Towards a Heterogeneous Query Language for Mathematical Knowledge*

Katja Berčič[1][0000−0002−6678−8975], Michael Kohlhase[1][0000−0002−9859−6337], and Florian Rabe[1][0000−0003−3040−3655]

Computer Science, FAU Erlangen-Nürnberg

Abstract. With more than 120.000 articles published annually in mathematical journals alone, mathematical search has often been touted as a killer application of computer-supported mathematics. But the artefacts of mathematics – e.g. mathematical documents, formulas, examples, algorithms, concrete data sets, or semantic web-style graph abstractions – that should be searched cover a variety of aspects. All are organized in complex ways and offer distinct challenges and techniques for search. Existing representation languages, the corresponding query languages and search systems usually concentrate on only one of these aspects. As a consequence, each system only partially covers the information retrieval needs of mathematical practitioners, and integrated solutions allowing multi-aspect queries are rare and basic.
We present an architecture for a generic multi-aspect search system and analyze the requirements on paradigmatic practical information retrieval needs.

1 Introduction and Related Work

Motivation Computers and Humans have complementary strengths: Computers can handle large data and computations flawlessly at enormous speeds. Humans can sense the environment, react to unforeseen circumstances and use their intuitions to guide them through only partially understood situations. We speak of a **horizontal** task if it involves systematically sifting through large volumes of data or carrying out large computations. This contrasts with a **vertical** task, which involves intricately combining multiple previously unclear methods in a limited domain. In general, humans excel (only) at vertical tasks, while machines excel (only) at horizontal ones. For example, in mathematics humans explore mathematical theories and come up with novel insights/proofs but may delegate symbolic/numeric computation, proof checking/search, data storage, and typesetting of documents to computers.

A general goal is to develop solutions for horizontal problems in mathematics and dovetail the solutions into the vertical workflows of practicing mathematicians. One of the most important horizontal problems is Mathematical

* The authors were supported by DFG grant RA-1872/3-1, KO 2428/13-1 OAF and EU grant Horizon 2020 ERI 676541 OpenDreamKit.

Information Retrieval (MIR), i.e., finding mathematical objects with particular properties — e.g. a counterexample, a theorem that allows rewriting a formula into a more tractable form, an article that describes a method applicable to a current problem, or an algorithm that computes a particular value.

Despite significant efforts and successes, current MIR systems are far behind practical needs. It is not even clear how to best design a good MIR system. Many existing mathematical tools are highly specialized, e.g., into proof assistants, computer algebra systems, mathematical databases, or narrative languages like LaTeX or HTML+MathML. Current MIR solutions often exploit this specialization by custom-fitting indexing and querying solutions to the data model of the tool, e.g., using substitution tree indexing for a set of theorems or SQL queries for a mathematical database. But MIR is often needed outside such a tool, e.g., imagine a mathematics-aware Google-like interface that finds semantically relevant results in the Coq library, the arXiv, and the OEIS. Thus, the question arises how to query heterogeneous mathematical knowledge, i.e., how to design representation and query languages that allow finding results in many different libraries using vastly different representation languages.

Contribution We present the high-level design of an indexing and querying infrastructure that we believe to be an interesting candidate for a comprehensive solution. We cannot provide a detailed scalable solution at this point. In fact, we believe more conceptual and experimental research is needed before that would be feasible.

Concretely, our design is based on the ideas of [Car+20a], which classify mathematical libraries and objects by five aspects: deductive, computational, narrative, databases containing concrete objects, and organizational ontologies — a classification that matches existing tools and optimized indexing and querying solutions quite well. Our key ideas are: (i) While every library typically has one primary aspect (e.g., deductive for a Coq library), it may contain objects of any other aspect as well (e.g., narrative comments). (ii) Our solution is centered around a set of specialized indexes (one per aspect), and indexing a library generates entries in each of these indexes. (iii) For each index, existing solutions provide optimized querying support (e.g., SQL for concrete databases), and these supply the atomic queries of a comprehensive MIR system. (iv) Complex queries arise by combining atomic ones (e.g., intersection), and query evaluation is based on decomposing a query into atomic ones that are executed by the respective tools. This paper is a short version of [BKR20], which has more details and examples.

Related Work **Information retrieval** (IR) is the activity of obtaining information relevant to an **information need** from a collection of resources. In MIR, both the resources and the information need are mathematical in nature. Current approaches to MIR have mostly been technology-oriented, focusing either on formula search or on adapting traditional IR techniques to include formula data. [GSC15] gives a survey and [Aiz+16] a description of the NTCIR MIR challenges. An exception to this is the work reported in [ST16; Sta+18] which

concentrates on extracting semantic/mathematical information from mathematical documents and then use it for information retrieval.

Our architecture can be seen as a variant of the data integration system using a Global-as-View schema mapping in the sense of [DHI12], which combines different relational databases. They use a query language based on what they call the mediated global schema, which is induced as the union of the local schemas under a database view. That kind of heterogeneity problem is much simpler because it involves only mediating between different schemas in a fixed aspect (namely relational databases) whereas the MIR problem requires mediating across different aspects. It remains an open question whether such SQL-specific solutions can be applied directly to MIR: the awkwardness of encoding knowledge of the other aspects in SQL may be offset by the high levels of optimization in existing solutions such as Apache Drill [AD]; but also see [Cho+05].

Overview In the next section we will show that mathematical resources and information needs have more aspects than the formulas and words used in MIR so far. In Section 3 we present an architecture for a generic multi-aspect representation and search system, in Section 4 we discus indexing concrete mathematical data, and in Section 5 we specify a cross-aspect query language for MIR. Section 6 concludes the paper.

A Multi-Aspect Library The Online Encyclopedia of Integer Sequences (OEIS) [Slo03; OEIS], a popular web portal that contains information on more than 300.000 integer sequences, is an example of a mathematical library whose contents range over multiple aspects.

Internally, the OEIS uses a line-based text format to represent this information. Listing 1.1 shows a fragment of the representation for the Fibonacci numbers. Lines are prefixed by a classifier letter (%I for identifiers, %S for a prefix of the sequence, %N for the "name", %C for comments, %D for references, %A for the OEIS author, and %F for formulae) that indicates the **item class**.

```
%I A000045 M0692 N0256
%S A000045 0,1,1,2,3,5,8,13,21,34,55,89,144,233,377,610,987
%N A000045 Fibonacci numbers: F(n) = F(n-1) + F(n-2) with F(0) = 0 and F(1) =
     1.
%C A000045 Also sometimes called Lamés sequence.
%D A000045 V. E. Hoggatt, Jr., Fibonacci and Lucas Numbers. Houghton, Boston,
     MA, 1969.
%F A000045 F(n) = ((1+sqrt(5))^n-(1-sqrt(5))^n)/(2^n*sqrt(5))
%F A000045 G.f.: Sum_{n>=0} x^n * Product_{k=1..n} (k + x)/(1 + k*x). - _Paul
     D. Hanna_, Oct 26 2013
%F A000045 This is a divisibility sequence; that is, if n divides m, then a(n
     ) divides a(m)
%A A000045 _N. J. A. Sloane_, Apr 30 1991
```

Listing 1.1. OEIS Sources for Sequence A000045 (Fibonacci Numbers)

The OEIS portal features a simple boolean search engine which allows to search for sequences by OEIS ID, name, keywords, and subsequence (this can contain anonymous wildcards for integers and subsequences). Additionally, atomic queries can be qualified by **prefixes** that restrict keywords to the various classes

of items or change the sequence matching algorithm (e.g. from signed to unsigned equality on components). The query results of this query are a sequence of complete presentations of the sequence information ordered by "relevance", which combines match quality, sequence popularity and number. There is a variant called superseeker (an e-mail server) that " tries hard to find an explanation for a number sequence" combining information from the OEIS and other sources.

2 Aspects of Math Resources and Information Needs

In [Car+20a] we have identified the following five basic **aspects** of mathematics:

i) **Inference**: deriving statements by *deduction* (i.e., proving), *abduction* (i.e., conjecture formation from best explanations), and *induction* (i.e., conjecture formation from examples).
ii) **Computation**: algorithmic manipulation and simplification of mathematical expressions and other representations of mathematical objects.
iii) **Concretization**: generating, collecting, maintaining, and accessing collections of examples that suggest patterns and relations and allow testing of conjectures.
iv) **Narration**: bringing the results into a form that can be digested by humans, usually in mathematical documents like articles, books, or preprints, that expose the ideas in natural language but also in diagrams, tables, and simulations.
v) **Organization**, i.e., the modular structuring of mathematical knowledge.

These aspects — their existence and importance to mathematics — should be rather uncontroversial. Figure 1 illustrates their tight relation: we locate the organization aspect at the centre and the other four aspects at the corners of a tetrahedron, since the latter are all consumers and producers of the mathematical knowledge represented by the former. [Car+20b] gives a survey of paradigmatic mathematical software systems by the five aspects they address.

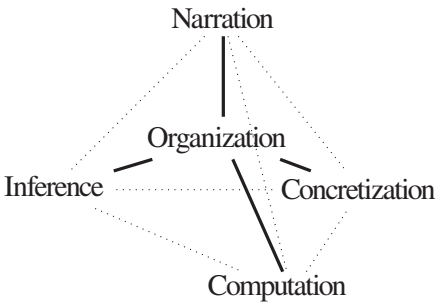

Fig. 1. Five Aspects of Math Artefacts

We use the term **symbolic** to cover deductive (aspect Inference) or computational (aspect Computation) in this paper. While these libraries are pragmatically very different and are thus distinguished in the classification above they can be treated in the same way for the purpose of search. Coming back to OEIS, we see that it contains all five aspects of mathematical knowledge:

1. symbolic knowledge: the formulae, even though in this case they are informal ASCII art; there is also computer code,
2. concrete knowledge: the sequence prefix,
3. narrative knowledge: the name and comments,

4. organizational knowledge: the identifiers and references.

Mathematical information needs typically involve combinations of these five aspects. A paradigmatic example is the quest for "*all published integer sequences that are not (yet) listed in the OEIS*" of an OEIS editor who wants to extend OEIS coverage. Answering this information need will involve finding integer sequences in documents (a combination of concretized and narrative knowledge), determining whether these documents are published (i.e. part of the archival literature; this involves organizational metadata), and pruning out the OEIS sequences. An OEIS user might be interested in "*the integer sequences whose generating function is a rational polynomial in* $\sin(x)$ *that has a Maple implementation not affected by the bug in module M*". This additionally involves symbolic knowledge about generating function (formula expressions), and Maple algorithms.

We take these examples as motivation to develop an approach for multi/cross-aspect information retrieval now.

3 Heterogeneous Indexing of Mathematical Libraries

We motivate and introduce some general concepts that can be seen as fundamental assumptions from which much of our proposed design is derived.

Fragments of a Library We require that libraries of any aspect define document fragments and assign unique identifiers (URIs) to them. These fragments will be *used* critically in the interface specification for query engines. In particular, query results contain at least a set of fragments that match the query (plus possibly other information, e.g., how or how well they match the query).

Identifying and Producing the fragments is natural as individual libraries typically already have a corresponding concept, e.g.:

- An organizational library already focuses on introducing concepts with unique identifiers. Each such concept is a fragment, with the same id.
- A symbolic library is structured into files which contain a tree structure of nested theories/modules/etc. whose leaves are declarations for named types, functions, etc.. Each node is a fragment with a qualified identifier. The underlying languages usually already define fragments and their identifiers in this way because they need them for intra-logical referencing.
- A concretized library is essentially a set of database tables (however the actual implementation may look like), and each table row is a fragment. In practice, typically one column serves as a key, and the triple of database, table name, and key provides the fragment identifier. For example, in many mathematical tables that contain enumerations of objects (e.g., in LMFDB [Lmf]), the key can be obtained by concatenating multiple properties of the object that, together, uniquely characterize it.
- A narrative library is structured both non-semantically into sections, paragraphs, etc. and semantically into statements like definitions and theorems.

These are often numbered in the presentation, and internally labels are used to identify them. Those are the fragments, and their identifiers.

Thus, it is straightforward to extend an existing implementation of a language L in such a way that it can produce the list of fragment-id pairs in a L-libraries. This is the basic functionality of what we call a **harvester** for L below.

Findable Objects in a Fragment Next, to describe what it means for a fragment to match a query, we assume that every fragment has some internal structure that allows defining *occurrences* of *objects* in the fragment. This is the main task of the harvester: it has to define what exactly an occurrence is and produces for each fragment the list of objects in it.

Most of the time, these objects have the same aspect as the containing library. For example, if L is a symbolic language, the most important objects are symbolic expressions such as the types of the declarations or the formulas in theorems. Similarly, in a narrative library, they are n-grams of words, and in a table-based database, they are the primitive database values in the table cells.

However, it is critical to observe that the same library may contain objects of many different aspects. In fact, libraries of any primary aspect can and in practice often do contain objects of the other aspects as well. Some of these objects work in the same way across libraries, although the concrete syntax may vary. Any library can contain:

- metadata attributing narrative or symbolic objects to a fragment,
- cross-references to fragments of any other library,
- contain narrative comments.

Other such cross-aspect objects are specific to the combination of aspects, e.g.:

- The text of a fragment of a narrative library may be interspersed with symbolic expressions. This occurs in virtually every scientific document.
- A table in a database can use a schema that declares some columns to contain objects of other aspects. These may be narrative objects represented as a string, or symbolic objects encoded as primitive database values (e.g., a polynomial encoded as a list of integer coefficients).
- An expression in a symbolic fragment may contain references to concrete objects stored externally, e.g., when using a database for persistent memoization. This can be useful in mathematical computation systems[1], which often need to handle complex pure functions.

Thus, it would be a mistake to assume that a library of aspect A is indexed in an A-index and queried with an A-query language. Instead, every library fragment F can contain objects O_i of any aspect A_i. We require that it be possible to find F as a result of queries in any aspect A_i. For example, a symbolic query (i.e., a symbolic expression with some free variables) can be matched against the symbolic objects found in F irrespective of the aspect of the library containing F.

[1] https://github.com/OpenDreamKit/OpenDreamKit/raw/master/WP6/D6.9/report-final.pdf

Heterogeneous Indexing While every library can contain objects of any aspect, the aspects of the objects may not be neglected: indexing and efficient querying differs vastly across aspects. For example, querying n-grams of words is different from querying symbolic expressions.

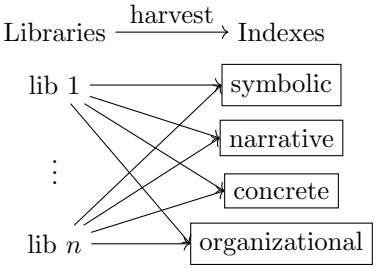

Fig. 2. Heterogeneous Indexing

From the above, we can derive the general design of an indexing infrastructure. Figure 2 gives an overview. For every library, we need to run a harvester, which returns the set of findable objects, each consisting of (i) an aspect and an object of that aspect (ii) the identifier of the containing fragment, (iii) optionally, any other information about the occurrences of the object, e.g., the position within the fragment. The findable objects of all libraries are collected and stored in aspect-specific indexes, i.e., we use one index per aspect and arbitrarily many libraries. (If this runs into scalability issues, we can federate the individual indexes, but that is an implementation issue.)

For example, these indexes could be

- a triple store like GraphDB [Eso] for *organizational* metadata and cross-references,
- a substitution tree index like MathWebSearch [HKP14] for *symbolic* objects,
- a text indexer like Elastic search [Eso] for *narrative* objects,
- a yet-to-be-developed index for *concrete* values that we discuss in Section 4.

The harvesters are specific to a library language and often integrated with the respective tool. For example, a Coq harvester could be integrated with Coq to harvest any library written in Coq while compiling it; a CSV harvester could be a stand-alone tool to harvest any concrete database represented as a CSV dump. Alternatively, if the language-specific part has already been abstracted away by exporting libraries in aspect-independent formats such as OMDoc, one can write language-independent harvesters once and for all.

In all cases, it may make sense to write four different harvesters (one for each index aspect) for the same language. For example, the Coq harvester for narrative objects may be written as a stand-alone parser of the Coq language that extracts all comments.

Indexing Induced Objects Finally, we mention an optional concept that we expect to become relevant in practice as well even though it might not be present in the first implementations: harvesters that generate new objects that did not physically occur in the fragment but logically belong to it. We call these *induced* objects. There are many instances of induced objects, e.g.:

- In an organizational library, we can take the transitive closure of a relation.
- In a symbolic language that uses some kind of inheritance or logical imports, a fragment F might be a class/module etc., and we can index also objects logically occurring in F through inheritance. That is already done routinely in many documentation generation tools, especially for object-oriented pro-

gramming languages. We also built a symbolic index like that for MMT in [IKP14].
- Sometimes, especially in deduction systems, the most feasible way to implement the harvester is to instrument the kernel. But the kernel may perform extensive normalization, in which case the index will contain the objects induced by normalization. Unfortunately, that also means it might not contain some objects originally in the library (because they are normalized away), which is a known problem with indexing deductive libraries.

4 Indexing Concrete Values

Motivation Above we have described concrete values as being stored in relational databases. Conceptually, this fits well for all the datasets we have surveyed [MDT]. In practice, the situation may differ a bit. Sometimes the information is more efficiently retrieved by on demand computation. For example, the authors of the GAP small groups library [EBO] used computer algebra system integration to bring the average space demand down to less than two bits per group without significantly sacrificing the speed of retrieval. When computation is not feasible, custom compression is often needed to manage the size of large datasets like [FL]. All of these datasets could be stored as simple tables, but that has not been seen as advantageous so far.

Even if all libraries used relational tables, an indexing and querying solution should not necessarily be built on operations such as filters and joins. These are what SQL focuses on, and work for MIR needs where the user knows the data format and how to extract information from it. SQL is less useful for more exploratory MIR tasks, which is what we focus on in this paper. Relational database indexes usually focus on each index providing fast access to the rows in one table. This is not as suitable for the design from Section 3, where we require a single index holding all objects (i.e., the entries of all cells) searcheably.

It is therefore helpful to develop an indexing solution that goes beyond just taking the union of the individual datasets. We have already collected some initial experiences in the MathDataHub system [DMH], where datasets are broken down into parts that roughly correspond to mathematical properties (group order, number of triangles in a graph, ...). Such an approach supports indexing subsets of datasets and allows for building new datasets from old ones. Even though that work predates and is in fact not always consistent with the ideas developed in this paper, some parts of it can be seen as an ad-hoc prototype solution of a concrete index.

In the sequel, we follow our design from the previous section and specify how a relational database can be used to build an index of concrete objects. Note that in this design, the entire database serves as the index, and that use of the word "index" must be distinguished from any internal indexes kept by the database implementation.

The Symbolic-Concrete Spectrum We use the following intuition to distinguish between symbolic and concrete objects: Symbolic objects include free names

(constants, variables) and thus cannot always be reduced to a value. Concrete objects, on the other hand, are closed and fully evaluated. The distinction is not as clear-cut as one might think:

- A closed expression containing bound variables is a borderline case; we consider it symbolic.
- A polynomial (with evaluated coefficients) contains the variable name; but if we consider those names to be string values (which is done in many datasets), the whole polynomial can be seen as concrete.
- Irrational numbers such as e or $\sqrt{2}$ contain names but are still generally considered to be values.

The distinction is important because it leads to differences in indexing. The MIR needs for symbolic objects focus on their structure. Symbolic objects can be stored efficiently in a substitution tree index and queried by unification queries as done in MathWebSearch. However, the more desirable querying up to inferable/computable properties is difficult, e.g., search/unification up to associative and commutative properties is a well-known difficult problem. On the other hand, many interesting properties of concrete objects are (often efficiently) computable, e.g., it is easy to check if a finite prefix of an integer sequence contains a certain subsequence. Thus, it is desirable to index concrete objects and their properties in a way that supports such queries.

To make the distinction precise, we have introduced a rigorous treatment in [WKR17]. Firstly, we standardized a set of types (numbers, strings, lists, and tuples) for concrete objects commonly used in data representation languages such as JSON or CSV. Secondly, we standardized a notation of codecs that represent symbolic objects as concrete ones. This allows treating any mathematical object as a triple of its symbolic representation, a codec, and the corresponding encoded concrete object.

An Index Design We use a relational database with one table for each type in our standardized language of concrete objects. Each table has a column "value" holding the object using a chosen standard encoding.

For each type we define a set of operations that are precomputed and stored with the objects (e.g., the factorization of an integer or the roots of a polynomial), and their results are stored in additional columns. However, these columns do not hold the actual result objects; instead, the results are concrete objects that are themselves stored in the index, and the columns just hold references to them. (A recursion threshold is used in case this process does not terminate.)

In practice, we must distinguish between different kinds of precomputed operations. Some will require so much mathematical knowledge that they can only be computed by computer algebra systems. Those computations may or may not be linkable via the database's foreign function interface. On the other end of the spectrum, some computations will be so easy that they can be carried out by the database on the fly, e.g., in a function-based SQL index.

Overall, this design has the advantage of being extensible. We can easily add new types (i.e., tables) and new precomputed operations (i.e., columns). This results in a formal language of types, constructors for objects of these types,

and operations on such objects, which we call MDDL (for mathematical data description language).

Concrete Queries A concrete query over this index is of the form `SELECT` $X_1 : T_1, \ldots, X_n : T_n$ `WHERE` $P(X_1, \ldots, X_n)$. Here the T_i are types and P is a computable MDDL-expression of boolean type. The X_i represent objects in the index of type T_i and are bound in P. The intended semantics is that it returns all substitutions to the X_i for which P is true.

It is straightforward to develop more complex query languages, but even this simple form is quite difficult to implement. Most critically, even if P is computable, it may not be efficiently computable. And even if it is, it may not be practical to program the computation inside an SQL database.

On the other hand, many simple forms of P can be directly translated to SQL queries. For example, if f is one of the precomputed values for T, then `SELECT` $X : T$ `WHERE` $f(X) = 5$ becomes the SQL query `SELECT value FROM` T `WHERE` $f = 5$.

Open Problems While we are convinced in general of the utility of this design, several open problems remain, for which further research is needed. We discuss these in the remainder.

In some cases, our design will explode. For example, storing all subsequences of an OEIS sequence may become infeasible quickly even if attention is restricted to fixed-length prefixes of sequences. Thus, special indexing techniques must be developed for individual types and operations.

Another issue is the choice of codec in the index. For each type, we can choose a standard codec and use it to represent the objects in that type's table. Then harvesters that find encoded objects in different encodings must transcode them into the standard encoding. However, in some cases this will be inefficient — the most common example is the trade-off between sparse and dense encodings of lists.

But even in the seemingly trivial case of integers, this can become an issue: For example, in [WKR17], we encountered multiple different encodings of unlimited precision integers transcoding between which was not always trivial. This is aggravated in connection with the next issue discussed below: different codecs may commute more easily with different mathematical operations. Therefore, it may be necessary to use multiple tables for the same type — one per codec. This will make retrieval harder as results from all tables have to be considered; moreover, the same object might exist in multiple tables.

Finally, if an index is hosted by a relational database, it is desirable to match mathematical operations to primitive database operations. But this is difficult because the database only sees the encoding. For example, computing the degree of a univariate polynomial encoded as a list of coefficients can easily be done by the database by taking the length of the list. But computing its roots requires decoding it, computing the roots in custom code, presumably in a computer algebra system, and then encoding the results.

5 A Heterogeneous Query Language

Overview Figure 3 shows the general search architecture we propose. On the left we have any number of libraries, which are harvested into four aspect-specific indexes as described above. A user query Q is expressed in a cross-aspect query language described below. It is passed to a query engine that separates Q into a set of aspect-specific atomic queries Q_i, for which the respective database returns result R_i. These are then aggregated into the overall result R that is returned to the user. Note that our drawing uses exactly one query Q_i per aspect — that is just an example, and there can be any number of queries (including zero) per index. It is also straightforward to extend the design with additional indexes if new kinds of indexes are conceived.

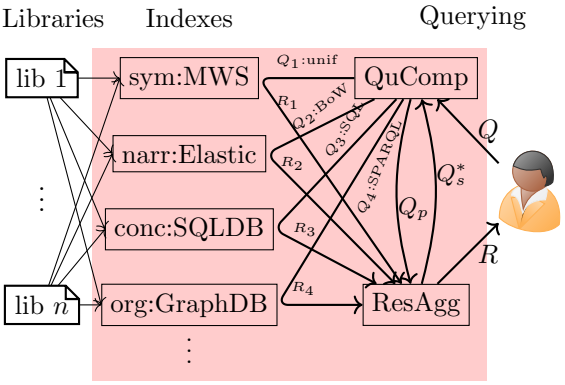

Fig. 3. The Search Architecture

In this paper, we focus on a relatively simple format for the queries: Every query Q consists of

- a list of query variables $X_1, \ldots, X_n$, we use upper case letters for them,
- a list of atomic queries $Q_i(X_1, \ldots, X_n)$.

Each atomic query is aspect-specific and resolved by lookup in the respective index. The intuition of the overall result R is to return the intersection of the atomic queries Q_i. More formally, the results R_i and R of the queries are substitutions for the query variables. The atomic queries are evaluated sequentially; each time some query variables may already have been instantiated by previous atomic queries, and the results are substitutions for the remaining ones.

More complex queries can easily be conceived, but this simple fragment captures not only the most practically relevant cases but also one of the biggest difficulties of heterogeneous queries: How can queries of different aspects meaningfully share query variables? The latter is what we discuss in the remainder.

Atomic Queries with Shared Variables To specify our query language in detail, we have to spell out the structure of the atomic queries. Here, we are mostly bound by the capabilities of the existing aspect-specific indexes except for occasionally deriving improvement suggestions for them.

All atomic queries are relative to a set of query variables ranging over formal objects. All query variables may by typed with MDDL types. The results are substitutions of the query variables with formal objects. Here the set of formal objects should be a large enough to subsume content MathML but should also allow any URI as an identifier even if it is not declared in some content dictionary

(e.g., any identifier of a paper, author, etc.) as well as sufficient literals as needed to build concrete objects.

Concretely, we assume the following:

- An **organizational atom** is an RDF triple $s\ p\ o$ possibly containing a query variable as the subject s or object o. It instantiates these with identifiers or literals.
- A **symbolic atom** is of the form $F \in \texttt{Symb}(S(X_1, \ldots, X_n))$ where S is some formal object with free query variables and F is a query variable. It substitutes F with the identifier of the fragment that contains an object matching S and substitutes the X_i according to that match.
- A **concrete atom** is as described in Section 4 except that the free variables are taken from the globally bound query variables X_i. Thus, it is simply an MDDL predicate. It substitutes the query variables with pairs of concrete object and codec.
- A **narrative atom** is of the form $F \in \texttt{Narr}(W_1, \ldots, W_m)$ where F is a query variable and each W_i is a string-valued object. The query instantiates F with the identifier that matches the bag of words containing the W_i. Due to the nature of implementations of narrative queries, the bag of words may not contain any free variables when sent to the narrative index, i.e., any W_i that are query variables must have been instantiated previously (with a string value) by some other atoms.

Both SPAQRL and MDDL queries naturally use a `SELECTWHERE` form with the `WHERE` clause containing a conjunction of atoms. This inspires our overall syntax for heterogeneous queries: `SELECT` V^* `WHERE` A^* where each V declares a query variable X as $X : T$, and each A is one of the four atoms. For convenience, we also allow undeclared query variables — these are simply dropped from the returned substitutions.

Notably, stand-alone symbolic query engines only use S as the query (rather than $F \in \texttt{Symb}(S)$) and return pairs of fragment identifiers and substitutions. Similarly, stand-alone narrative query engines usually only use the bag of words as the query. But in heterogeneous queries, we may want to use the fragment identifier in other atoms of the query. Therefore, we have extended the syntax for symbolic and narrative atoms with an explicit query variable referring to the fragment. The corresponding extension is not needed for organizational and concrete atoms.

A key difficulty is that atoms of different aspects instantiate variables with different kinds of objects, and these cannot always be directly substituted into atoms of other aspects. For example, consider a symbolic atom $F \in \texttt{Symb}(X^2)$ that substitutes X with some identifier MathML symbol s. We can still use the variable X is a subsequent narrative atom by converting it to a string, e.g., by using the name of s. But if X is substituted with a composite MathML object, we have to first evaluate it into a string, which may or may not be possible or easy. Similarly, we can still use X in a subsequent concrete atom, but only if we infer a codec that should be used to encode X into a concrete object; this codec can be inferred from the type declared for X in the query or in some cases

simply from the shape of X. Therefore, for each pair (a, b) of aspects, we need conversion rules that allow converting objects substituted by a-atoms to objects usable in b-atoms. Figure 4 gives an overview of possible conversions for column heads a and row head b.

instantiating query* instantiates with	organizational id or literal	symbolic symb. obj.	concrete conc. obj.+codec
used by ... queries via ...			
organizational	as is	ids, literals: as is other: evaluate	
symbolic	as is	as is	decode
concrete	literals: as codes ids: fail	encode^P	as is
narrative	ids: name as string literals: as string	other: evaluate	value as string

*: narrative queries never instantiate variables; ^{P} marks partial conversions

Fig. 4. Conversions of objects across queries of different aspects

Of course, if a query contains multiple atoms of the same aspects, it may be reasonable to merge them. Multiple organizational atoms can be directly joined into a SPARQL query, and similarly, multiple concrete atoms can be translated jointly into a single SQL query. However, two additional and conflicting implementations strategies must be considered: On the one hand, it is desirable to first execute those atomic queries that fill in many query variables. That makes later queries more specific and thus more efficient. On the other hand, it is desirable to first execute those atomic queries that return the fewest results. Because every result leads to a different substitution, all subsequent atomic queries using those query variables must be duplicated for each result. It remains an open question which strategy works best in practice, and it is unlikely that a single best strategy exists. But there is a large databases literature to draw experience from.

While it is, in our experience, not very common to find queries that naturally combine all four index types, combinations of two or three are quite common.

Example 1. Consider a concrete library of graphs in a table that additionally stores human-recognizable names and arc-transitivity for each graph (for example, [EET]). These are harvested into a concrete index with a type and codec for graphs, e.g., the `sparse6` format [McF], a Boolean computed property for the arc-transitivity, and a string property for the name. Additionally, consider all papers from the Cornell e-Print arXiv harvested into the same narrative index [SK08], and an organizational index that stores triples for the BIBO publication and SPAR semantic publishing ontologies.

Q1: Find arc-transitive graphs that are mentioned by name in articles in journals with h-index greater than 50.

can be encoded in the following query using the concrete, narrative, and organizational aspects:

$$\texttt{SELECT}\ G : \mathit{Graph}\ \texttt{WHERE}$$
$$\texttt{arcTransitive}(G), F \in \texttt{Narr}(\texttt{Name}(G), \text{"}\texttt{graph}\text{"}),$$
$$F\ \texttt{partOf}\ P, P\ \texttt{bibo:publishedIn}\ J, J\ \texttt{spar:hasHindex}\ H, H > 50$$

The first atom in the `WHERE`-clause returns all arc-transitive graphs G in the concrete index.

The second atom retrieves the names of these graphs and runs a narrative query for them. This includes evaluating the expression $\texttt{Name}(G)$ into a string by retrieving the corresponding value from the concrete index. To avoid false-positives, we include the word "`graph`" in the narrative atom. It instantiates F with the identifier of the matching fragment, presumably a part of a paper.

The next three atoms are organizational atoms that perform a SPARQL query retrieving first the identifier P of the paper containing F, the identifiers J of the journal it appeared in, and its h-index H. H is a concrete value that is reused in the final concrete query on the size of H.

Finally, we throw away all variables from the obtained substitutions except for the graphs G. Alternatively, we could include P in the `SELECT`-clause to also return the paper.

In the above example, we see how a query compiler should consider merging consecutive organizational atoms into a single SPARQL query. In that case, the last concrete atom of the example could, because it is so simple, alternatively and more efficiently be included in that SPARQL query as well. Moreover, the atoms in the `WHERE`-clause were ordered in a way that previous queries restrict the scope of the subsequent ones. More generally, the query compilers should reorder the atoms automatically.

6 Conclusion and Future Work

We have presented a high-level design for a cross-aspect query language and search engine for mathematical information retrieval. The crucial observation is that mathematical information needs address multiple aspects and even though mathematical libraries often have a primary aspect, they usually also contain or reference material of other aspects as well. Our cross-aspect search architecture proposes to harvest all objects into aspect-specific indexes. Correspondingly, the proposed query language combines atomic queries from existing aspect-specific query languages and a query compiler distributes them to the respective indices. The query language is more than just a sum of the four parts as it allows to share variables between the aspect-specific sub-queries and compute non-trivial joins.

We have conducted a requirement analysis on the respective basis technologies and have confirmed the principal adequacy of the query language on paradigmatic, cross-aspect information needs. This shows that existing search/indexing technologies are essentially sufficient for cross-aspect search except for the concrete aspect, where our previous work in MathDataHub provides a good first step.

The obvious next step is an implementation of a distributed cross-aspect search engine as sketched as part of the MathHub system. MathHub already has already collected most of the largest theorem prover libraries (symbolic), the 1.5M preprints of the arXiv, and several large collections of concrete mathematical objects in a common representation format and assigned uniform identifiers to their fragments. MathHub already integrates symbolic and narrative indices, and the MMT system which MathHub employs for knowledge management – while not a dedicated index – can already answer complex symbolic and organizational queries [Rab12].

References

[AD] *Apache Drill – Schema-free SQL Query Engine for Hadoop, NoSQL and Cloud Storage*. URL: https://drill.apache.org (visited on 03/02/2020).

[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: https://tinyurl.com/sofcxjs.

[BKR20] Katja Berčič, Michael Kohlhase, and Florian Rabe. *Towards a Heterogeneous Query Language for Mathematical Knowledge – Extended Report*. 2020. URL: https://kwarc.info/kohlhase/papers/tetrasearch.pdf (visited on 03/27/2020).

[Car+20a] Jacques Carette et al. "Big Math and the One-Brain Barrier – The Tetrapod Model of Mathematical Knowledge". In: *Mathematical Intelligencer* (2020). in press. URL: https://arxiv.org/abs/1904.10405.

[Car+20b] Jacques Carette et al. "The Space of Mathematical Software Systems – A Survey of Paradigmatic Systems". preprint; http://arxiv.org/abs/2002.04955. 2020.

[Cho+05] Eugene Inseok Chong et al. "An Efficient SQL-based RDF Querying Scheme". In: *Proceedings of the 31. VLDB Conference*. 2005.

[DHI12] AnHai Doan, Alon Halevy, and Zachary Ives. *Principles of Data Integration*. Elsevier, 2012.

[DMH] *Datasets on MathHub.info*. URL: https://data.mathhub.info (visited on 09/24/2019).

[EBO] Bettina Eick, Hans Ulrich Besche, and Eamonn O'Brien. *SmallGrp – The GAP Small Groups Library*. URL: https://www.gap-system.org/Manuals/pkg/SmallGrp-1.3/doc/chap1.html (visited on 10/13/2018).

[EET] Steve Wilson and Primož Potočnik. *A Census of edge-transitive tetravalent graphs*. URL: https://jan.ucc.nau.edu/~swilson/C4FullSite/index.html (visited on 01/23/2019).

[Eso] *Elastic Search*. Feb. 20, 2014. URL: http://www.elasticsearch.org/ (visited on 02/20/2014).

[FL] Jukka Kohonen. *Lists of finite lattices (modular, semimodular, graded and geometric)*. URL: https://www.shsu.edu/mem037/Lattices.html (visited on 01/25/2019).

[GSC15] Ferruccio Guidi and Claudio Sacerdoti Coen. "A Survey on Retrieval of Mathematical Knowledge". In: *Intelligent Computer Mathematics 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.

[HKP14] Radu Hambasan, Michael Kohlhase, and Corneliu Prodescu. "MathWebSearch at NTCIR-11". In: *NTCIR 11 Conference*. Ed. by Noriko Kando, Hideo Joho, and Kazuaki Kishida. Tokyo, Japan: NII, Tokyo, 2014, pp. 114–119. URL: https://tinyurl.com/wzj7mcg.

[IKP14] Mihnea Iancu, Michael Kohlhase, and Corneliu-Claudiu Prodescu. "Representing, Archiving, and Searching the Space of Mathematical Knowledge". In: *Mathematical Software - ICMS 2014 - 4th International Congress*. Ed. by Hoon Hong and Chee Yap. Vol. 8592. LNCS. Springer, 2014, pp. 26–30. ISBN: 978-3-662-44198-5. DOI: 10.1007/978-3-662-44199-2_5.

[Lmf] *The L-functions and Modular Forms Database*. http://www.lmfdb.org. [Online; accessed 27 August 2016].

[McF] Brendan McKay. *Description of graph6, sparse6 and digraph6 encodings*. URL: http://users.cecs.anu.edu.au/~bdm/data/formats.txt (visited on 03/22/2019).

[MDT] Katja Berčič. *Math Databases Table*. URL: https://mathdb.mathhub.info/ (visited on 01/15/2020).

[OEIS] *The On-Line Encyclopedia of Integer Sequences*. URL: http://oeis.org (visited on 05/28/2017).

[Rab12] Florian Rabe. "A Query Language for Formal Mathematical Libraries". In: *Intelligent Computer Mathematics*. Ed. by Johan Jeuring et al. LNAI 7362. Berlin and Heidelberg: Springer Verlag, 2012, pp. 142–157. ISBN: 978-3-642-31373-8. arXiv: 1204.4685 [cs.LO].

[SK08] Heinrich Stamerjohanns and Michael Kohlhase. "Transforming the arXiv to XML". In: *Intelligent Computer Mathematics*. Ed. by Serge Autexier et al. LNAI 5144. Springer Verlag, 2008, pp. 574–582. URL: https://kwarc.info/kohlhase/papers/mkm08-arXMLiv.pdf.

[Slo03] Neil J. A. Sloane. "The On-Line Encyclopedia of Integer Sequences". In: *Notices of the AMS* 50.8 (2003), p. 912.

[ST16] Yiannos Stathopoulos and Simone Teufel. "Mathematical Information Retrieval based on Type Embeddings and Query Expansion". In: *Proceedings of COLING 2016*. ACL, 2016, pp. 2344–2355. URL: https://www.aclweb.org/anthology/C16-1221.

[Sta+18] Yiannos Stathopoulos et al. "Variable Typing: Assigning Meaning to Variables in Mathematical Text". In: *NAACL 2018 Proceedings*. ACL, 2018, pp. 303–312. DOI: 10.18653/v1/N18-1028.

[WKR17] T. Wiesing, M. Kohlhase, and F. Rabe. "Virtual Theories – A Uniform Interface to Mathematical Knowledge Bases". In: *Mathematical Aspects of Computer and Information Sciences*. Ed. by J. Blömer et al. Springer, 2017, pp. 243–257.