest file needs to be recreated. With an environment variable the harvester can be configured to periodically pull updated files from the GitHub repository and then generate new harvest files. In this way the harvest can be kept in sync with the nLab sources. Every file of the source is searched for formulas that are encoded as MathML elements. Since these elements contain no Content MathML annotation, it is necessary to convert these formulas to Content MathML. To do so, the TeX annotation of the MathML elements are extracted and sent to a LaTeXML daemon. This returns the formula as MathML element structured as Parallel Markup. From that result the Content MathML part is added as a mws:expr element to the harvest file and the complete result is, as stated above, added to the mws:data element. Additionally, all the text content from the page is extracted and is also added to the mws:data element.

4.2 LaTeXML Plugin for iTex

The nLab uses iTex [Disa] as a markdown language for the math snippets. This is similar to LaTeX but is rendered to Presentation MathML with the help of iTex2MML [Disb]. Though iTex uses similar commands as LaTeX, there are some differences in some commands. Therefore, LaTeXML could not support all of the possible commands that may appear while harvesting. To handle these differences we need a plugin for

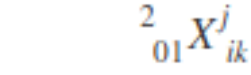

Figure 3: MathML of the iTex multiscripts command

LaTeXML, the source code can be found here [Lat]. This provides an extended version of the LaTeXML-math-profile with some additional preloaded packages and some new bindings to cover the commands that are named differently or that do not exist in that way in ordinary LaTeX. An example for a command that is supported in iTex but not in LaTeX is the multiscripts command. `\multiscripts{^2_0_1}{X}{^j_i_k}` is converted with iTex to the output shown in Figure 3 but is not a valid LaTeX/TeX command. So the plugin provides the source code, so that LaTeXML can generate the right MathML representation for that.

To allow the users of our search application to query not only with common LaTeX phrases, the query syntax parser, which is an instance of the LaTeXML daemon, also uses that plugin. Thus, the user can also use the iTex specific commands in queries.

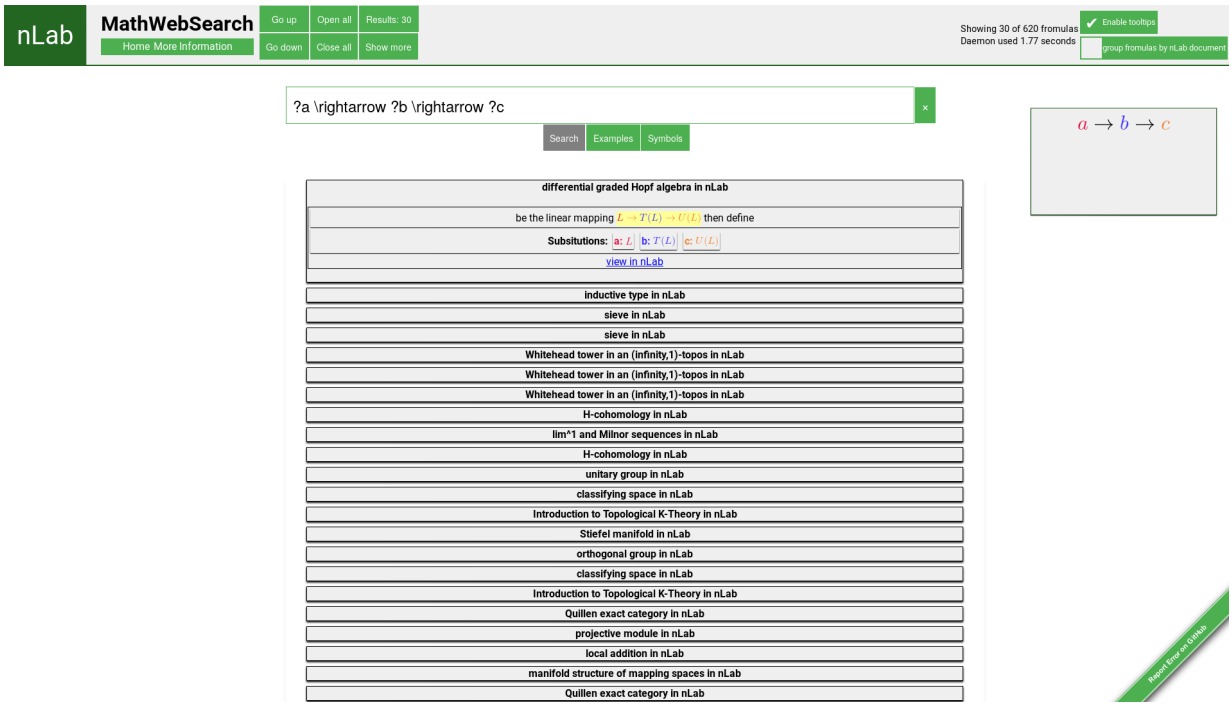

Figure 4: Screenshot of the frontend, showing all the view components [Nlag]

5 A New Frontend for MathWebSearch

It is not reasonable to let the user directly query our search engine, because this means sending and receiving JSON objects aswell as reading and writing MathML annotated XML fragments. Thus, we need some interface for the user to interact with the search engine. This user interface should run in the user's browser. In the following section the structure and the functionality of the newly developed frontend (upper left part of Figure 2) will be presented. An overview about the appearance of the frontend can be seen Figure 4 and a deployed instance can be tested at [Nlag].

These are the requirements we set for our user interface:

R1. Provide a possibility to type in some input (encoded as LaTeX phrase).

R2. Render a preview of the users input query.

R3. Send the query to the server.

R4. Present the results.

5.1 Basic Idea behind the Structure

The frontend was developed with the React framework which was mentioned in Section 3 in combination with the Typescript language. This is a programming language that is developed by Microsoft and is essentially a superset of JavaScript. It adds while development the option of static typing and is later transpiled to JavaScript [Ts], so common browsers can run the code.

The basic architecture of the frontend is inspired by the Flux architecture, which was already mentioned in Subsection 3.3. So we are having a store for the global state of the application, some action creators that allow updates of the state and some components which are providing what the user can see and interact with.

Since this is a relatively small project and the global state is not too big and complex we did not use a library that implements the Flux architecture, like Redux[8]. But some of the Flux concepts, mainly having a store and some actions to manage and distribute the global state through the application, are adapted and implemented by using the React Hooks API. This also should provide the option to make an easy transition to such a library in the future, if that becomes relevant.

The following description of the architecture of the frontend sticks near to the structure of the code. We discuss the store, the connection to backend and the different view components in the coming subsections. The code of the implementation can be found here [Mwsd].

5.2 Store/Global State

The global state of the frontend consists of the current search query that the user has typed in, the MathML representation of this query, that was received from the LaTeXML backend, the results that were gathered from the mwsapi or directly from the MWS server, when sending this query to it and some metainformation like the number of all results that match this query and the current pagination of the results that were requested so far.

The store is created with the React function createContext[9] with some initial state, that is more or less empty except from a possible first query that can be obtained from the url with which the site was accessed. With the React function useContext the store is provided along with the dispatcher throughout the application. So in every component the data in the store can be accessed or new data can be sent to the store.

To manipulate the state there is a variety of actions. These are JavaScript objects containing a type element which indicates the dispatcher what kind of action should be performed, and a payload element which contains the new data that should be added to the state. As an example the action to update the input text can be seen in Listing 2. With this, the new input text from the user, stored in the variable new_text, goes to the state and the formula

[8] https://redux.js.org/

[9] https://reactjs.org/docs/context.html

```
{
  type: "UPDATE_INPUT_TEXT",
  payload: {input_text: new_text, input_formula: null }
}
```

Listing 2: Action to update the input text

is reset to trigger a new conversion. Other actions will be: convert the LaTeX phrase to a MathML snippet, search for a query and so on. They are all following this schema of having a type and a payload and serving as input for the above-mentioned dispatch function.

While these actions are created in the corresponding action created, it is possible to make an asynchronous request to the backend. For example when the convert action is created, which is for converting the input text to a MathML snippet, there is a request sent to the LaTeXML daemon. How this is implemented will be described in the next section.

5.3 Communication with the Backend

The basic idea behind the structure of backend calls is to have an abstract class that implements the Fetch API calls[10] to the backend. Because this is a bunch of boilerplate code and is needed for every kind of server requests. Then some specialized classes can inherit from this base class and implement some specific code for packing or unpacking a request. So we have a client class for the requests to the LaTeXML daemon and a client class for the requests to the mwsapi daemon. To stay as flexible as possible, the frontend also implements a client for requests to a plain MWS daemon, so it is possible to run the frontend even without the mwsapi in between.

This was done to reduce redundancy of code, e.g. calling fetch, the function to send (HTTP-) requests, and to have a clean interface for the backend calls. This also fulfils our requirement R3, since the code allows us to send queries to the server.

5.4 (View-) Components

The presentation of the data in the global store and the possibilities to provide some user input is split up in different React function components. These are described in this subsection.

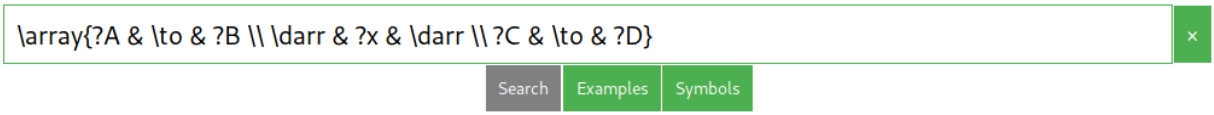

Figure 5: Searchbar with a query

5.4.1 Searchbar

The Searchbar is a React function component that provides a HTML form and can be seen in Figure 5. The form is the input field for the user, so it is possible to type in some search queries. When a user types in some new text, the action for updating the input text is created and the global state is updated. When the submission event of the form is triggered, a search action is created, which leads to a search request to the MWS backend. Optionally, two kinds of buttons can be included. These provide a dropdown menu. One that shows a list of same example queries, and another that shows a list of commands for some special symbols or characters so that they can be added easily to a query. The queries are some normal LaTeX expressions that present some mathematical formula, with one difference that can be seen in Figure 5: the characters with a preceding question mark. These are converted to the query variables and are provided by LaTeXML-Plugin for MWS [Mwsc] (see Figure 6). This component meets our requirement R1, since this allows the user to insert search queries.

5.4.2 Preview Window

The Preview Window is the component that shows the user how his input query, which is a LaTeX phrase, looks like, after it is converted to an MathML snippet (see Figure 6). If the global state receives a new input text, the action to convert that new input text is triggered. If this is successfully done, the formula is rendered and presented to the user, otherwise an error message appears in this field. In this step, the different query variables are differently colored. The same color are also used in the presentation of the results, to color an explicit subterm in a search hit. This feature did not exists in the old frontend and it helps the user to retrace in a search result which query variable is substituted with which subformula. This component completes our requirement R2, since we show a preview of the user's query.

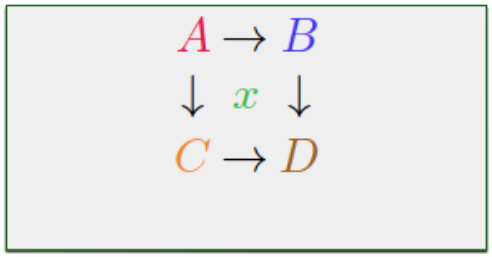

Figure 6: Preview of a user query

[10] `https://developer.mozilla.org/en-US/docs/Web/API/Fetch_API`

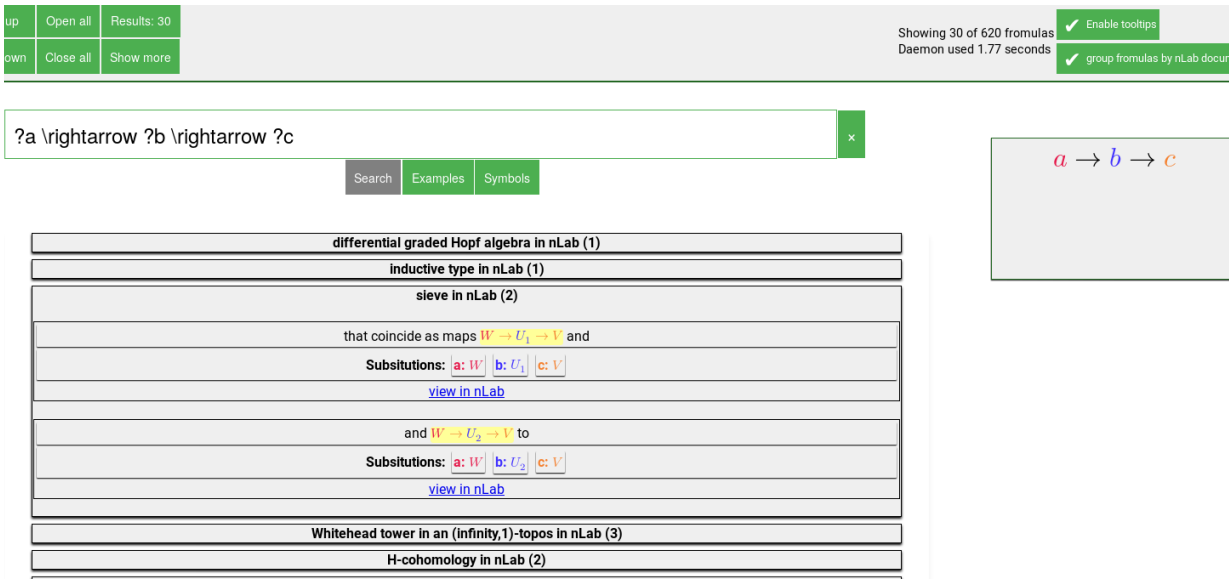

Figure 7: Grouped search results [Nlag]

5.4.3 List of Results

If a new set of formulas, that were extracted from an answer of the backend, was added to the store, it can be presented to the user (see Figure 4). For every formula that was found, an entry in the list of results is created. To keep the view as clear as possible initially for every formula, only the title of the page where it belongs to, is shown. To gather more information about a search hit, the user can click on such an entry to expand it. Then the complete formula and the substitution for every query variable is presented. Like in the preview window the variables or rather their substitutions are also colored in the corresponding color, to make visualization of the substitution clearer. As in Figure 4 shown, the query variable a in the preview window is colored red and in the first search hit the corresponding substitution L is colored red as well.

Up to ten words before and after this formula are shown, if the harvest files contain the text from the source files, to give some small impression of the context (see Figure 7). If the harvests contain links to the origin of the formulas, then this is also presented to the user. For example, in Figure 4 or Figure 7 the View in nLab element provides a Hyperlink to the corresponding nLab page.

Since it happens very often that multiple formulas from the same source page were hit in one query, for example see the multiple occurrence of sieve in nLab in Figure 4, there is the option to group all the results by the document they come from. This is done by checking the button labeled with *group by nLab document* as shown in the upper right corner of Figure 7 and Figure 4. The number next to the name of page indicates how many formulas in the document were hit with the search query. This is a feature that is introduced in the new frontend and did not exist in the old one and it helps to have a clear

(a) The Buttons, the user can interact with

(b) Checkboxes and statistics

Figure 8: The two parts of header

aggregation of the results. With the help of this component the user has a clear overview of the results that were gathered for his query, so this completes the requirement R4.

5.4.4 Interaction Panel in the Header

To have a single place for all the interaction possibilities, except these that modify the searchbar, we created a sticky header that contains all these buttons. This header always sticks to the top of the page and is always visible (see Figure 4).

Such a panel is also something that did not exist in the old frontend. It includes some buttons that improve the users experience. They are shown in Figure 8a. The buttons labeled with *Go Up*/*Go Down* are jumping directly to the top or the bottom of the page. This essentially convenient for navigating when there is already a long list of results. For instance, the user can jump with one click to the top to enter a new search query.

The buttons labeled with *Open All*/*Close All* are there for opening or closing all the entries in the result list with a single click. This is convenient if the user already opened at lot of results and he wants be get back a clear overview.

The button named with *Results: 10* allows the user to set the numbers of results a single search query should request. Below that comes the *Show more* button, that appears only if there are more search results available and allows the user to trigger the next fetch request to get more results.

The right corner of the header contains some statistics and then two checkboxes (for a detailed view see Figure 8b). The statistics show the user how many of the available formulas for the current query are already shown in the result list and how long the daemon needed to process the last query. The upper checkbox is for enabling or disabling the tooltips, which are shown when hovering above a button and provide some hints for the usage. The lower one is for grouping the entries in the result list, like explained in subsubsection 5.4.3.

5.5 Configuration Options

To keep the frontend as general as possible there are some configuration options. As mentioned earlier, it is possible to choose if the frontend is powered by a mwsapi daemon or a plain instance of the MWS daemon. Furthermore, the name of the branding, which was the nLab in the figures above, can be easily customized. It is also possible to disable the buttons that show the examples or the symbols. To change the contents these buttons provide, it is just necessary to modify a JSON file. Along with that, there are a set of predefined color schemes, that can be chosen, therefore the appearance can be adjusted to the contents used. All this customizations are done with environment variables, which is especially convenient when the frontend is used in a docker container. So this frontend can be easily used in other deployments of the MWS system and can be adjusted to different needs.

6 Conclusion

In this thesis I presented a formula search application for the nLab based on the MWS search engine and LaTeXML with a custom plugin for the formula conversion and a new React based user interface.

This is a major improvement of the MWS ecosystem, since the new frontend is not specific to the nLab. Thus, it will be easy to deploy a MWS instance on another corups in the future. Because it is just needed to provide a harvester and to plug the set of docker containers together. A further improvement regarding the frontend is that the communication with a mwsapi layer is possible as well as the communication with a plain MWS instance.

For the user we introduced some convenient functionalites like the option to group the formula hits by the documents they belong to or the possibility to open or close all entries at once. Furthermore, the subsituiton variables and the corresponding (sub-) formulas are now highlighted in different colors, so it is easy to see which query variable was substituted by which term in the search hit.

6.1 Chasing Diagrams

The big question that comes to mind is whether it is possible to chase a diagram? So which results do we get if we query for such a diagram, as we saw in the Introduction. As we can see in Figure 9, it is possible to find diagrams with the help of MWS. We have even found the most general page containing a diagram in the nLab along with 328 other diagrams that are shaped like this.

Let us remind the $z : 1 \to N$ morphism from the NNO, the example from the introduction. So if we want to look for more categories or concepts that also have such a morphism, we may search for $1 \to ?X$. Then we find the NNO in nLab page, but also the page about recursion, the successor and the real numbers object (see Figure 10). So we find a lot of information that obviously relates to our initial question and is not directly linked in the article about the NNO.

Even though this seems to work quite well, there are some deep-set problems which are related to the input data of the MWS engine.

Disambiguation Since the inputs are the mathematical expressions of the corpus it is necessary that they are extracted and converted properly. It is a problem that the semantics of such an expression is really hard to disambiguate. Although LaTeXML works quite well, there are still some more or less obvious issues. A prominent example is that LaTeXML produces for $f(x)$, $f * x$ and fx the same Content MathML output, even though they most likely mean different things. That leads to the issue that if we search for the one matter, we also get hits for the other matter. In Figure 11 it was probably searched

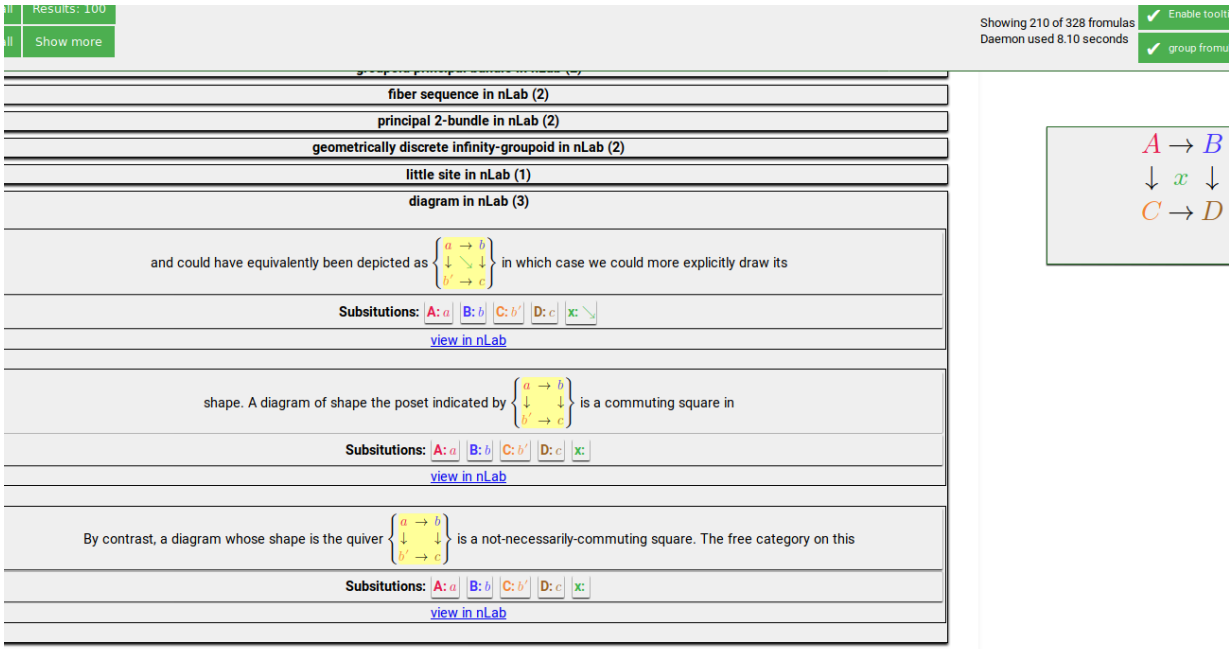

Figure 9: We found the diagram

for a function application for an arbitrary function f and an arbitrary parameter x. But the second hit was the page of the logarithm and the formula '$e = 2.71828182845\ldots$'. So while the search process the $?f$ was replaced with the '2.71828182845' and the $?x$ with the '...' but with having in mind that the user most likely searched for a function application this is a bad or really confusing result.

Table-like Environments A similar example is that LaTeXML treats also the most table-like environments (like matrix, array and align) in the same way. That leads to the problem that if an aligned environment is used as an aligned equation like this:

$$\begin{aligned} 2 &= 1 + 1 \\ 1 + 1 &= 2 \end{aligned}$$

It can be found with the query

$$\begin{matrix} ?a & ?b \\ ?c & ?d \end{matrix}$$

but not with a query like $?a = ?b$. So in this case the information that the first environment encodes some equations is lost and is treated like a normal matrix. An example for that is shown in Figure 12. Here a 2×2-Matrix was queried but the first result is something that is obviously an aligned equation. These two issues are related to the problem that often similar looking formulas do not have the same meaning.

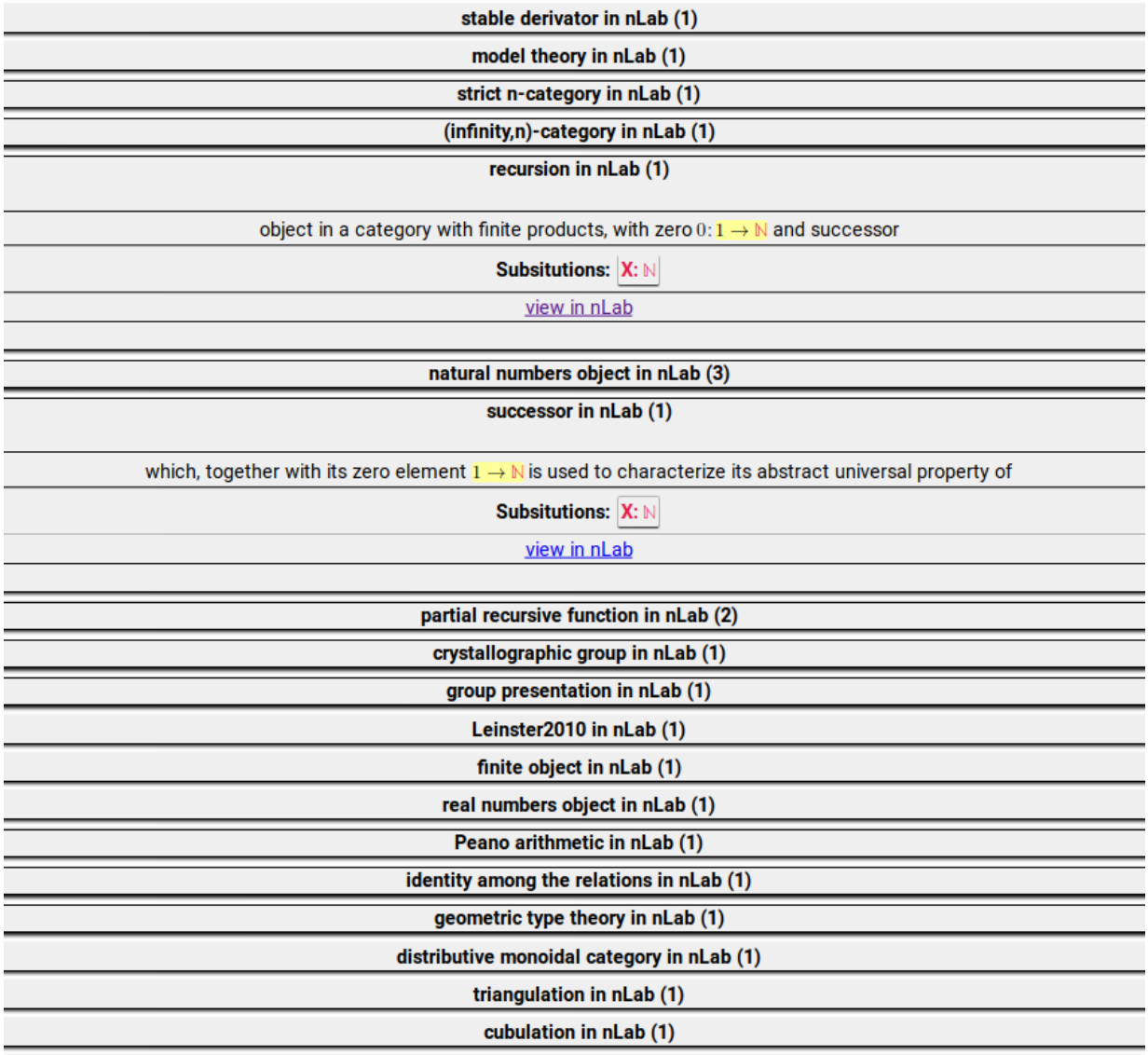

Figure 10: Results for search query for $1 \to ?X$

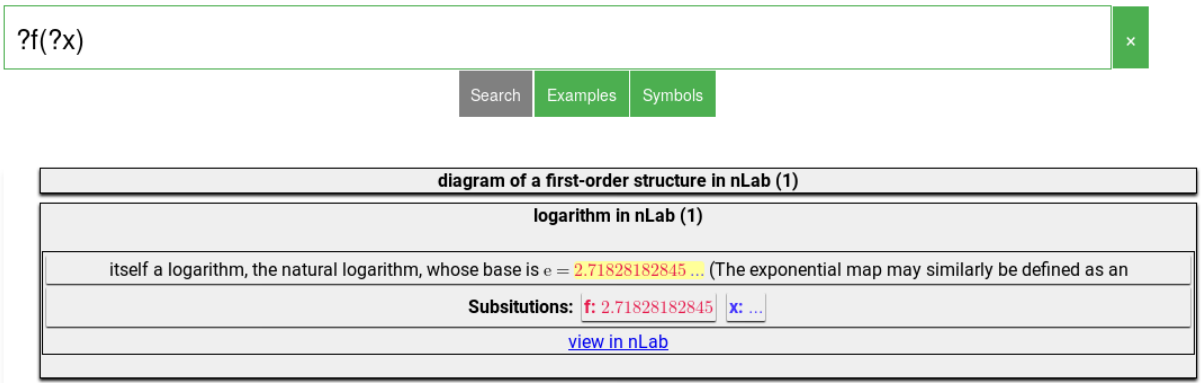

Figure 11: Problems with disambiguation

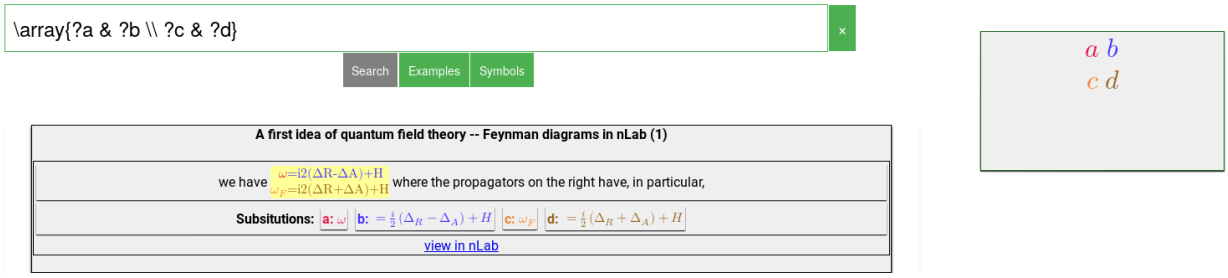

Figure 12: Search query for 2×2-Matrix with a confusing result

6.2 Future Work

The work presented in this thesis makes a significant step towards a usable MWS system. But there is still some room for improvements, so now follows a discussion of the most pressing concerns.

Internationalization A next step to improve the frontend would be to implement a feature that makes internationalization possible, so the possibility for the user to change the language of the interface.

Better Disambiguation As shown in Subsection 6.1, there are issues with the disambiguation of the mathematical expressions in the nLab corpus. These issues are related to the problem that mathematical expressions often look similar but mean completely different things depending on the context or the usage. The opposite problem, that symbols that mostly meaning the same but looking a bit different are treated differently, also occurs sometimes. For instance, the queries $?a \to ?b$ and $?a \longrightarrow ?b$ do not produce the same output, but in the most cases there is no special semantic meaning in using a long arrow over a shorter one. So it is shown that a formula search engine is in the practical use only as good as the data it is fed with. Therefore it might be worth to try some different methods of the

conversion for the LaTeX or Presentation MathML encoded formulas to Content MathML.

Result Ranking A feature that would improve the user experience of MWS is a ranking functionality of the results. Up to now, the results are just returned in the order in which they are found in the index, but not ordered in some relation to the search query. There exist some approaches to measure the similarity between a formula and the query formula and use that as ranking function (e.g. [ZZ19]). But ranking on formula level is really hard, because the meaning of a formula depends strongly on the context were it appears. The comparison of weather two or more formulas are similar is therefore still an open research question. So it might be interesting to evaluate if ranking on document level works. This might allow to use a centrality measure or PageRank to rank the documents.

Document Retrieval For this it is necessary that the MWS responses are also on document granularity and not on formula granularity. It might be an interesting approach to make not only formula retrieval, but also document retrieval with MWS possible.

Currently, the complete corresponding data node from the harvest document is returned by MWS for every formula, which matches the query. Thus, if there is a document with a lot of formulas and a search query hits multiple formulas in this document, for every hit MWS sends the complete data node. This causes some really long response times if a document with more than a thousand formulas is hit several times (e.g. around 22 seconds for a query that only contains 10 formula hits, where the decoding of the JSON takes that long). To make sure to return a data node only once in a search query would reduce this problem.

Combined Keyword and Formula Search Another extension that might be quite useful, would be to allow also the combination of formula and keyword search. The combination of MWS and an Elasticsearch instance to do that, the so-called TeMaSearch [D6.116; HK15], already exists. But this is also hindered by the difference in the result granularity, since Elasticsearch returns a list of documents and MWS a list of formulas. If the backend supports that option, then the frontend should be extended to provide the possibility to input these keywords.

Acknowledgements

I would like to thank Prof. Kohlhase for his support and the valuable suggestions. Further, I would like to thank Tom Wiesing for the support in implementing and deploying the frontend. I would also like to thank Deyan Ginev for the tips and hints on working with LaTeXML. A special thank goes to the proofreaders of my work: Franziska and Tobias.

References

[Aiz+14] Akiko Aizawa et al. "NTCIR-11 Math-2 Task Overview". In: *NTCIR 11 Conference*. Ed. by Noriko Kando, Hideo Joho, and Kazuaki Kishida. Tokyo, Japan: NII, Tokyo, 2014, pp. 88–98. URL: http://research.nii.ac.jp/ntcir/workshop/OnlineProceedings11/pdf/NTCIR/OVERVIEW/01-NTCIR11-OV-MATH-AizawaA.pdf.

[Aiz+16] Akiko Aizawa et al. "NTCIR-12 MathIR Task Overview". In: *Proceedings of the 12th NTCIR Conference on Evaluation of Information Access Technologies*. Ed. by Noriko Kando, Tetsuya Sakai, and Mark Sanderson. Tokyo, Japan: NII, Tokyo, 2016, pp. 299–308. URL: http://research.nii.ac.jp/ntcir/workshop/OnlineProceedings12/pdf/ntcir/OVERVIEW/01-NTCIR12-OV-MathIR-ZanibbiR.pdf.

[Ber+19] Katja Berčič et al. *Report on OpenDreamKit deliverable D6.10: Towards Mathematical Data as VRE components*. Tech. rep. OpenDreamKit, 2019. URL: https://github.com/OpenDreamKit/OpenDreamKit/raw/master/WP6/D6.10/report-final.pdf (visited on 10/17/2019).

[Bra+08] Tim Bray et al. *Extensible Markup Language (XML) 1.0 (Fifth Edition)*. W3C Recommendation. World Wide Web Consortium (W3C), Nov. 26, 2008. URL: http://www.w3.org/TR/2004/REC-xml-20081126.

[Bs4] *Beautiful Soup Documentation*. URL: https://www.crummy.com/software/BeautifulSoup/bs4/doc/ (visited on 07/22/2019).

[D6.116] Michael Kohlhase and Alexandru Glontaru. *Full-text Search (Formulae + Keywords) over LaTeX-based Documents*. Deliverable D6.1. OpenDreamKit, 2016. URL: https://github.com/OpenDreamKit/OpenDreamKit/raw/master/WP6/D6.1/report-final.pdf.

[Dav+19] Kenny Davila et al. "Tangent-V: Math Formula Image Search Using Line-of-Sight Graphs". In: *European Conference on Information Retrieval*. Springer. 2019, pp. 681–695.

[Disa] Jacques Distler. *iTex Commands*. URL: https://golem.ph.utexas.edu/~distler/blog/itex2MMLcommands.html (visited on 07/22/2019).

[Disb] Jacques Distler. *itex2MML*. URL: https://golem.ph.utexas.edu/~distler/blog/itex2MML.html (visited on 12/01/2019).

[Es] *ElasticSearch Homepage*. URL: https://www.elastic.co/ (visited on 10/29/2019).

[Fbfa] *Flux - GitHub Repository*. URL: https://github.com/facebook/flux (visited on 11/29/2019).

[Fbfb] *Flux Application architecture for building user interfaces*. URL: https://facebook.github.io/flux/ (visited on 09/17/2019).

[Gra95] Peter Graf. "Substitution tree indexing". In: *Rewriting Techniques and Applications*. Ed. by Jieh Hsiang. Berlin, Heidelberg: Springer Berlin Heidelberg, 1995, pp. 117–131. ISBN: 978-3-540-49223-8.

[GSC15] Ferruccio Guidi and Claudio Sacerdoti Coen. "A Survey on Retrieval of Mathematical Knowledge". In: *Intelligent Computer Mathematics 2015*. Conferences on Intelligent Computer Mathematics (Washington DC, USA, July 13, 2015–July 17, 2015). Ed. by Manfred Kerber et al. LNCS 9150. Springer, 2015, pp. 296–315. ISBN: 978-3-319-20615-8. DOI: 10.1007/978-3-319-20615-8_20.

[HK15] Radu Hambasan and Michael Kohlhase. "Faceted Search for Mathematics". In: *Proceedings of the LWA 2015 Workshops: KDML, FGWM, IR, and FGDB*. Ed. by Ralph Bergmann, Sebastian Görg, and Gilbert Müller. Oct. 2015, pp. 33–44. URL: http://ceur-ws.org/Vol-1458/D05_CRC1_Hambasan.pdf.

[ISO] *ISO/IEC 40314:2016(en) Information technology — Mathematical Markup Language (MathML) Version 3.0 2nd Edition*. URL: https://www.iso.org/obp/ui/#iso:std:iso-iec:40314:ed-1:v1:en (visited on 08/04/2019).

[Jsx] *Introducing JSX*. URL: https://reactjs.org/docs/introducing-jsx.html (visited on 11/12/2019).

[KMP12] Michael Kohlhase, Bogdan A. Matican, and Corneliu C. Prodescu. "MathWebSearch 0.5 – Scaling an Open Formula Search Engine". In: *Intelligent Computer Mathematics*. Conferences on Intelligent Computer Mathematics (CICM) (Bremen, Germany, July 9, 2012–July 14, 2012). Ed. by Johan Jeuring et al. LNAI 7362. Berlin and Heidelberg: Springer Verlag, 2012, pp. 342–357. ISBN: 978-3-642-31373-8. URL: http://kwarc.info/kohlhase/papers/aisc12-mws.pdf.

[Koh14] Andrea Kohlhase. "Math Web Search Interfaces and the Generation Gap of Mathematicians". In: *Mathematical Software - ICMS 2014 - 4th International Congress*. Ed. by Hoon Hong and Chee Yap. Vol. 8592. LNCS. Springer, 2014, pp. 586–593. ISBN: 978-3-662-44198-5. DOI: 10.1007/978-3-662-44199-2_88.

[Koh16] Andrea Kohlhase. "Math Web Search Interfaces and the Generation Gap of Mathematicians". In: *Workshop Human-Computer Algebra Interaction, Kassel, Germany*. 2016. URL: http://minimair.org/hcai2016/math_web_search.pdf.

[KP] Michael Kohlhase and Corneliu Prodescu. *MathWebSearch Manual*. Web Manual. Jacobs University. URL: https://github.com/KWARC/mws/doc/manual/manual.pdf (visited on 04/07/2012).

[KŞ06] Michael Kohlhase and Ioan Şucan. "A Search Engine for Mathematical Formulae". In: *Proceedings of Artificial Intelligence and Symbolic Computation, AISC'2006*. Ed. by Tetsuo Ida, Jacques Calmet, and Dongming Wang. LNAI 4120. Springer Verlag, 2006, pp. 241–253. URL: http://kwarc.info/kohlhase/papers/aisc06.pdf.

[Lat] *A LaTeXML plugin for adding the iTeX2MML utility.* URL: https://github.com/MathWebSearch/LaTeXML-plugin-iTeX2MML (visited on 07/22/2019).

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

[Man+19] Behrooz Mansouri et al. "Tangent-CFT: An Embedding Model for Mathematical Formulas". In: *Proceedings of the 2019 ACM SIGIR International Conference on Theory of Information Retrieval*. ACM. 2019, pp. 11–18.

[Mwsa] *A Docker Container responsible for automatically updating a MathWebSearch instance.* URL: https://github.com/MathWebSearch/mws-cron (visited on 09/23/2019).

[Mwsb] *A golang library and set of tools to setup, query and maintain a MathWebSearch and ElasticSearch instance for use within TemaSearch.* URL: https://github.com/MathWebSearch/mwsapi (visited on 07/18/2019).

[Mwsc] *A LaTeXML extension for generating MathWebSearch queries from TeX.* URL: https://github.com/MathWebSearch/LaTeXML-Plugin-MathWebSearch (visited on 08/21/2019).

[Mwsd] *MathWebSearch frontend.* URL: https://github.com/MathWebSearch/mws-frontend (visited on 08/01/2019).

[Mwse] *MWS Harvests GitHub Wiki.* URL: https://github.com/MathWebSearch/mws/wiki/MWS-Harvests (visited on 07/18/2019).

[Mwsf] *mws-indexer - A script that can safely update the index used by MathWebSearch.* URL: https://github.com/MathWebSearch/mws-indexer (visited on 09/23/2019).

[MY03] Bruce R. Miller and Abdou Youssef. "Technical Aspects of the Digital Library of Mathematical Functions". In: *Annals of Mathematics and Artificial Intelligence* 38.1-3 (2003), pp. 121–136. URL: citeseer.ist.psu.edu/599441.html.

[MZO19] Behrooz Mansouri, Richard Zanibbi, and Douglas W Oard. "Characterizing Searches for Mathematical Concepts". In: *2019 ACM/IEEE Joint Conference on Digital Libraries (JCDL)*. IEEE. 2019, pp. 57–66.

[Nlaa] *A file-based mirror of the nLab wiki (HTML format).* URL: https://github.com/ncatlab/nlab-content-html (visited on 07/18/2019).

[Nlab] *diagram chasing.* URL: https://ncatlab.org/nlab/show/diagram+chasing (visited on 11/14/2019).

[Nlac] *Harvester for the nLab.* URL: https://github.com/MathWebSearch/nlab_harvester (visited on 07/18/2019).

[Nlad] *nLab.* URL: https://ncatlab.org/nlab/show/HomePage (visited on 10/02/2019).

[Nlae] *nLab - Natural Numbers Object.* URL: https://ncatlab.org/nlab/show/natural+numbers+object (visited on 11/30/2019).

[Nlaf] *nLab - nPOV*. URL: https://ncatlab.org/nlab/show/nPOV (visited on 10/17/2019).

[Nlag] *nlabSearch - A MathWebSearch instance for the nLab*. URL: https://nlabsearch.mathweb.org (visited on 09/18/2019).

[Oli+17] Ricardo M Oliveira et al. "A distributed system for SearchOnMath based on the Microsoft BizSpark program". In: *arXiv preprint arXiv:1711.04189* (2017).

[Reaa] *React A JavaScript library for building user interfaces*. URL: https://reactjs.org/ (visited on 09/17/2019).

[Reab] *React Hook Introduction*. URL: https://reactjs.org/docs/hooks-intro.html (visited on 09/17/2019).

[Sch+18] Moritz Schubotz et al. "Improving the representation and conversion of mathematical formulae by considering the textual context". In: *TUGBoat* 39.3 (2018), pp. 228–240.

[See18] Johannes-Sebastian See. "Building Mathhub using React". Bachelor's Thesis. Friedrich-Alexander University Erlangen-Nuernberg, 2018. URL: https://github.com/AiraComet/BachelorThesis/blob/master/thesis.pdf.

[SL11] Petr Sojka and Martin Líška. "Indexing and Searching Mathematics in Digital Libraries – Architecture, Design and Scalability Issues." In: *Intelligent Computer Mathematics*. Ed. by James Davenport et al. LNAI 6824. Springer Verlag, 2011, pp. 228–243. ISBN: 978-3-642-22672-4.

[Ts] *Typescript*. URL: https://www.typescriptlang.org/ (visited on 09/17/2019).

[YM19] Abdou Youssef and Bruce R Miller. "Explorations into the Use of Word Embedding in Math Search and Math Semantics". In: *International Conference on Intelligent Computer Mathematics*. Springer. 2019, pp. 291–305.

[ZZ19] Wei Zhong and Richard Zanibbi. "Structural Similarity Search for Formulas Using Leaf-Root Paths in Operator Subtrees". In: *Advances in Information Retrieval*. Ed. by Leif Azzopardi et al. Cham: Springer International Publishing, 2019, pp. 116–129. ISBN: 978-3-030-15712-8. DOI: 10.1007/978-3-030-15712-8_8.