Virtual Theories – A Uniform Interface to Mathematical Knowledge Bases

Tom Wiesing[1] Michael Kohlhase[1] Florian Rabe[2]

[1] FAU Erlangen-Nürnberg
[2] Jacobs University Bremen

Abstract. To support mathematical research, engineering, and education by computer systems, we need to deal with the differences between mathematical content collections and information systems available today. Unfortunately, these systems – ranging from Wikipedia to theorem prover libraries are usually only accessible via a dedicated web information system or a low-level API at the level of the raw database content. What we would want is a "programmatic, mathematical API" which would give access to the knowledge-bases programmatically via their mathematical constructions and properties.
This paper takes a step into this direction by interpreting large knowledge bases as OMDoc/MMT theories – modular representations of mathematical objects and their properties. For this, we generalize OMDoc/MMT theories to "virtual theories" – theories so big that they do not fit into main memory – and update its knowledge management algorithms so that they can work directly with objects stored in external knowledge bases. An additional technical contribution is the introduction of a codec system that bridges between low-level encodings in databases and the abstract construction of mathematical objects.

1 Introduction

There are various large-scale sources of mathematical knowledge. These include
- generic information systems like Wikipedia,
- collections of informal but rigorous mathematical documents – e.g. research libraries, publisher's "digital libraries", or the Cornell preprint arXiv,
- literature information systems like zbMATH or MathSciNet,
- databases of mathematical objects – like the GAP group libraries, the Online Encyclopedia of Integer sequences (OEIS [**Sloane:OEIS**; Inc]), and the L-Functions and Modular Forms Database (LMFDB [Cre16; LMFDB]),
- fully formal theorem prover libraries like those of Mizar, Coq, PVS, and the HOL systems.

We will use the term **mathematical knowledge bases** to refer to them collectively and restrict ourselves to those that are available digitally. They are very useful in mathematical research, applications, and education. Commonly these systems are only accessible via a dedicated web interface that allows humans to query or browse the databases. A programmatic interface, if it exists at

all, is usually system specific, to use it, users need to be familiar both with the mathematical background and internal structure of the system in question. No predominant standard exists, and these interfaces usually only expose the low-level raw database content. We claim that mathematicians and other scientists desire a "programmatic, mathematical API" that gives access to the knowledge-bases programmatically via their mathematical constructions and properties. We focus on addressing this problem in this paper.

For our implementation we interpret mathematical knowledge bases as OMDoc/MMT theory graphs – modular, flexi-formal representations of mathematical objects, their properties, and relations. This embedding gives us a common conceptual framework to handle different knowledge sources, and the modular and heterogeneous nature of OMDoc/MMT theory graph can be used to reconcile differing ontological commitments of the knowledge sources with in this conceptual framework.

To cope with the scale of common mathematical knowledge bases we generalize OMDoc/MMT theories to "virtual theories", which allow for unlimited, dynamically growing number of declarations. We also update the knowledge management algorithms in the MMT system so that they can directly deal with the databases underlying the knowledge bases. Here we provide a systematic solution for encoding/decoding between low-level representations in standard databases and high-level mathematical representations.

This paper proceeds as follows: In Section 2 we give a short overview of OMDoc/MMT theory graphs along with the Math-In-The-Middle approach developed in the OpenDreamKit project, our primary use-case for virtual theories. Section 3 discusses LMFDB and its interface as an example of a very large state-of-the-art mathematical knowledge base, and Section 4 shows how it can be represented as a set of virtual theories. Section 5 introduces the codec architecture and describes how to access virtual theories at the semantic/mathematical level, and Section 6 makes QMT queries aware of virtual theories. Section 7 concludes the paper.

2 Virtual Research Environments for Mathematics: the Math-in-the-Middle Approach

The work reported in this paper originates from in the EU-funded OpenDreamKit [ODK] project that aims to create virtual research environments (VRE) enabling mathematicians to make efficient use of existing open-source mathematical knowledge systems. These systems include computer algebra systems like SageMath and GAP as well as mathematical data bases such as the LMFDB, which must be made interoperable for integration into a VRE. In the OpenDreamKit project we have developed the Math-in-the-Middle (MitM) approach, which posits a central ontology of mathematical knowledge, which acts as a pivot point for interoperability; see [Deh+16] for a description of the approach and [Koh+] for a technical refinement and large-scale interoperability case study.

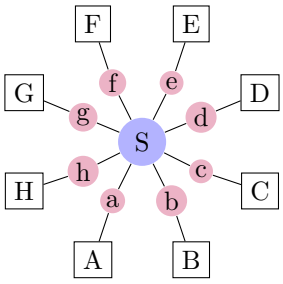

Fig. 1. The MiTM Approach to Connecting Systems.

The MitM ontology in the center of Figure 1 models the true, underlying mathematical semantics in OMDoc/MMT and allows translation between this centrally formalized knowledge and the systems on the boundary via views and alignments. This mathematical knowledge is modeled using the well-established theory graph paradigm and is stored inside our OMDoc/MMT-based MathHub system [MH].

The knowledge in the mathematical software systems – denoted by square boxes in Figure 1 – also modeled via OMDoc/MMT theory graphs the **API theories** – the corresponding red circles; these are generated from the knowledge bases in the systems by a custom process. The API theories allow us to implement translation with the help of OMDoc/MMT views and alignments between the ontology – the Math-In-The-Middle – and each of the systems and use these translations for transporting computational tasks between the systems.

The realization of the MitM approach crucially depends on the information architecture of the OMDoc/MMT language [Koh06; RK13] and its implementation in the MMT system [Rab13; MMT].

In OMDoc/MMT knowledge is organized in **theories**, which contain information about mathematical concepts and objects in the form of **declarations**. Theories are organized into an "object-oriented" inheritance structure via **inclusions** and **structures** (for controlled multiple inheritance), which is augmented via truth-preserving mappings between theories called **views**, which allow to relate concepts of pre-existing theories and transport theorems between these. Inclusions, structures, and views impose a graph structure on the represented mathematical knowledge, called a **theory graph**.

We observe that even very large mathematical knowledge spaces about abstract mathematical domains can be represented by small, but densely connected, theory graphs, if we make all inherited material explicit in a process called **flattening**. The OMDoc/MMT language provides systematic names (MMT URIs) for all objects, properties, and relations in the induced knowledge space, and given the represented theory graph, the MMT system can compute them on demand.

Generally, knowledge in a knowledge space given by a theory graph loaded by the MMT system can be accessed by either giving it's MMT URI, or by uniquely describing it via a set of conditions. To achieve the latter, MMT has a Query Language called QMT [Rab12], which allows even complex conditions to be specified. Currently, the MMT system loads the theory graph into main memory at startup and interleaves incremental flattening and query evaluation operations on the MMT data structures until the result has been produced.

In [Koh+] we show that the MitM approach, its OMDoc/MMT-based realization, and distribution via the SCSCP protocol are sufficient for distributed, federated computation between multiple computer algebra systems (Sage, GAP,

and Singular), and that the MitM ontology of abstract group theory can be represented in OMDoc/MMT efficiently. This setup is effective because

- the knowledge spaces behind abstract and computational mathematics can be represented in theory graphs very space-efficiently: The compression factors between a knowledge space and its theory graph – we call it the **TG factor** – exceeds two orders of magnitude even for small domains.
- only small parts of the knowledge space are traversed for a given computation.

But the OpenDreamKit VRE must also include mathematical data sources like the LMFDB or the OEIS, which contain millions of mathematical objects. For such knowledge sources, the classical MMT system is not yet suitable:

- the knowledge space corresponding to the data base content cannot be compressed by "general mathematical principles" like inheritance. Indeed, redundant information is already largely eliminated by the data base schema and the "business logic" of the information system it feeds.
- typically large parts of the knowledge space need to be traversed to obtain the intended results to queries.

Therefore, we extend the concept of OMDoc/MMT theories – which carry the implicit assumption of containing only a small number of declarations (see [FGT92] for a discussion) – to **virtual theories**, which can have an unlimited (possibly infinite) number of declarations. To contrast the intended uses we will call the classical OMDoc/MMT theories **concrete theories**. In practice, a virtual theory is represented by concrete approximations: OMDoc/MMT works with a concrete theory, whose size changes dynamically as a suitable backend infrastructure generates declarations on demand.

3 Example: The API and Structure of LMFDB

The "L-Functions and Modular Forms Database" (LMFDB [LMFa]) is a large database, storing among other mathematical objects several thousand L-Functions and curves along with their properties. Technically, it uses a MongoDB database with a Python web frontend. We use this as an example of a virtual theory. Before we go into this in more detail, we have a closer look at the structure and existing APIs to of LMFDB.

3.1 The Structure of LMFDB

LMFDB has several sub-databases, e.g., for elliptic curves or transitive groups. Within each of these, every object is stored as a single JSON record. Figure 2 shows an example: each property of this JSON object corresponds to a property of the underlying mathematical object. For example, the `degree` property – here 1 – of the JSON objects corresponds to the degree of the underlying elliptic curve.

Other properties are more complex: the value of the `isogeny_matrix` property is a list of lists representing a matrix. This disconnect between JSON

```
{
    "degree": 1,
    "x-coordinates_of_integral_points": "[5,16]",
    "isogeny_matrix": [[1,5,25],[5,1,5],[25,5,1]],
    "label": "11a1",
    "_id": "ObjectId('4f71d4304d47869291435e6e')",
    ...
}
```


Fig. 2. Part of an elliptic curve in LMFDB (some fields omitted for brevity)

encoding and mathematical meaning can become much more severe, e.g., the `x-coordinates_of_integral_points` field is semantically a list of integers but (due to the sizes limits on integers) is encoded as a string.

3.2 An API for LMFDB Objects

Querying is an important application for mathematical knowledge bases. The LMFDB API [Lmf] exposes a querying interface that can be used either by humans via the web or programmatically via JSON-based GET requests over HTTP. A screenshot of the former interface can be seen in Figure 3.

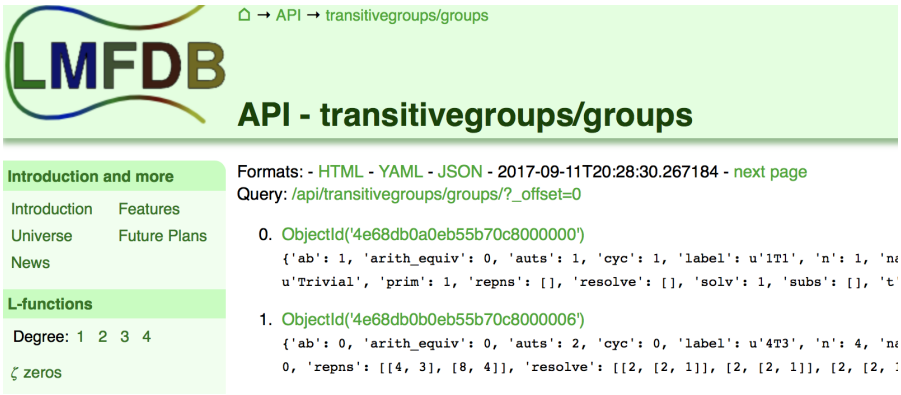

Fig. 3. The Web-Interface for the LMFDB API.

Queries must name the sub-database to be queried and consist of a set of key-value pairs that correspond to an SQL `where` clause. However, while LMFDB offers a programmable API for accessing its contents, this API sits at the level of the underlying MongoDB, and not the level of mathematical objects. For example, to retrieve all Abelian objects in the subdatabase of transitive groups, we expect to use the key-value pair `commutative= true`. However, these values

need to be encoded to be understood by MongoDB. We need to realize that the database schema actually uses the key `ab` for commutativity, that it has boolean values, and that the schema encodes `true` as `1`. Thus, the actual query to send is `http://www.lmfdb.org/api/transitivegroups/groups/?ab=1`.

In this example, all steps are relatively straightforward. But in general, e.g. when searching for all elliptic curves with a specific isogeny matrix, this not only requires good familiarity with the mathematical background but also with the system internals of the particular LMFDB sub-database; a skill set commonly found in neither research programmers nor average mathematicians.

Our diagnosis is that LMFDB – and most other mathematical knowledge databases – suffer from two problems:

- *human/computer mismatch*: humans have problems interacting with LMFDB programmatically, because they must speak the system language instead of mathematical language
- *computer/computer mismatch*: mathematical computer systems cannot interoperate with LMFDB without extending their code, because their system languages differ.

Using the MitM approach we have presented in Section 2, we can solve both problems at the same time by lifting the communication to the level of OMDoc/MMT-encoded MitM objects, which both MitM-compatible software systems and humans can understand.

4 LMFDB as a Set of Virtual Theories

The mathematical software systems to be integrated via the MitM approach have so far been computation-oriented, e.g., computer algebra systems. Their API theories typically declare types and functions on these types (the latter including constants seen as nullary functions). Even though database systems differ drastically from these in many respects, they are very similar at the MitM level: a database like LMFDB defines

- some types: each table's schema is essentially one type definition,
- many constants: each table entry is one constant of the corresponding type.

Thus, we can apply essentially the same approach. In particular, the API theories must contain definitions of the database schemas.

From a system perspective, virtual theories behave just like concrete theories, but without the assumption of being able to load all declarations from some file on disk at once. Instead, virtual theories load declarations in a lazy fashion when they are needed. MMT stores concrete theories as XML files. Because most external knowledge bases use databases with low-level APIs, we must allow virtual theories to be stored in external database. Apart from standard software engineering tasks, this leaves three conceptual problems we had to solve:

P1 Turn the database schemas and tables into OMDoc/MMT theories and declarations.

P2 Lift data in *physical* representation (as records of the underlying database) to OMDoc/MMT object in *semantic* representation.

P3 Translate semantic queries to queries about physical representations so that they can be executed directly on the database without loading the entire theory into MMT.

We deal with **P1** here, with **P2** in Section 5, and with **P3** in Section 6.

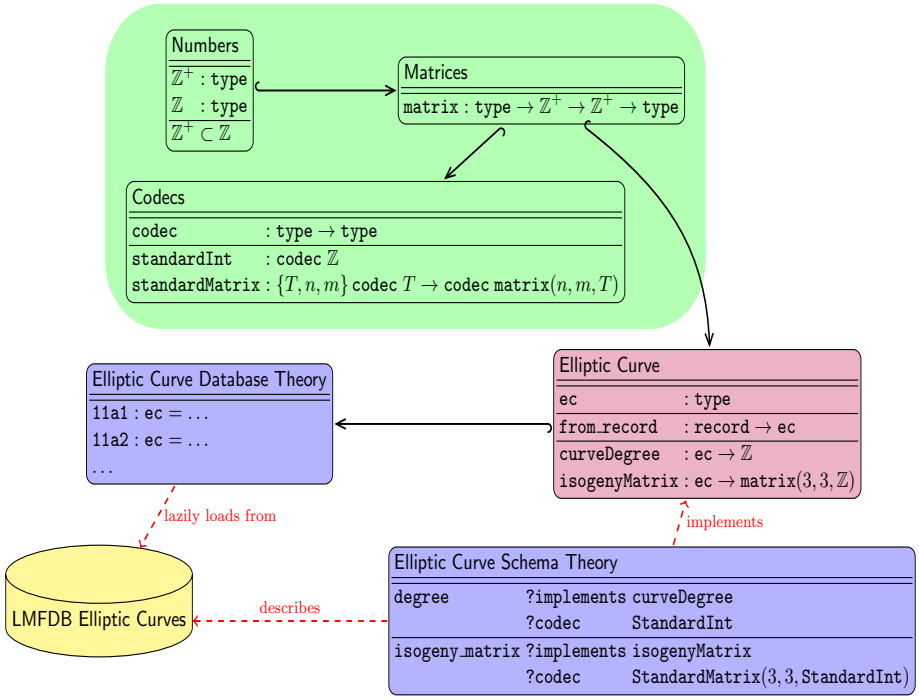

Fig. 4. Virtual theory for LMFDB elliptic curves (some declarations omitted)

A sketch of our overall solution is given in Figure 4. The math in the middle comprises preexisting formalizations of general mathematics, here numbers and matrices (in green), and novel LMFDB-specific ones, here elliptic curves (in red). Moreover, we introduce a specification of various codecs to translate between physical and semantic representations. The remaining theories (in blue) form the LMFDB API theories: the schema theory and the database theory, which we describe below.

The set of constants in a database table – while finite – can be arbitrarily large. In particular, all LMFDB tables[1] are just finite subsets of infinite sets, whose size is not limited by mathematical specifications but by computational

[1] Technically, LMFDB is implemented using MongoDB and comprises a set of sets (each one called a database) of JSON objects. However, due to the conventions used, we can also understand it conceptually as a set of tables of a relational database, keeping in mind that every row is a tuple of arbitrary JSON objects.

power: the database holds all objects that users have computed so far and grows constantly as more objects are computed. LMFDB tables usually include a naming system that defines unique identifiers (which are used as the database keys) for these objects, and these identifiers are predetermined even for those objects that have not been computed yet. Thus, it is not practical to fix a set of concrete API theories. Instead, the API theories must be split into two parts: for each database table, we need

- a concrete theory called the **schema theory** that defines the schema and other relevant information about the type of objects in the table and
- a virtual theory called the **database theory** that contains one definition for each value of that type (using the LMFDB identifier as the name of the defined constant).

LMFDB's technical realization does not require formalizing the schema of each table. Instead, the tables are generated systematically and therefore follow an implicit schema that can – in principle – be obtained from the documentation or reverse-engineered from the tables. However (and here LMFDB critically differs from, e.g. the OEIS), the mathematics involved in the tables is so deep that this is not possible in practice for all but a few experts. Therefore, we sat down with the original author of one of the best-documented tables – John Cremona for the table of elliptic curves – and formalized the corresponding schema in OMDoc/MMT.

In the following, we will use this table as a running example. Our methods extend immediately to any other table once its schema has been formalized.

Our formalization models elliptic curves in a very simple fashion by using an abstract type `ec`. The constructor `from_record` takes an MMT record and returns an elliptic curve. Properties of elliptic curves are formalized as functions out of this type. We list only two here as examples: the degree, an integer, and the isogeny matrix, a 3×3 matrix of integers. We omit the relevant axioms, which are not essential for our purposes here. Recall that the Math-in-the-Middle approach models mathematical knowledge "in the middle" independent of any particular system. This is exactly the case here – the model of elliptic curves does not rely on LMFDB, nor any other system, so that we can integrate other knowledge sources about elliptic curves or to future versions of the LMFDB with changed structure.

5 Accessing Virtual Theories

We now address **P2**: lifting physical to semantic representations. Intuitively, it is straightforward how to implement a virtual theory V: we use an initially empty concrete theory C, and whenever an identifier id of V is requested, MMT dynamically adds the corresponding declaration of id to C. MMT already abstracts from the physical realizations of persistent storage using the *backend* interface: essentially a backend is any component that allows loading declarations. Thus, we only have to implement a new backend that connects to LMFDB, retrieves the JSON object with identifier id, and turns it into an OMDoc/MMT declaration.

However, this glosses over a major problem: the databases used for the scalable physical storage of large datasets usually offer only very simple data structures. For example, a JSON database (as underlies LMFDB) offers only limited-precision integers, boolean, strings, lists, and records as primitive objects and does not provide a type system. Consequently, the objects stored in the database are very different from the sophisticated mathematical objects expected by the schema theory. Therefore, databases like LMFDB must encode this complex mathematical objects as simple database objects.

5.1 Concrete Encodings of Mathematical Objects

Consider, for example, the field `degree` from Figure 2 above. Its *semantic* type in the MitM-formalization is $\mathbb{Z}$. However, its *physical* type in LMFDB is `IEEE754` a mixture of 64-bit floating point numbers and strings: integers that exceeds $2^{53}-1$ are stored as JSON strings containing the corresponding decimal representation. We speak of *encoding* when translating semantic objects to their physical representations and of *decoding* in the dual case, and we speak of *codecs* when referring to a pair of an encoding and a decoding function.

To formally specify codecs, we introduce a new OMDoc/MMT theory `Codecs` as a part of the MitM ontology. Our codecs are indexed by semantic types: the type constructor `codec` maps a semantic type to a new type of codecs for it. For instance, the object `StandardInt` of type `codec` $\mathbb{Z}$ is a codec that translates between LMFDB's idiosyncratic float/string-representation and MitM's integers. Note that there can be multiple different codecs for the same semantic type. For example, `IntAsArray` encodes integers x as lists of 64-bit integers consisting of the digits of x with respect to base 2^{64}.

Codecs

`codec`	: `type` → `type`	
`StandardPos`	: `codec` $\mathbb{Z}^+$	JSON number if small enough, else JSON string of decimal expansion
`StandardNat`	: `codec` $\mathbb{N}$	
`StandardInt`	: `codec` $\mathbb{Z}$	
`IntAsArray`	: `codec` $\mathbb{Z}$	JSON List of Numbers
`IntAsString`	: `codec` $\mathbb{Z}$	JSON String of decimal expansion
`StandardBool`	: `codec` $\mathbb{B}$	JSON Booleans
`BoolAsInt`	: `codec` $\mathbb{B}$	JSON Numbers 0 or 1
`StandardString`	: `codec` $\mathbb{S}$	JSON Strings

Fig. 5. Codecs specified in MMT ($\mathbb{N}$, $\mathbb{Z}$, $\mathbb{Z}^+$ are as usual, $\mathbb{B}$ are booleans, and $\mathbb{S}$ are Unicode strings)

We do not (and do not have to) define the actual encoding/decoding functions in OMDoc/MMT. It is more important to identify the codecs needed in practice, introduce names for them, and spell out their semantics. Then it is

straightforward to implement them in any other programming language used interfacing with LMFDB.

In particular, we have implemented them in Scala, the language underlying the MMT system. Additionally, the Codecs theory annotates each codec declaration with a reference to the Scala class implementing the codec. That way, MMT can run the encoding/decoding functions of the codec.

The above is only sufficient for atomic semantic types, which typically correspond to one (or more) atomic codecs. Consider now the field `isogeny_matrix` of elliptic curves. The semantic representation of one possible value (namely for the curve `11a1`) of this field is the matrix on the right.

$$M = \begin{pmatrix} 1 & 5 & 25 \\ 5 & 1 & 5 \\ 25 & 5 & 1 \end{pmatrix}$$

The semantic type operator Matrix takes 1 type argument (the element type, integers in this case) and two value arguments (the dimensions, 3 and 3 in this case) and constructs the respective matrix type. In principle, one could give a codec for each matrix type that comes up in a database schema. But a much more elegant solution is to specify **codec operator**s in analogy to type operators. A codec operator for a type operator with k type and l value arguments, takes k codec and l value arguments. For example, a codec operator for matrices takes a codec C : `codec` E for the element type E and the dimensions m and n and returns a codec of type `codec` (Matrix $E\,m\,n$).

Codecs (continued)

`StandardList` : $\{T\}$ `codec` $T \to$ `codec` List(T)	JSON list, recursively coding each element of the list
`StandardVector` : $\{T, n\}$ `codec` $T \to$ `codec` Vector(n, T)	JSON list of fixed length n
`StandardMatrix` : $\{T, n, m\}$ `codec` $T \to$ `codec` Matrix(n, m, T)	JSON list of n lists of length m

Fig. 6. Second annotated subset of the codecs theory containing a selection of codec operators found in MMT. Compare with Figure 5.

Like codecs, codec operators are represented in MMT in two ways: as declarations inside the theory `Codecs` (see Figure 6 for a list, compare again with Figure 4) and as a corresponding Scala function that maps codecs to codecs. When reading the declarations, note that we make use of the dependent function types of the MitM foundation: curly brackets denote dependent function arguments, i.e., arguments that may occur in later argument types and the result type.

With these declarations, we recover the LMFDB encoding of isogeny matrices by applying the codec operator `StandardMatrix`, which encodes matrices as lists of lists, to the codec `StandardInt` and the dimension 3 and 3. The result-

ing codec `StandardMatrix`$(\mathbb{Z}, 3, 3,$ `StandardInt`$)$ encodes the above matrix as [[1.0,5.0,25.0],[5.0,1.0,5.0],[25.0,5.0,1.0]].

5.2 Choosing Encodings in Schema Theories

If we ignore encoding issues, schema theories are straightforward: they contain one declaration of the same name for each field within an LMFDB record. This specifies only the semantic type of each field and does not relate it to the MitM formalization. To handle the encoding as a physical type, we annotate each declaration with the codec that the databases for the values of that field. Moreover, to connect the schema theories to the MitM formalization, we additionally annotate each field with the corresponding property of elliptic curves from the MitM theory. We can now understand the last unexplained parts of Fig. 4. `?implements` is the symbol used to annotate the metadatum, which MitM property a schema field corresponds to. And `?codec` similarly annotates the codec to each field.

For example, the `degree` field implements the `curveDegree` property in the elliptic curve theory and uses the `StandardInt` codec. Thus, the schema theories determine the entire relation between semantic and physical objects.

The database theory is a virtual theory and contains one declaration per LMFDB record. Given the URI of an object in the respective database, our MMT backend for LMFDB first retrieves the appropriate record from LMFDB – in the case of `11a1` this corresponds to retrieving the JSON found in Figure 2. Then, for each field, it uses the annotated codec (which is an OMDoc/MMT expression) to build an actual codec (as a runnable Scala function) and runs its decoding function. Next, it passes the resulting record to the `from_record` constructor, which yields an elliptic curve in the MitM theories. Finally, this elliptic curve is added as a new declaration in the database theory.

6 Translating Queries

Recall that MMT has a general-purpose Query Language called QMT [Rab12], which allows users to find knowledge subject to even complex conditions. We continue by briefly addressing **P3**: query translation; for a complete discussion we refer the interested reader to [Wie17].

In practice, most queries involving virtual theories so far have a shape similar to the one that LMFDB supports: Finding all objects within a single sub-database for which a specific field has a specific value. As an example, consider again the query of finding all Abelian transitive groups. QMT has an MMT-powered surface syntax, which can be used to express this query as:

```
x in (related to ( literal 'lmfdb:db/transitivegroups?group ) by (object declares))
  | holds x (x commutative x *=* true)
```

The example consists of two parts, first we find all objects declared in the `lmfdb:db/transitivegroups?group` theory (line 1), and then we restrict this set of results to all those for which the `commutative` property is `true` (line 2).

Notice that this the example shown here is the formal equivalent of the LMFDB query shown in Section 3.2. The key difference is that this query does not require knowing the record structure of LMFDB– apart from knowing the proper sub-db, instead it only relies on knowing the mathematical semantics (commutativity) of the query in question.

Recall that to evaluate a query prior to the introduction of virtual theories, the MMT system loaded the theory graph into main memory and then interleaved incremental flattening and query evaluation operations on the MMT data structures until a result had been produced. But it is infeasible to first load all potentially relevant data into memory, and only then proceed with evaluation. This would require loading a copy of LMFDB into main memory, something that virtual theories were designed to avoid.

The low-level API of LMFDB and similar system provides a new approach for making queries towards virtual theories. First, the MMT query is translated into a system-specific information-retrieval language – in the case of LMFDB this is a MongoDB-based syntax. Next, this translated query is sent to the external API. Upon receiving the results, these are translated back into OMDoc/MMT with the help of already existing functionality in the appropriate virtual theory backend.

This leaves just one problem unsolved – translating queries into the system-specific API. However, it is insufficient to simply translate queries as a whole: One hand a general QMT query may or may not involve a virtual theory, on the other hand, it may also involve several (unrelated) virtual theories. This makes it necessary to filter out queries involving virtual theories, and assign them to a specific backend, and then translate only these parts.

Achieving this automatically is a non-trivial problem. Queries are inductive in nature, and one could attempt to intercept each of the intermediate results. However, this would require a check on each intermediate result to first determine if it comes from a virtual theory or not, and then potentially switching the entire evaluation strategy, leading to a computationally expensive implementation.

Instead of intercepting each result, we extended the Query Language to allows users to annotate sub-queries for evaluation with a specific virtual theory backend. This allows the system to immediately know which parts of a query have to be evaluated in MMT memory, and which have to be translated and sent to an external system. This turns the example above into:

```
use "lmfdb" for {*
  x in (related to ( literal ‘lmfdb:db/transitivegroups?group )
    by (object declares)) | holds x (x commutative x *=* true)
*}
```

Here, we have simply wrapped the entire query with a `use lmfdb` statement, indicating the query should be evaluated using LMFDB.

The encoding of this specific query can be achieved using codecs – in fact we have already seen above in Section 3.2 how this is achieved. The query corresponds to the URL `http://www.lmfdb.org/api/transitivegroups/groups/?ab=1`. Next, the LMFDB API returns a set of JSON objects corresponding to

all Abelian transitive groups. These can then be decoded into OMDoc/MMT objects using the procedure described in Section 5.2, i.e. for each field we look up the corresponding codec and use it to deconstruct the field, eventually creating an MMT record. Afterwards, these OMDoc/MMT terms can then be passed to the user as a result to the query.

7 Conclusion

We have shown how to extend the Math-in-the-Middle framework for integrating systems to mathematical data bases like the LMFDB. The main idea is to embed knowledge sources as virtual theories, i.e. theories that are not – theoretically or in practice – limited in the number of declarations and allow dynamic loading and processing. For accessing real-world knowledge sources, we have developed the notion of codecs and integrated them into the MitM ontology framework. These codecs (and their MitM types) lift knowledge source access to the MitM level and thus enable object-level interoperability and allow humans (mathematicians) access using the concepts they are familiar with. Finally, we have shown a prototypical query translation facility that allows to delegate some of the processing to the underlying knowledge source and thus avoid thrashing of virtual theories.

Related Work Most other integration schemes employ a **homogenous approach**, where there is a master system and all data is converted into that system. A paradigmatic example of this is the Wolfram Language [**WolframLanguage:wikipedia**] and the Wolfram Alpha search engine [Wol], which are based on the Mathematica kernel. This is very flexible for anyone owning a Mathematica license and experienced in the Mathematica language and environment.

The MitM-based approach to interoperability of data sources and systems proposed in this paper is inherently a **heterogeneous approach**: systems and data sources are kept "as is", but their APIs are documented in a machine-actionable way that can be utilized for remote procedure calls, content format mediation, and service discovery. As a consequence, interaction between systems is very flexible. For the data source integration via virtual theories presented in this paper this is important. For instance, we can just make an extension of MMT or Sage which just act as a programmatic interface for e.g. LMFDB.

Future Work We have discussed the MitM+virtual theories methodology on the elliptic curves sub-base of the LMFDB, which we have fully integrated. We are currently working on additional LMFDB sub-bases. The main problem to be solved is to elicit the information for the respective schema theories from the LMFDB community. Once that is accomplished, specifying them in the format discussed in this paper and writing the respective codecs is straightforward.

Moreover, we are working on integrating the the Online Encyclopedia of Integer Sequences (OEIS [**Sloane:OEIS**; Inc]). Here we have a different problem: the OEIS database is essentially a flat ASCII file with different slots (for initial

segments of the sequences, references, comments, and formulae); all minimally marked up ASCII art. In [LK16] we have already (heuristically) flexiformalized OEIS contents in OMDoc/MMT; the next step will be to come up with codecs based on this basis and develop schema theories for OEIS.

Acknowledgements The authors gratefully acknowledge the fruitful discussions with other participants of work package WP6, in particular John Cremona on the LMFDB and Dennis Müller on early versions of the OMDoc/MMT-based integration. We acknowledge financial support from the OpenDreamKit Horizon 2020 European Research Infrastructures project (#676541).

References

[Cre16] John Cremona. "The L-Functions and Modular Forms Database Project". In: *Foundations of Computational Mathematics* 16.6 (2016), pp. 1541–1553. DOI: 10.1007/s10208-016-9306-z.

[Deh+16] Paul-Olivier Dehaye et al. "Interoperability in the OpenDreamKit Project: The Math-in-the-Middle Approach". In: *Intelligent Computer Mathematics 2016*. Ed. by Michael Kohlhase et al. LNAI 9791. Springer, 2016. URL: https://github.com/OpenDreamKit/OpenDreamKit/blob/master/WP6/CICM2016/published.pdf.

[FGT92] William M. Farmer, Josuah Guttman, and Xavier Thayer. "Little Theories". In: *Proceedings of the 11th Conference on Automated Deduction*. Ed. by D. Kapur. LNCS 607. Saratoga Springs, NY, USA: Springer Verlag, 1992, pp. 467–581.

[Inc] OEIS Foundation Inc., ed. *The On-Line Encyclopedia of Integer Sequences*. URL: http://oeis.org (visited on 05/28/2013).

[Koh+] Michael Kohlhase et al. "REGULAR-T1: Knowledge-Based Interoperability for Mathematical Software Systems". submitted to MACIS-2017. URL: https://github.com/OpenDreamKit/OpenDreamKit/blob/master/WP6/MACIS17-interop/submit.pdf.

[Koh06] Michael Kohlhase. *OMDoc – An open markup format for mathematical documents [Version 1.2]*. LNAI 4180. Springer Verlag, Aug. 2006. URL: http://omdoc.org/pubs/omdoc1.2.pdf.

[LK16] Enxhell Luzhnica and Michael Kohlhase. "Formula Semantification and Automated Relation Finding in the OEIS". In: *Mathematical Software - ICMS 2016 - 5th International Congress*. Ed. by Gert-Martin Greuel et al. Vol. 9725. LNCS. Springer, 2016. DOI: 10.1007/978-3-319-42432-3.

[Lmf] *LMFDB - API*. http://www.lmfdb.org/api/. visited on: 09/17/2017.

[LMFDB] The LMFDB Collaboration. *The L-functions and Modular Forms Database*. URL: http://www.lmfdb.org (visited on 02/01/2016).

[MH] *MathHub.info: Active Mathematics*. URL: http://mathhub.info (visited on 01/28/2014).

[MMT] *MMT – Language and System for the Uniform Representation of Knowledge*. project web site. URL: https://uniformal.github.io/ (visited on 08/30/2016).

[ODK] *OpenDreamKit Open Digital Research Environment Toolkit for the Advancement of Mathematics*. URL: http://opendreamkit.org (visited on 05/21/2015).

[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. arXiv: 1204.4685 [cs.LO].

[Rab13] Florian Rabe. "The MMT API: A Generic MKM System". In: *Intelligent Computer Mathematics*. Ed. by Jacques Carette et al. Lecture Notes in Computer Science 7961. Springer, 2013, pp. 339–343. DOI: 10.1007/978-3-642-39320-4.

[RK13] Florian Rabe and Michael Kohlhase. "A Scalable Module System". In: *Information & Computation* 0.230 (2013), pp. 1–54. URL: http://kwarc.info/frabe/Research/mmt.pdf.

[Wie17] Tom Wiesing. "Enabling Cross-System Communication Using Virtual Theories and QMT". Master's Thesis. Bremen, Germany: Jacobs University Bremen, Aug. 2017. URL: https://github.com/tkw1536/MasterThesis/raw/master/thesis.pdf.

[Wol] *Wolfram—Alpha*. URL: http://www.wolframalpha.com (visited on 01/05/2013).

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

[LMFb] The LMFDB Collaboration. *The L-functions and Modular Forms Database*.