REPORT ON OpenDreamKit DELIVERABLE D6.10

Towards Mathematical Data as VRE components

K. BERČIČ, M. KOHLHASE, F. RABE, T. WIESING
COMPUTER SCIENCE, FAU ERLANGEN-NÜRNBERG

Due on	31/08/2019 (M48)
Delivered on	31/08/2019
Lead	Friedrich-Alexander Universität Erlangen/Nürnberg (FAU)
Progress on and finalization of this deliverable has been tracked publicly at: `https://github.com/OpenDreamKit/OpenDreamKit/issues/134`	

The title of this deliverable was originally *Full-text search (Formulae + Keywords) in OpenDreamKit*, but in the last grant proposal amendment the scope was broadened to a report on the remaining WP6 activities and achievements – also to account for the new task **T6.11**. As the focus last reporting period was on integrating mathematical data, the title was changed – by oversight the title in the grant agreement has not been updated – to better account for this.

This report summarizes the achievements in Work Package 6 over the last year of the OpenDreamKit project. Concretely, this report reports on the results of **T6.10**: *Math Search Engine* and **T6.11**: *Isabelle Case Study* and tasks **T6.6** to **T6.8** (case studies in mathematical data sets).

In the last year of OpenDreamKit, significant progress has been made in four areas:

(1) the understanding of D/K/S-bases: we have codified the experiences in the OpenDreamKit project into a tetrapodal view on “doing mathematics”, which extends the Data/Knowledge/Software model of the proposal by “narration” and “organization” aspects.
(2) the Knowledge (K) aspect, where we have developed an exporter from library the Isabelle Theorem prover (Archive of Formal Proof) to an extensive RDF triple store, which can be queried by standard SPARQL queries for semantic search.
(3) the data (D) aspect, where we have
    (a) developed both an innovative model of (deep) FAIR in mathematics (and have integrated it with the MitM paradigm developed in OpenDreamKit),
    (b) have implemented in a prototypical system (data.mathhub), and have
    (c) evaluated it on the mathematical community outside the core OpenDreamKit community.
(4) computational mathematical documents (the S aspect of D/K/S or the “narration” and “computation” aspects of the finer tetrapod model). Here we have developed a formula harvester for jupyter notebooks and a formula search engine that builds on them (as envisioned in task **T6.10**).

CONTENTS

1. INTRODUCTION

This report summarizes the achievements in Work Package 6 over the last year of the OpenDreamKit project[1]. Concretely, this report reports on the results of **T6.10**: *Math Search Engine* and **T6.11**: *Isabelle Case Study* and tasks **T6.6** to **T6.8** (case studies in mathematical data sets). The work on persistent memoization in Python and GAP (see task **T6.9**) is excluded from this report because it has been reported on in D6.9.

In the last year, significant progress has been made in four areas, which we will introduce in the next subsections (1.1 to 1.4) and then detail the latter three in the corresponding Sections 2 to 4.

1.1. A New Conceptual Model for Mathematical VREs

The probably most fundamental progress for a mathematical research environment is a new understanding of the aspects of virtual research environments reached via the intensive work with the semantics of VREs in WP6. We have codified the experiences in the OpenDreamKit project into a tetrapodal view on "doing mathematics" – i.e. the activity to be supported by a VRE.

We propose that all mathematical VREs need to support the following four main aspects at scale:

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) **Tabulation**: 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.

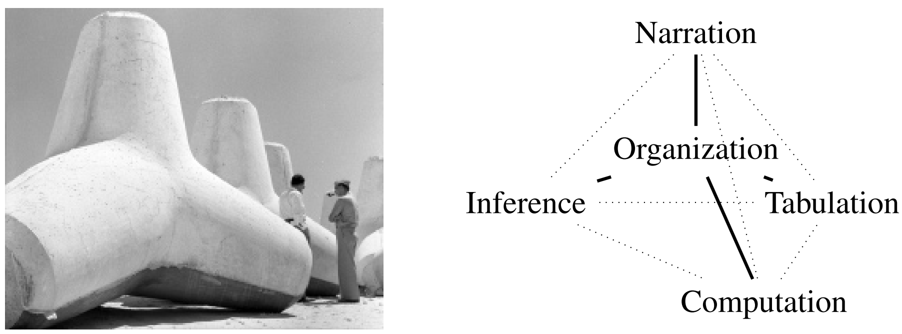


FIGURE 1. Five Aspects of Math VREs, a Tetrapod Structure

Computer support exists for all of these four aspects of Big Math, e.g.,

i) theorem provers like Isabelle, Coq, or Mizar;
ii) computer algebra systems like GAP, SageMath, Maple, or Mathematica; and
iii) mathematical data bases like the L-functions and Modular Forms Data Base (LMFDB) [**Cremona:LMFDB1** **lmfdb:on**] and the Online Encyclopedia of Integer Sequences (OEIS) [**Sloane:OEIS**];
iv) online journals, mathematical information systems like zbMATH or MathSciNet, preprint servers like arXiv.org, or research-level help systems like MathOverflow.

[1] The title of this deliverable was originally *Full-text search (Formulae + Keywords) in OpenDreamKit*, but in the last grant proposal amendment the scope was broadened to a report on the remaining WP6 activities and achievements – also to account for the new task **T6.11**. As the focus last reporting period was on integrating mathematical data, the title was changed to better account for this.

Humans can easily integrate these four aspects and do that for all mathematical developments, the corresponding integration in software systems is still a significant problem. It is our experience from the OpenDreamKit project that one of the prerequisites is a **modular organization** of all four aspects in terms of a joint **mathematical ontology** – the Math-in-the-Middle (MitM) ontology introduced in WP6 of the OpenDreamKit project.

Figure 1 organizes these four aspects into a tetrapodal structure with the MitM ontology. This model extends and refines he Data/Knowledge/Software model of the proposal by "narration" and "organization" aspects (Data (D) $\widehat{=}$ Tabulation, Knowledge (K) $\widehat{=}$ Inference, and Software (S) $\widehat{=}$ Computation). For details and a discussion in terms of "big math" developments like the classification of finite simple groups, consult [**CarFarKohRab:bmobb19**].

The tetrapod model is consistent with the OpenDreamKit structure and organization: the software aspect is mostly covered by WP5: High Performance Math. Computing and WP3: Component Architecture in OpenDreamKit, where the latter develops the aspect of modular **organization** of the software. WP4: User Interfaces covers the interaction between **Narration** and **Software** – again, one of the innovations in the Jupyter framework concerns the modular **Organization** of software interfaces. **Inference** and **Tabulation** are covered in WP6: Data/Knowledge/Software-Bases.

We will now introduce three aspects that have been studied in detail in OpenDreamKit in the last year and make up the technical bulk of this report in the sections after this one.

1.2. Formal Knowledge Bases

For the Knowledge (K) aspect (see **Inference** in Figure 1) we have developed an exporter from library the Isabelle Theorem prover (Archive of Formal Proof) to an extensive RDF triple store, which can be queried by standard SPARQL queries for semantic search.

For many decades, the development of a universal database of all mathematical knowledge, as envisioned, e.g., in the QED manifesto [**qed**], has been a major driving force of computer mathematics. Today a variety of such libraries are available. These are most prominently developed in proof assistants such as Coq [**coq**] or Isabelle [**isabelle**] and are treasure troves of detailed mathematical knowledge. However, despite the enormous potential for many applications of and in virtual research environments, this treasure is usually locked into system- and logic-specific representations that can only be understood by the respective theorem prover system. For example, this precludes applications such as finding related object in knowledge bases from within computation-oriented systems as used in OpenDreamKit.

Therefore, we have developed interface standards that allow maintainers of formal libraries to make their content available to outside systems. In this deliverable, we report on complementing our existing OMDoc/MMT standard for representing entire knowledge bases with a new Upper Library Ontology (ULO). ULO is a standard ontology for exchanging high-level information about mathematical libraries that systematically abstracts all symbolic knowledge away and only retain what can be easily represented relationally. That allows for semantic web-style representations, for which simple and standardized formalisms such as OWL2 [**w3c:owl2-xml**], RDF [**rdf**], and SPARQL [**w3c:SPARQL-Rec:13**] as well as highly scalable tools are readily available. While it is well-known that ontology language–based relational formalisms are inappropriate for symbolic knowledge like formulas, algorithms, and proofs, it is this high-level information that is often critical important for integration into virtual research environments, e.g., to realize benefits like search.

We report on this in Section 2. The bulk of this section is taken by a report on the new task (**T6.11**: *Isabelle Case Study* that we have added to OpenDreamKit in the last amendment of the grant agreement. In this task, we export the large Isabelle knowledge bases as both OMDoc/MMT and ULO format, resulting in datasets in the double-digit GB range. We show the utility of the generated ULO data by setting up a relational query engine that provides easy access to certain library information that was previously hard or impossible to determine.

Relational Datasets Many mathematical datasets take the form of large SQL/CSV–style datasets that enumerate all or a selection of interesting objects satisfying certain properties, e.g., all finite groups up to a certain size. These can be produced in bulk or grow from individual user submissions.

They are generally produced, published, and maintained with virtually no systematic attention to the FAIR principles [**FAIR**; **WilDumAal:FAIR16**] for making data findable, accessible, interoperable, and reusable. In fact, often the sharing of data is an afterthought — see [**Bercic:cmo:wiki**] for an overview of mathematical datasets and their "FAIR-readiness".

Moreover, the inherent complexity of mathematical data makes it very difficult to share in practice: even freely accessible datasets are often very hard or impossible to reuse, let alone make machine-interoperable because there is no systematic way of specifying the relation between the raw data and its mathematical meaning. Therefore, unfortunately FAIR mathematical datasets essentially do not exist today.

1.3. Mathematical Libraries

For the the data (D) aspect of "doing mathematics" (see the **Tabulation** corner in Figure 1) we have

(1) developed an both an innovative model of (deep) FAIR in mathematics (and have integrated it with the MitM paradigm developed in OpenDreamKit),
(2) have implemented in a prototypical system (data.mathhub.info), and have
(3) evaluated it on the mathematical community outside the core OpenDreamKit community.

Traditionally, mathematics has not paid particular attention to the creation and sharing of data — the careful computation and publication of logarithm tables is a typical example of the extent and method. This has changed with the advent of computer-supported mathematics, and the practice of modern mathematics is increasingly data-driven. Today it is routine to use mathematical datasets in the Gigabyte range, including both human-curated and machine-produced data. Examples include the L-Functions and Modular Forms Database (LMFDB; ~ 1 TB data in number theory) [**Cremona:LMFDB16**; **lmfdb:on**] and the GAP Small Groups Library [**GapSmallGroups:on**] with ~ 450 million finite groups. In a few, but increasingly many areas, mathematics has even acquired traits of experimental sciences in that mathematical reality is "measured" at large scale by running computations.

There is wide agreement in mathematics that these datasets should be a common resource and be open and freely available. Moreover, the software used to produce them is usually open source and free as well. Such an ecosystem is embraced by the mathematics community as a general vision for their future research infrastructure [**NAS14**], adopted by the International Mathematical Union as the Global Digital Mathematics Library initiative [**GDML:on**].

To better understand the scale of the problem, Figure 2 gives an overview over some state-of-the-art libraries. Here we already use the division into four kinds of mathematical data that we will develop in Section 3.3.

We present a uniform infrastructure for sharing, finding, and searching mathematical libraries of any kind. Moreover, we seed this infrastructure with a collection of representative libraries (all of them state of the art and large) as described below.

1.4. Mixed Computational and Narrative Documents

Finally, for computational mathematical documents (the S aspect of D/K/S structures from the OpenDreamKit proposal or the **narration** and **computation** aspects of the finer tetrapod model from Figure 1). Here we have developed a formula harvester for Jupyter notebooks and a formula search engine that builds on them (as envisioned in task **T6.10**).

Dataset	Description
Symbolic Knowledge	
Theorem prover libraries	≈ 5 proof libraries, $\approx 10^5$ theorems each, ≈ 200 GB
Computer algebra systems	e.g., SageMath distribution bundles ≈ 4 GB of various tools and libraries
Modelica libraries	> 10 official, > 100 open-source, ≈ 50 commercial, > 5.000 classes in the Standard Library, industrial models can reach .5M equations
Relational Data	
Integer Sequences	≈ 330K sequences, ≈ 1 TB
Sequence Identities	$\approx$ .3M sequence identities, ≈ 2.5 TB
Highly symmetric graphs, maps, polytopes	≈ 30 datasets, $\approx 2 \cdot 10^6$ objects, ≈ 1 TB
Finite lattices	7 datasets, $\approx 17 \cdot 10^9$ objects, ≈ 1.5 TB
Combinatorial statistics and maps	≈ 1.500 objects
SageMath databases	12 datasets
L-functions and modular forms	≈ 80 datasets, $\approx 10^9$ objects, ≈ 1 TB
Linked Data	
zbMATH	$\approx$ 4M publication records with semantic data, $\approx$ 30M reference data, $>$ 1M disambig. authors, $\approx 2,7$M full text links: $\approx$ 1M OA
swMATH	$\approx$ 25K software records with $>$ 300K links to $>$ 180K publications
EuDML	$\approx$ 260K open full-text publications
Wikidata	34 GB linked data, thereof about 4K formula entities, interlinked, e.g., with named theorems, persons, and/or publications
Narrative Data	
arXiv.org	$\approx$ 300K math preprints (of $\approx$ 1.6M) most with LaTeX sources
EuDML	$\approx$ 260K open full-text publications, digitized journal back issues
MathOverFlow	$\approx 1,1$M questions/answers, $\geq$ 11K answer authors
Stacks project	≥ 6000 pages, semantically annotated, curated, searchable textbook
nLab	$\geq$ 13K pages on category theory and applications
Jupyter/Sage Notebooks	$\sim$ 10K public Jupyter notebooks with Sage kernel on GitHub

FIGURE 2. Summary of mathematical libraries

To make this possible, we had to invest a heavy dose of software engineering into the MathWebSearch system: Even though the system has successfully been used as a formula search engine in zbMATH publication information system (see `https://zbmath.org/formulae/`), the deployment of the system required a lot of domain-specific development and workflow integration. To this end we have developed Go bindings for the MathWebSearch daemon, documented the interfaces, and provide a web application wrapper. With this, specific applications only need a domain-specific harvester and minimal customization of a generic front-end.

1.5. Acknowledgments and Prior Dissemination

The tetrapod model sketched in Section 1.1 was developed in close collaboration with Profs. William Farmer and Jacques Carette from McMaster University in Canada. The latter is a member of the OpenDreamKit advisory board. It has been pre-published in [**CarFarKohRab:bmobb19**].

Task **T6.11** was carried out as a subcontract by Makarius Wenzel, the lead developer of the Isabelle system and the only person who could viably carry out the development due to the intimate system knowledge needed for the project. Some parts of Section 1.2 are adapted from his descriptions of his work in the context of this subcontract.

The work reported in Section 2 has been published as [**ConKohMue:rdaml19**] for ULO. The export facilities of Isabelle and the integration with MMT have been integrated with Isabelle in its June 2019 release. It was reported on in two blog posts by Wenzel[2] and a paper is forthcoming.

The theoretical work reported in Section 3 was submitted for publication as [**CarFarKohRab:bmobb19**], while earlier work on the data.mathhub.info system (see Section 3) was reported in [**BerKohRab:tumdi19**].

The software engineering work on the MathWebSearch system from Section has not been published. The concrete application to searching Jupyter notebooks has been carried out together with Dr. Nicolas Thiérry.

[2] `https://sketis.net/2018/isabelle-mmt-export-of-isabelle-theories-and-import-as-omdoc-content`, and `https://sketis.net/2019/mmt-as-component-for-isabelle2019`

2. Formalized Knowledge

The design of OMDoc/MMT predates OpenDreamKit, and we have reported on it previously [**Kohlhase:OMDoc1.2**; **RabKoh:WSMSML13**; **DehKohKon:iop16**; **KohMuePfe:kbimss17**]. Therefore, we focus on the design of ULO here, which we report in Section 2.1.

Then, in Section 1.2, we report on **T6.11**, which exports the Isabelle knowledge base into OMDoc/MMT and ULO. The latter includes the generation of ULO data from concrete libraries in RDF format. The resulting Isabelle Theorem dataset is massive, e.g., resulting in $> 10^7$ RDF triples, requiring over 10 CPU-hours to generate. Contrary to the RDF formar, the OMDoc format does not abstract from the knowledge base, making it even larger. (In fact, we had to redesign MMT to allow for streaming compressed OMDoc files because it became infeasible to process them otherwise.) In parallel work, we have developed a similar export of the Coq library [**MueRabSac:cltg19**], which is even bigger, but which we do not present in detail here.

Finally, in Section 2.3, we demonstrate how to leverage these lightweight, high-level representations in practice. As an example application, we set up a relational query engine based on Virtuoso. It answers complex queries instantaneously, and even simple queries allow obtaining information that was previously impossible or expensive to extract. Example queries include asking for all theorems of any library whose proof uses induction on $\mathbb{N}$, or all authors of theorems ordered by how many of the proofs are incomplete, or all dependency paths through a particular library ordered by cumulative check time (which would enable optimized regression testing).

2.1. The Upper Library Ontology

We use a simple data representation language for upper-level information about libraries. This **Upper Library Ontology** (ULO) describes objects in theorem prover libraries, their taxonomy, and relations as well as organizational and information. The ULO allows the export of upper-level library data from theorem prover libraries as RDF/XML files (see §2.2), and gives meaning to them. The ULO is implemented as an OWL2 ontology, and can be found at `https://gl.mathhub.info/ulo/ulo/blob/master/ulo.owl`. All new concepts have URIs in the namespace `https://mathhub.info/ulo`, for which we use the prefix ulo: below. In the sequel we give an overview of the ULO, and we refer to [**ULODoc:on**] for the full documentation.

2.1.1. *Individuals.* Individuals are the atomic objects relevant for mathematical libraries. Notably, they do not live in the `ulo` namespace but in the namespace of their library.

These include in particular all globally named objects in the library such as theories/modules/etc, types, constants, functions, predicates, axioms, theorems, tactics, proof rules, packages, directories, files, sections/paragraphs, etc. For each library, these individuals usually share a common namespace (an initial segment of their URI) and then follow a hierarchic schema, whose precise semantics depends on the library.

Additionally, the individuals include other datasets such as researchers as given by their ORCID or real name, research articles as given by their DOI, research software systems as given by their URI in swMATH[3], or MSC[4] and ACM[5] subject classes as given by their respective URIs. These individuals are not generated by our export but may occur as the values of key-value attributions to the individuals in prover libraries.

2.1.2. *Classes.* Classes can be seen as unary predicates on individuals, tags, or soft types. The semantic web conventions tend to see them simply as special individuals that occur as values of the is-a property of other individuals. Figure 3 gives an overview of the most important classes in the ULO.

[3] `https://swmath.org/software/NUMBER`
[4] `http://msc2010.org/resources/MSC/2010/CLASS`
[5] `https://www.acm.org/publications/class-2012`

Logical Role The logical classes describe an individual's formal role in the logic, e.g., the information that $\mathbb{N}$ is a type but 0 an object.

`ulo:theory` refers to any semantically meaningful group of named objects (declarations). There is a wide range of related but subtly different concepts using words such theory, class, signature, module type, module, functor, locale, instances, structure, locale interpretation, etc.

Inside theories, we distinguish five classes of declarations depending on what kind of entity is constructed by an individual: `ulo:type` if it constructs types or sorts like $\mathbb{N}$ or $list$; `ulo:function` if it constructs inhabitants of types like $+$ or nil; `ulo:predicate` if it constructs booleans/propositions such as $=$ or $\mathrm{nonEmpty}$; `ulo:statement`[6] if it establishes the truth of a proposition such as any axioms, theorem, inference rule; and finally `ulo:universe` if it constructs collections of types such as Set or Class.

Note that while we hold the distinction of these five classes to be universal, concrete logics may not always distinguish them syntactically. For example, HOL identifies functions and predicates, but the extractor can indicate whether a declaration's return type is the distinguished type of booleans. Similarly, Curry-Howard-based systems identity predicates and types as well as statements and objects, which an extractor may choose to separate.

Orthogonally to the above, we distinguish declarations by their definition status: `ulo:primitive` if it introduces a new concept without a definition such as an urelement or an axiom; and `ulo:derived` if it can be seen as an abbreviation for an existing concept like a defined operator or a theorem. For example, intersecting the classes `ulo:statement` and `ulo:derived`, we capture all theorems.

While the primitive-derived distinction is clear-cut for definition-based systems like Coq, it is trickier for axiom-based systems like Isabelle: an Isabelle definition actually consists of a primitive concept with a defining axioms for it. For that purpose, we introduce the `ulo:defines` property in §2.1.3.

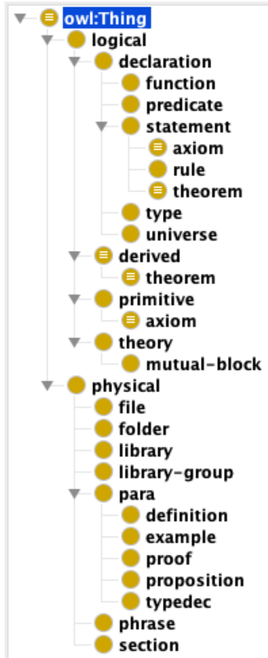


FIGURE 3. ULO Classes

Physical Role The physical classes describe an individual's role in the physical organization of a library. This includes for an individual i:

- `ulo:section` if i is an informal grouping inside a file (chapter, paragraph etc.)
- `ulo:file` if i is a file
- `ulo:folder` if i is a grouping level above source files inside a library, e.g., a folder, sub-package, namespace, or session
- `ulo:library` if i is a library. Libraries have logical URIs and serve as the root objects containing all other individuals. A library is typically maintained and distributed as a whole, e.g., via a GitHub repository. A library has a logical URI and the URIs of individuals are typically formed relative to it.
- `ulo:library-group` if i is a group of libraries, e.g., a GitHub group.

In addition we define some classes for the lowest organizational level, called *logical paragraphs*. These are inspired by definition–example–theorem–proof seen in informal mathematics and often correspond to LaTeX environments. In formal libraries, the individuals of these classes may be the same as the ones for the logical classes or different ones. For example, a document-oriented system like Isabelle could assign a physical identifier to a paragraph and a different logical one to the formal theorem inside it. These identifiers could then have classes `ulo:proposition`

[6]We have reconsidered the name of this class many times: all suggested names can be misunderstood. The current name stems from the intuition that axioms and theorems are the most important named truth-establishing declarations, and *statement* is a common way to unify them. Arguably more systematic would be *proof*: anything that establishes truth is formalized as an operator that constructs a proof.

and `ulo:statement` respectively. A purely formal system could omit the physical class or add it to the logical identifier, e.g., to mark a logical definition as an `ulo:example` or `ulo:counter-example`. Some of these, in particular, theorems given informal classes like "Lemma" or "Hauptsatz", a string which can be specified by the `ulo:paratype` relation (see below).

2.1.3. *Properties.* All properties are binary predicates whose first argument is an individual. The second argument can be an individual (**object property**) or a value (**data property**). Unless mentioned otherwise, we allow the same property to be used multiple times for the same individual.

The two kinds are often treated differently. For example, for visualization as a graph, we can make individuals nodes (using different colors, shapes etc. depending on which classes a node has) and object properties edges (using different colors, shapes, etc. for different properties). The data properties on the other hand would be collected into a key-value list and visualized at the node. Another important difference is during querying: object properties are relations between individuals and thus admit relational algebra such as union and intersection or symmetric and transitive closure. Data properties on the other hand are usually used with filters that select all individuals with certain value properties.

Library Structure Individuals naturally form a forest consisting e.g., of (from roots to leafs) library groups, libraries, folders, files, section, modules, groups of mutual recursive objects, constants. Moreover, the dependency relation between individuals (in particular between the leaves of the forest) defines an orthogonal structure.

`ulo:specifies`(i, j) expresses that j is a child of i in the forest structure. Thus, taking the transitive closure of `ulo:specifies` starting with a library, yields all individuals declared in a library.

`ulo:uses`(i, j) expresses that j was used to check i, where j may include extra-logical individuals such as tactics, rules, notations. A very frequent case is for j to be an occurrence of a logical individual (e.g. a constant or a theorem). The case of occurrences leads to the question about what information can be attached to an occurrence. Examples could be: the number of repetitions of the occurrence; whether the occurrence of a constant induces a dependency on the type only, or on the actual definition as well; where the occurrence is located (e.g. in the statement vs proof, in the type vs body or in more specific positions, like as the head symbol of the conclusion, see [**AGSTZ:ContMathSearchWhelp04**] for a set of descriptions of positions that is useful for searching up to instantiation). For now we decided to avoid to specify occurrences in the ontology, for the lack of a clear understanding of what properties will really be useful for applications. Integrating the ULO ontology with occurrences is left for future work towards ULO 1.0.

Semantic Relations between Declarations Relational representations treat individuals as black boxes. But sometimes it is helpful to expose a little more detail about the internal structure of a declaration. For that we define the following properties:

- `ulo:defines`(i, j) is used to relate a declaration j to its definition i if the two have different identifiers, e.g., because they occur in different places in the source file, or because i is a defining axiom for a constant j.
- `ulo:justifies`(i, j) relates any kind of argument i to the thesis j it supports. The most important example is relating a proof to its theorem statement if the two have different identifiers.
- `ulo:instance-of`(i, j) relates a structuring declaration j to the theory-like entity i that realizes, e.g., a module to its module type, an instance to its (type) class, a model to its theory, or an implementation to its specification.

- `ulo:generated-by`(i, j) expresses that i was generated by j, e.g., the user may define an inductive type j and the systems automatically generated an induction schema i.
- `ulo:inductive-on`(i, j) expresses that i is defined/proved by induction on the type j.

Informal Cross-References First we define some self-explanatory cross-references that are typically (but not necessarily) used to link individuals within a library. These include `ulo:same-as`, `ulo:similar-to`, `ulo:alternative-for`, `ulo:see-also`, `ulo:generalizes`, and `ulo:antonym-of`.

Second we define some cross-references that are typically used to link a knowledge item in a library to the outside. Of particular relevance are:

- `ulo:formalizes`(i, j) indicates that j is an object in the informal realm, e.g., a theorem in an article, that is formalized/implemented by i.
- `ulo:aligned-with`(i, j) indicates that i and j formalize/implement the same mathematical concept (but possibly in different ways).

Data Properties All properties so far were object properties. Data properties are mostly used to attach metadata to an individual. We do not introduce new names for the general-purpose metadata properties that have already been standardized in the Dublin Core such as `dcterms:creator`, `dcterms:title`, `dcterms:contributor`, `dcterms:description`, `dcterms:date`, `dcterms:isVersionOf`, `dcterms:source`, `dcterms:license`. But we define some new data properties that are of particular interest for math libraries:

- `ulo:name`(i, v) attributes a string v to a declaration that expresses the (user-provided) name as which it occurs in formulas. This is necessary in case an individual generated URI is very different from the name visible to users, e.g., if the URI is generated from an internal identifier or if the name uses characters that are illegal in URIs.
- `ulo:sourceref`(i, v) expresses that v is the URI of the physical location (e.g., file, line, column in terms of UTF-8 or UTF-16 characters) of the source code that introduced i.
- `ulo:docref`(i, v) expresses that v is the URI reference to a place where f is documented (usually in some read-only rich text format).
- `ulo:check-time`(i, v) expresses that v is the time (a natural number giving a time in milliseconds) it took to check the declaration that introduced i.
- `ulo:external-size`(i, v) expresses that v measures the source code of i (similar to positions above).
- `ulo:internal-size`(i, v) expresses that v is the number of bytes in the internal representation of i including inferred objects and generated proofs.
- `ulo:paratype`(i, v) gives the "type" of a logical paragraph, i.e. something like "Lemma", "Conjecture", ... This is currently a string, but will become a finite enumeration eventually.

Locations, sizes, and times may be approximate

Organizational Status Finally, we define a few (not mutually exclusive) classes that library management–related information such as being experimental or deprecated. Many of these are known from software management in general. The unary properties are realized as data properties, where the object is an explanatory string, the binary relations as object properties. An important logic-specific class is `ulo:automatically-proved` — it applies to any theorem, proof step, or similar that was discharged automatically (rather than by an interactive proof).

2.2. Exporting the Isabelle Knowledge Base

2.2.1. *Isabelle Overview.* Isabelle is a generic platform for tools based on formal logic. It is generally known for its Isabelle/HOL library, which provides many theories and add-on tools (implemented in Isabelle/ML) in its `Main` theory and the `main` group of library sessions. Some other (much smaller) Isabelle logics are FOL, LCF, ZF, CTT (an old version of Martin-Löf Type Theory), but today most Isabelle applications are based on HOL. The foundations of Isabelle due to Paulson [**paulson700**] are historically connected to *logical frameworks* like Edinburgh LF: this fits nicely to the LF theory used in logic formalizations in MMT. User contributions are centrally maintained in AFP, the *Archive of Formal Proofs* (`https://www.isa-afp.org`).

From a high-level perspective, Isabelle is better understood as *document-oriented proof assistant* or *document preparation system* for domain-specific formal languages [**Wenzel:IIdsflitd18**]. It allows flexible nesting of sub-languages, and types, terms, propositions, and proofs (in Isabelle/Isar) are merely a special case of that. The result of processing Isabelle document sources consists of internal data structures in Isabelle/ML that are private to the language implementations. Thus it is inherently difficult to observe Isabelle document content by external tools, e.g. to see which λ-terms occur in nested sub-languages.

Isabelle/PIDE [**Wenzel:2014:ITP-PIDE**], the Prover IDE framework, integrates all activities of development for proofs and programs into a *semantic editor* Isabelle/jEdit. While the user is composing text, the prover provides real-time feedback about its meaning: that rich PIDE markup information can be rendered as conventional GUI metaphors (e.g. text color, squiggly underline, tooltips, hyperlinks, icons in the border) [**Wenzel:2019:MKM**]. It is also possible to run Isabelle/PIDE in "headless mode", to let some Isabelle/Scala program observe markup as a formal library is processed in Isabelle/ML.

The standard distribution of Isabelle[7] includes the Isabelle/HOL library with various applications, but the main bulk of applications is in the *Archive of Formal Proofs* (AFP)[8], which is organized like a scientific online journal. In August 2019, Isabelle/AFP had 492 articles by 331 authors, comprising 120 MB source text total. Formal checking requires approx. 50h CPU time or 2h elapsed time on high-end multicore hardware. The result is a collection of heap images for the internal state of Isabelle/ML and some HTML or PDF documents that resemble conventional mathematical texts (with relatively little formal markup being used here).

2.2.2. *General Technical Aspects of Exporting Isabelle Knowledge Bases.* Isabelle/ML is the *internal implementation language* for logic-based tools, has direct access to the inference kernel in the manner of the "LCF-approach" [**Gordon-Milner-Wadsworth:1979**]. Isabelle/Scala is the *external interface language* to manage Isabelle processes and resulting content: it has access to the physical world with GUI frameworks, TCP servers, database engines etc.

To support exports like ours systematically, Wenzel had previously already instrumented the theory processing in Isabelle/ML to allow arbitrary *presentation* for every theory node as it is finished: an ML function gets access to all commands with their intermediate theory values; it can extract suitable information and *export* it via a private channel to Isabelle/Scala. The standard configuration of Isabelle provides options like `export_theory` or `export_proofs` to externalize certain aspects of theory content as a *session database*. For example, see command-line tools like `isabelle build -o export_theory HOL-Analysis` to build such a database (for SQLite) and `isabelle export -l HOL-Analysis` to query it later on.

Isabelle/Scala provides direct access to session builds and their results as an API for *typed functional-object-oriented programming*: this is more robust and versatile than command-line tools. For our export, we use the Isabelle/Scala APIs to provide dedicate command-line `isabelle mmt_import`: options and arguments similar to `isabelle build` specify a

[7] `https://isabelle.in.tum.de`
[8] `https://www.isa-afp.org`

collection of sessions to be processed, the results are given to the MMT Scala API to write files in OMDoc format; there is extra output of RDF/XML files.

Our export (Isabelle repository version e6fa4b852bf9) turns this content into OMDoc and RDF/XML. Our command-line tool uses regular Scala APIs of MMT (without intermediate files), and results are written to the file-system in OMDoc and RDF/XML format. It runs a Prover IDE session (PIDE) under program control: a collection of *sessions* with corresponding *theories* is turned into formal source edits that are given to Isabelle/PIDE for semantic processing; the underlying prover process continuously produces markup reports as it explores the meaning of the source. Whenever some theory node (with all imports) is finished, a separate Scala operation is invoked to *commit* the result as PIDE *document snapshot*. Thus the application can harvest theory exports produced up to that point, and the system can edit-out already processed theory-subgraphs to free resources, while the overall process is still running.

Thus the full material of AFP (hundreds of sessions, thousands of theories) can be digested on a high-end multicore machine within approx. 24h elapsed time using 60 GB RAM for Isabelle/ML and 30 GB RAM for Isabelle/Scala. This is approximately a factor 10 more than a conventional batch build, but the PIDE session is able to "see" more detailed information about the formal meaning of the sources (e.g. the use of term constants within the theory text, like the IDE front-end uses to provide hyperlink operations).

2.2.3. *Exporting Symbolic Knowledge in OMDoc/MMT Format.* We export all *logical foundations* of theory documents (types, consts, facts, but *not* proof terms), and aspects of *structured specifications* (or "little theories") (locales and locale interpretations, which also subsumes the logical content of type classes). Figure 4 shows a screenshot of the MMT browser showing a small part (the definition of binomial coefficients in Isabelle/HOL) of the exported Isabelle knowledge base.

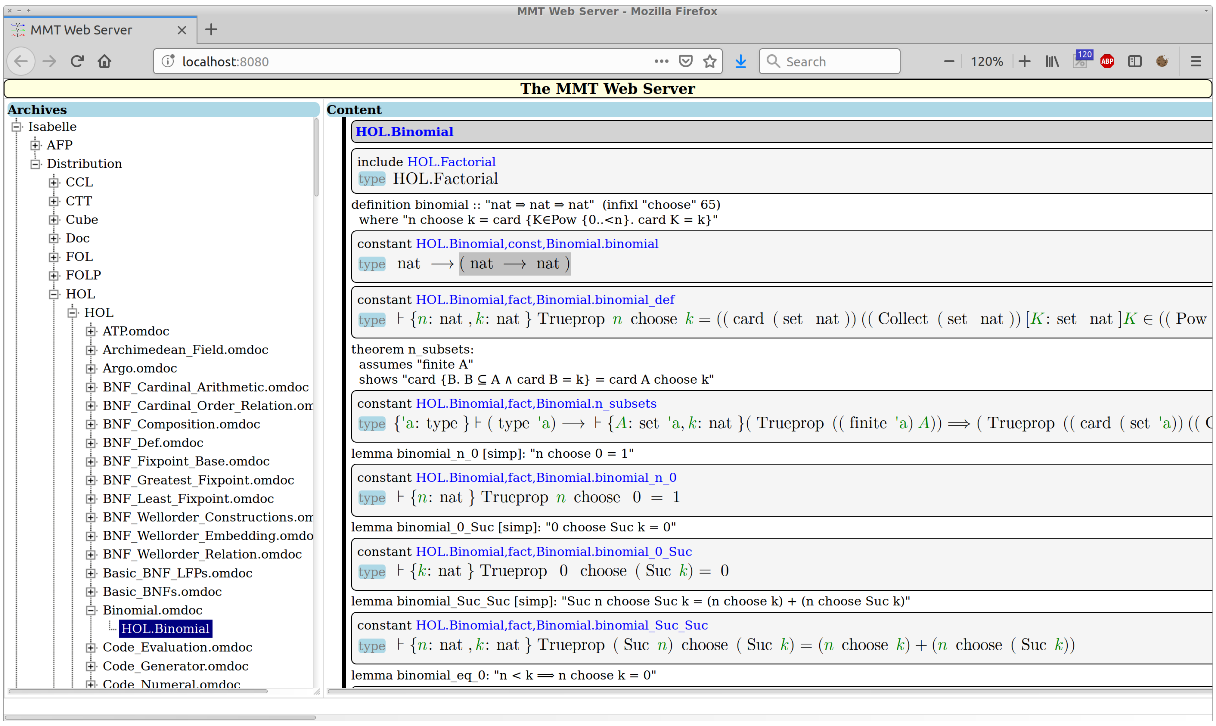


FIGURE 4. MMT browser showing Isabelle content

Module System All module system–level Isabelle entities are represented as MMT theories and morphisms. These use a manually written MMT meta-theory that formalizes Isabelle/Pure — the logic underlying Isabelle.

Besides plain Isabelle theories, this includes the order-sorted algebra of *type classes* (subclass relation) and *type arities* (image behavior of type constructors wrt. type class domains and ranges) in the sense of Nipkow and Prehofer [**Nipkow-Prehofer:1993**] as well as locales in the sense of Ballarin [**Ballarin2014**] and type classes as special locale interpretations in the sense of Haftmann and Wenzel [**Haftmann-Wenzel:2006:classes**; **Haftmann-Wenzel:2009**].

Isabelle locales are *named contexts* with type parameters (implicit), term parameters (**fixes**), and premises (**assumes**). Within such a "little theory" it is possible to spell out definitions, statements, proofs as usual — all results are understood as relative to the context. The foundations will contain an extra prefix of type variables, term abstractions and assumptions according to the context.

The export of locales preserves some of its internal structure, notably the locale dependency relation stemming from the construction of locales and sub-locales (by definition), as well as later locale interpretations (by proof).

Type classes are special locales with a single type parameter and a canonical locale interpretation to connect *type class parameters* (polymorphic constants with class constraints) to *locale parameters* (fixed variables of the context). The export shows the result of the interpretation, but *without* an explicit report on their connection.

Declarations Every declaration in a theory-like scope is represented as an MMT declaration. This includes **types** (base types and type constructors), term **constants** (including functions, binders, quantifiers as higher-order constants), **axioms** (including equational axioms that count as primitive definitions), and **theorems** (propositions with a proof).

Isabelle definitions become equational axioms in MMT. This includes theorems, which are treated as defined constants in MMT via the Curry-Howard representation. However, the actual proofs, which become the definientia of these constants, are not exported by default (proof terms are prohibitively large), but it is possible to reconstruct the source text in the Isabelle/Isar proof language from the position information.

Type definitions in the sense of Gordon and Pitts [**pitts93**] are interpreted definitionally within the standard semantics of the HOL logic. The export facility provides access to the key information: the old representing type, the new abstract type, the name of the morphisms between the two with the axiom stating the relation. This information allows recovering HOL typedefs faithfully, where Isabelle/Pure theory content would only show the individual particles. It also serves as an example to "query" derived specification mechanisms in Isabelle/ML, to expose its own level of abstraction to the exporter.

Term constants with indication of derived specifications mechanisms, e.g. **primrec** functions, **inductive** or **coinductive** relations are handled by querying generic information in Isabelle/Pure about functional or relation specifications (so-called "Spec Rules"). The Isabelle/HOL implementations provide this data on their own account.

Expressions The expressions of Isabelle's λ-calculus are represented as MMT terms. This part of the translation is relatively straightforward and similar to previous exports to OMDoc/MMT (e.g., [**KalRab:hollight:14**; **KohMueOwr:mpagsiuf17**; **MueRabSac:cltg19**]). Only the treatment of variables warrants further description, and we focus on it in the sequel.

Isabelle variables come in various flavours: free variables (e.g. `x`), schematic variables with index (e.g. `?x10`) and bound variables (e.g. `x` in `λx::τ. x` which is concrete syntax for the de-Bruijn index abstraction `Abs (x, τ, B.0)` where `x` is retained as a comment). To fit smoothly into the lambda-calculus of MMT, variable names are standardized as follows:

Schematic variables are renamed to fresh free variables. Since schematic variables are morally like a universal quantifier prefix, this preserves the logical meaning of a formala (e.g. a theorem statement).

Bound variable comments in abstractions are renamed locally to avoid clashes with free variables in the same scope. Thus the comment can be used literally in MMT/LF as named abstraction, ignoring the unnamed de-Bruijn index representation of Isabelle.

Finally, Isabelle type variables are decorated with type class constraints, e.g. `'a::order` for types that belong to the class `order` defined in the Isabelle/HOL library (e.g. `nat` with its standard order): this ensures certain *operations* with a link to overloaded term constants (e.g. `less :: 'a => 'a => bool`), as well as logical *premises* on these operations (e.g. stating that `less` is a strict order on the type).

Isabelle class operations are managed by extra-logical means, e.g. by the code-generator for Haskell or Standard ML, to produce the expected *dictionary construction* to eliminate the implicit overloading. In MMT this will merely result in uncontrolled polymorphism: constant definitions consist of multiple (non-overlapping) equational specifications depending on the type argument.

Isabelle class premises become logical constraints in a straight-forward manner: a type class is a predicate over types in LF, so `'a::c` means that the predicate `c` applied to type `'a` holds. Statements with class constraints $\phi(\texttt{'a::c})$ are augmented by a prefix of preconditions $\texttt{'a::c} \Longrightarrow \phi(\texttt{'a})$, effectively eliminating the constraint within the logic.

Identifiers Isabelle constants live in separate name spaces (for types, terms, theorems etc.). For example, there could be a type `Nat.nat` and a term constant of the same name (e.g. an operation to make a natural number). Qualification is usually by the theory *base* name, not the session-qualified long name; in rare situations, there is no theory qualification at all. In order to have all entities coexist within one big space of individuals in MMT, we use a triple that consists of (*long-theory-name*, *entity-name*, *entity-kind*) written as URI like as follows:

`https://isabelle.in.tum.de?`*long-theory-name*`?`*entity-name*`|`*entity-kind*

For example, `https://isabelle.in.tum.de?HOL.Nat?Nat.nat|type` refers to the type of natural numbers in the Isabelle/HOL library.

2.2.4. *Exporting Relational Knowledge in ULO Format.* Relations between formal items are output as RDF/XML triples, bypassing MMT and its OMDoc format; see also [**ConKohMue:rdaml19**]. This also captures some aspects of inductive and primitive recursive definitions via `ulo:inductive-on`. In addition to that paper, there is now full coverage of theorem dependencies via `ulo:uses`, spanning a rather large dependency graph over the source text: it relates theorem statements with used constants and proofs with used theorems.

Individuals Formal entities are identified by their *name* and *kind* as follows:

- The name is a long identifier (with dot as separator, e.g. `Nat.Suc`) that is unique within the current theory context (including the union of all theory imports). Long names are managed by namespaces within the formal context to allow partially qualified names in user input and output (e.g. `Suc`). The structure of namespaces is known to the prover, and not exported.
- The kind is a short identifier to distinguish the namespaces of formal entities, e.g. `type` for type constructors, `const` for term constants, `fact` for lists of theorems that are recorded in the context, but also non-logical items like `method` (Isar proof methods), `attribute` (Isar hint language) etc.

This name/kind scheme is in contrast to usual practice in universal λ-calculus representations like MMT/LF, e.g. there could be a type `Nat.nat` and a separate term constant of the same name. Moreover the qualification in long names only uses theory base names, not their session-qualified long name (which was newly introduced in Isabelle2017). So in order to support one big space of individuals over all Isabelle sessions and theories, we use the URI format that was already used for identifiers above.

Logic The primitive logical entities of Isabelle/Pure are types, terms, and theorems (facts). Additionally, Isabelle supports various theory-like structures. These correspond our declaration classes as follows:

- `ulo:theory` refers to global **theory** and local **locale** contexts. There are various derivatives of **locale** that are not specifically classified, notably **class** (type classes) and **experiment** (locales with inaccessible namespace).
- `ulo:type` refers to *type constructors* of Isabelle/Pure, and object-logic types of many-sorted FOL or simply-typed HOL. These types are syntactic, and not to be confused with the "propositions-as-types" approach in systems like Coq. Dependent types are represented as terms in Isabelle.
- `ulo:function` refers to *term constants*, which are ubiquitous in object-logics and applications. This covers a broad range of formal concepts, e.g. logical connectives, quantifiers (as operators on suitable λ-terms), genuine constants or mathematical functions, but also recursion schemes, or summation, limit, integration operators as higher-order functions.
- `ulo:statement` refers to individual theorems, which are projections from the simultaneous `fact` lists of Isabelle. Only the head statement of a theorem is considered, its proof body remains abstract (as reference Isar to proof text). Theorems that emerge axiomatically (command **axiomatization**) are marked as `ulo:primitive`, properly proven theorems as `ulo:derived`, and theorems with unfinished proofs (command **sorry**) as `ulo:experimental`.

The `ulo:specifies` and `ulo:specified-in` relations connect theories and locales with their declared individuals. The `ulo:uses` relation between those represents syntactic occurrence of individuals in the type (or defining term) of formal entities in Isabelle: it spans a large acyclic graph of dependencies. Again, this excludes proofs: in principle there could be a record of individuals used in the proof text or by the inference engine, but this is presently unimplemented.

The `ulo:source-ref` property refers to the defining position of formal entities in the source. Thanks to Isabelle/PIDE, this information is always available and accurate: the Prover IDE uses it for highlighting and hyperlinks in the editor view. Here we use existing URI notation of MMT, e.g. `https://isabelle.in.tum.de/source/FOL/FOL/FOL.theory#375.19.2:383.19.10` with offset / line / column of the two end-points of a text interval.

The `ulo:check-time` and `ulo:external-size` properties provide some measures of big theories in time (elapsed) and space (sources). This is also available for individual commands, but it is hard to relate to resulting formal entities: a single command may produce multiple types, consts, and facts simultaneously.

Semi-formal Documents We use `ulo:section` for the six levels of headings in Isabelle documents: **chapter**, **section**, ..., **subparagraph**. These are turned into dummy individuals (which are counted consecutively for each theory).

`ulo:file`, `ulo:folder`, `ulo:library` are presently unused. They could refer to the overall project structure Isabelle document sources in the sense of [**Wenzel:IIdsflitd18**], namely as *theories* (text files), *sessions* (managed collections of theories), and *project directories* (repository with multiple session roots).

For document metadata, we use the Dublin Core ontology. The Isabelle command language has been changed to support a new variant of *formal comment*. With one keystroke, the presentation context of a command may be augmented by arbitrary user-defined marker expressions. Isabelle/Pure already provides `title`, `creator`, `contributor` etc. from §2.1.3: they produce PIDE document markup that Isabelle/MMT can access and output as corresponding RDF.

This approach allows to annotate theory content *manually*: a few theories of `HOL-Algebra` already use the new feature sporadically. For automatic marking, metadata of AFP entries is

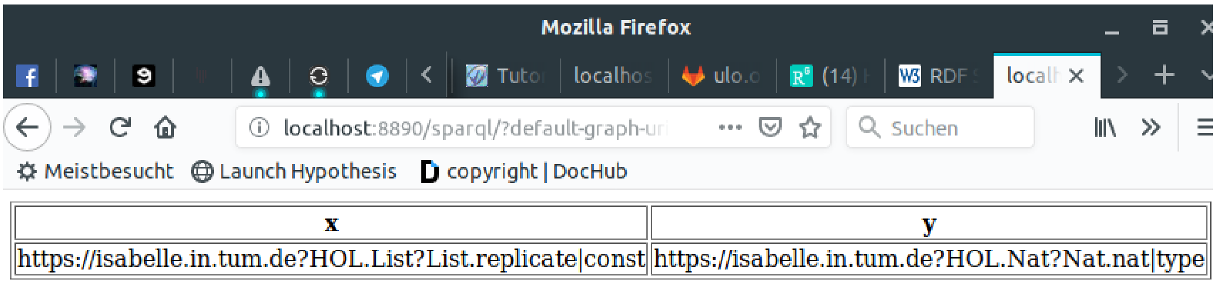


FIGURE 5. Virtuoso Output for the Example Query using Alignments

re-used for their theories. One could also digest comments in theory files about authors, but this is presently unimplemented.

2.2.5. *Statistics.* We present some statistics[9]. This gives an idea about overall size and scalability of the export facilities so far. The datasets are publicly available from `https://gl.mathhub.info/Isabelle`.

Library	Individuals	Relations	Theories	Locales	Types	Constants	Statements	RDF/XML file size	elapsed time
Distribution only group `main`	103,873	2,310,704	535	496	235	8,973	88,960	188 MB	0.5h
Distribution+AFP without `very_slow`	1,619,889	36,976,562	6,185	4,599	10,592	215,878	1,359,297	3,154 MB	16.5h

2.3. Applications

In this section, we evaluate the ULO framework, i.e. the ULO ontology and the generated RDF data by showing how they could be exploited using standard tools of the Semantic Web tool stack.

We have set up an instance of Virtuoso Open-Source Edition[10], which reads the exports described in Section 2.2 and provides a web interface with a SPARQL endpoint to experiment with the ULO dataset. Then we have tried several queries with promising results (just one shown below for lack of space). The queries are not meant to be a scientific contribution per se: they just show how much can be accomplished with the ULO dataset with standard tools in one afternoon.

Example query: all recursive functions on $\mathbb{N}$ For this, we use the ulo:inductive−on relation to determine inductive definitions on a type ?y, which we restrict to one that is aligned with the type nat_lit of natural numbers from the interface theory NatLiterals in the Math-in-the-Middle Ontology.

```
SELECT ?x ?y WHERE {
  ?x ulo:inductive−on ?y .
  http://mathhub.info/MitM/Foundation?NatLiterals?nat_lit ulo:aligned−with ?y . }
```

Note that we use alignments [**MueGauKal:cacfms17**] with concepts from an interface theory as a way of specifying "the natural numbers" across theorem prover libraries. The result is a list of pairs: each pair combines a specific implementation of natural numbers (Isabelle has several, depending on the object-logic), together with a function defined by reduction on it. A subset of the results of this query are shown in Figure 5.
[1] EdN:1

Transitive Queries The result of the query above only depends on the explicitly generated RDF triples. Semantic Web tools that understand OWL allow more complex queries. For example,

[9] Versions: Isabelle/9c60fcfdf495, AFP/d50417d0ae64, MMT/e6fa4b852bf9.

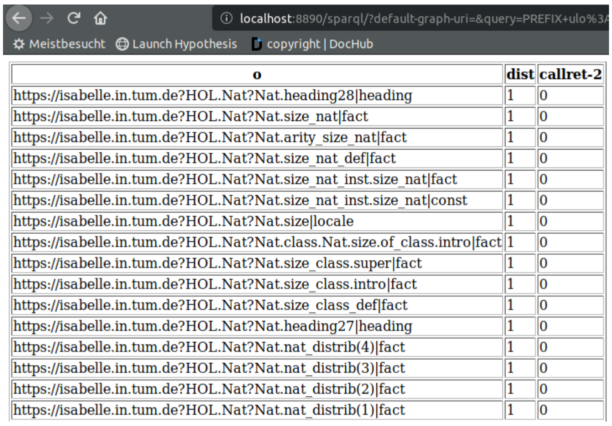
[10] `https://github.com/openlink/virtuoso-opensource`

[1] EDNOTE: check if Dennis can supply Isabelle screenshots

Virtuoso implements custom extensions that allow for querying the transitive closure of a relation. The resulting query syntax is a little convoluted, and we omit some details in the example below.

```
SELECT ?o ?dist WHERE { {
      SELECT ?s ?o WHERE { ?s ulo:declares ?o }
    }
    OPTION ( TRANSITIVE, t_distinct, t_in(?s), t_out(?o), t_min (1),
             t_max (10), t_step ('step_no') as ?dist ) .
    FILTER ( ?s = <https://isabelle.in.tum.de?HOL.Nat> )
  }
ORDER BY ?dist DESC 2
```

The above code queries for all symbols recursively declared in the (effectively randomly chosen) Theory `HOL.Nat` declaring the natural numbers and associated concepts; the output for that query is shown in Figure 6.



o	dist	callret-2
https://isabelle.in.tum.de?HOL.Nat?Nat.heading28|heading	1	0
https://isabelle.in.tum.de?HOL.Nat?Nat.size_nat|fact	1	0
https://isabelle.in.tum.de?HOL.Nat?Nat.arity_size_nat|fact	1	0
https://isabelle.in.tum.de?HOL.Nat?Nat.size_nat_def|fact	1	0
https://isabelle.in.tum.de?HOL.Nat?Nat.size_nat_inst.size_nat|fact	1	0
https://isabelle.in.tum.de?HOL.Nat?Nat.size_nat_inst.size_nat|const	1	0
https://isabelle.in.tum.de?HOL.Nat?Nat.size|locale	1	0
https://isabelle.in.tum.de?HOL.Nat?Nat.class.Nat.size.of_class.intro|fact	1	0
https://isabelle.in.tum.de?HOL.Nat?Nat.size_class.super|fact	1	0
https://isabelle.in.tum.de?HOL.Nat?Nat.size_class.intro|fact	1	0
https://isabelle.in.tum.de?HOL.Nat?Nat.size_class_def|fact	1	0
https://isabelle.in.tum.de?HOL.Nat?Nat.heading27|heading	1	0
https://isabelle.in.tum.de?HOL.Nat?Nat.nat_distrib(4)|fact	1	0
https://isabelle.in.tum.de?HOL.Nat?Nat.nat_distrib(3)|fact	1	0
https://isabelle.in.tum.de?HOL.Nat?Nat.nat_distrib(2)|fact	1	0
https://isabelle.in.tum.de?HOL.Nat?Nat.nat_distrib(1)|fact	1	0

FIGURE 6. Virtuoso Output for the Transitive Example Query

Interesting examples of library management queries which can be modeled in SPARQL (and its various extensions, e.g. by rules) are found in [**conf/lpar/AspinallDL12**]. Instead [**AGSTZ:ContMathSearchV**
MKM04:AspertiS04] show examples of interesting queries (approximate search of formulae up to instantiation or generalization) that can be implemented over RDF triples, but that requires an extension of SPARQL with subset and superset predicates over sets.

3. Mathematical Datasets

Modern mathematical research increasingly depends on collaborative tools, computational environments, and online databases, and these are changing the way mathematical research is conducted and how it is turned into applications. For example, engineers now use mathematical tools to build and simulate physical models based on systems of differential equations with millions of variables, combining building blocks and algorithms taken from libraries shared all over the internet.

State of FAIRness for Mathematical Datasets Mathematical datasets are generally produced, published, and maintained with virtually no systematic attention to the FAIR principles [**FAIR**; **WilDumAal:FAIR16**] for making data findable, accessible, interoperable, and reusable. In fact, often the sharing of data is an afterthought — see [**Bercic:cmo:wiki**] for an overview of mathematical datasets and their “FAIR-readiness”.

Moreover, the inherent complexity of mathematical data makes it very difficult to share in practice: even freely accessible datasets are often very hard or impossible to reuse, let alone make machine-interoperable because there is no systematic way of specifying the relation between the raw data and its mathematical meaning. Therefore, unfortunately FAIR mathematics essentially does not exist today.

Motivation Our ultimate goal is to standardize a framework for representing mathematical datasets. As a first step, we present data.mathhub.info, an infrastructure for systematically sharing relational datasets.

Such a standard for FAIR data representations in mathematics would lead to several incidental benefits:

- increased productivity for mathematicians by allowing them to focus on the mathematical datasets themselves while leaving issues of encoding, management, and search to dedicated systems,
- improved reliability of published results as the research community can more easily scrutinize the underlying data,
- collaborations via shared datasets that are currently prohibitively expensive due to the difficulty of understanding other researchers’ data, including collaborations across disciplines and with industry practitioners, who are currently excluded due to the difficulty of understanding the datasets,
- reward mathematicians for sharing datasets (which is currently often not the case), e.g., by making datasets citable and their reuse known,
- more sustainable research by guaranteeing that datasets can be archived and their meaning understood in perpetuity (which is essential especially in mathematics).

Contribution In this section we survey and systematize how mathematical data is represented and shared and analyze how it enables or prevents FAIR mathematics. We pay particular attention to the mathematics-specific aspects of FAIR sharing, which, as we will observe, go significantly beyond the original formulation of FAIR.

As a first step towards a universal framework, and as a concrete example of FAIR-enabling mathematics-specific infrastructure, we introduce data.mathhub.info. This is a platform for sharing relational mathematical datasets in a way that systematically enables FAIRness.

3.1. General Considerations

The FAIR principles as laid out in, e.g., [**WilDumAal:FAIR16**] are strongly inspired by scientific datasets that contain arrays or tables of simple values like numbers. In these cases, it is comparatively easy to achieve FAIRness. But in mathematics and related sciences, the objects of interest are often highly structured entities which are much less uniform. Moreover, the meaning and provenance of the data must usually be given in the form of complex mathematical data themselves — not just as simple metadata that can be easily annotated. Even more critically, while datasets in other disciplines are typically meant to be shared as a whole, it is very important for mathematical datasets to find, access, operate on, and reuse individual entries or sets of entries of a dataset. As a consequence, the representation and modeling of mathematical data is much more difficult than anticipated in [**WilDumAal:FAIR16**].

There are (at least) two aspects of FAIRness that are particularly important for mathematical data and are not strongly stressed in the original principles. The first one of these is that the data need to be semantics aware. Computer applications and mathematically sound, interoperable services can only work if the mathematical meaning of the data is FAIR in all its depth. We call this “deep FAIR”.

Due to the mathematical standard of rigor and the inherent complexity of mathematical data, deep FAIRness is both more difficult and more important for mathematics than for other scientific disciplines. That also means that mathematics is an ideal test case for developing the semantic aspects of the FAIR principles in general.

The second one is that the relevant principles need to apply to every datum. The importance of this requirement, particularly for identifiers (Findable), has already been pointed out in [**BilTen:fingerprint13**]. For example, while it is good that a catalogue of graphs has a globally unique and persistent identifier, but it is much better if in addition to that, every graph in the catalogue also has one. This also extends to other FAIR principles.

In the sequel, we discuss the four FAIR principles and the challenges they pose for mathematical data in increasing order of difficulty.

Accessible While they often lack unique identifiers, most mathematical datasets are available online on researchers’ websites or via repository managers like GitHub. Barriers typical for sensitive data are rare, and open sharing is common. However, the level of accessibility desirable in practice is much higher due to the wide variety of internal structure in mathematical datasets. Access to individual entries or the rich internal structure of these entries is less common.

Because each specialized tool is typically released with its own library, often written in tool-specific language, accessibility is very good for tool-associated data, but may be practically impossible across tools.

Reusable Mathematical datasets are typically not reusable or very hard to reuse in the sense of FAIR. First of all, they are often shared without licenses with the implicit, but legally false assumption that putting them online makes them public domain. In practice, this is often unproblematic because this false assumption of the publisher may be canceled out by the same false assumption by the reuser.

More critically, the associated documentation often does not cover how precisely the data was created or how the data is to be interpreted. This documentation is usually provided in ad hoc text files or implicitly in journal papers or software source code that potential users may not be aware of and whose detailed connection to the dataset may be elusive. And the lack of a standard for associating complex semantics and provenance data effectively precludes or impedes most reuse in practice.

Findable It is not common for datasets in mathematics to be indexed in registries. One often has to first find a paper describing the dataset, and then follow a link from there. The datasets themselves are sometimes searchable (such as [**OEIS:on**; **hog**]), and the objects inside them often get a dataset-level unique identifier. This is particularly successful for bibliographic metadata (e.g. in Math Reviews, zbMATH or swMATH). However, for individual datasets, identifiers are often non-persistent, e.g., when shared on researchers’ homepages.

Finding a mathematical object by its identifier or metadata is theoretically easy. But being findable in the sense of FAIR does not always imply being findable in practice: especially in mathematics, it is much more important to find objects by their semantic properties rather than by their identifier. The indexing necessary for this is very difficult.

For example, consider an engineer who wants to prevent an electrical system from overheating and thus needs a tight estimate for the term $\int_a^b |V(t)I(t)|dt$ for all a, b, where V is the voltage and I the current. Search engines like Google are restricted to word-based searches of mathematical articles, which barely helps with finding mathematical objects because there are no keywords to search for. Computer algebra systems cannot help either since they to do not incorporate the necessary special knowledge. But the needed information is out there, e.g., in the form of

Theorem 17. (Hölder's Inequality)
If f and g are measurable real functions, $l, h \in \mathbb{R}$, and $p, q \in [0, \infty)$, such that $1/p + 1/q = 1$, then

$$\int_l^h |f(x)g(x)|\,dx \leq \left(\int_l^h |f(x)|^p\,dx\right)^{\frac{1}{p}} \left(\int_l^h |g(x)|^q\,dx\right)^{\frac{1}{q}} \tag{1}$$

and will even extend the calculation $\int_a^b |V(t)I(t)|dt \leq \left(\int_a^b |V(x)|^2\,dx\right)^{\frac{1}{2}} \left(\int_a^b |I(x)|^2\,dx\right)^{\frac{1}{2}}$ after the engineer chooses $p = q = 2$ (Cauchy-Schwarz inequality). Estimating the individual values of V and I is now a much simpler problem.

Admittedly, Google would have found the information by querying for "Cauchy-Schwarz Hölder", but that keyword itself was the crucial information the engineer was missing in the first place. In fact, it is not unusual for mathematical datasets to be so large that determining the identifier of the sought-after object is harder than recreating the object itself.

Interoperable The FAIR principle base interoperability on describing data in a "formal, accessible, shared, and broadly applicable language for knowledge representation". But due to the semantic richness of mathematical data, defining an appropriate language to allow for interoperability is a hard problem itself. Therefore, existing interoperability solutions tend to be domain-specific, limited, and brittle.

For trivial examples, consider the dihedral group of order 8, which is called D_4 in SageMath but D_8 in GAP due to differing conventions in different mathematical communities (geometry vs. abstract algebra). Similarly, $0°C$ in Europe is "called" $271.3K$ in physics. In principle, this problem can be tackled by standardizing mathematical vocabularies, but in the face of millions of defined concepts in mathematics, this has so far proved elusive. Moreover, large mathematical datasets are usually shared in highly optimized encodings (or even a hierarchy of consecutive encodings), which knowledge representation languages must capture as well to allow for data interoperability.

3.2. FAIRness for Different Kinds of Mathematical Data

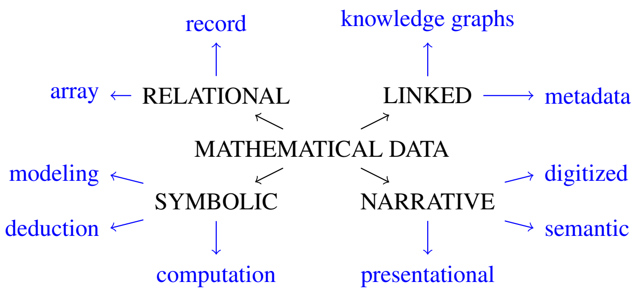


FIGURE 7. Kinds of mathematical data

In order to better analyze the current state of the art of FAIRness in mathematical data, we introduce a novel categorization of mathematical data. An overview is given in Figure 7. Each kind of data makes different abstractions or focuses on different aspects of mathematical reality, resulting in characteristic strengths and weaknesses. We summarize these in Figure 8.

Kind of data	Sym.	Rel.	Lin.	Nar.
Machine-understandable	+	+	+	−
Complete description	+	+	−	+
Applicable to all objects	+	−	+	+
Easy to produce	−	+	+	+

FIGURE 8. Advantages of different kinds of data

* Symbolic data consists of formal expressions such as formulas, formal proofs, programs, etc. These are written in a variety of highly-structured formal languages specifically designed for individual domains and with associated tools. The most important such domains are **modeling**, **deduction**, and **computation** employing modeling languages, logics, resp. programming languages. The associated tools like simulation tools, proof assistants, resp. computer algebra systems can understand the entire semantics of the data.

Because symbolic data allows for abstraction principles such as underspecification, quantification, and variable binding, it can capture the complete semantics of any mathematical object. However, the formalization of a typical narrative theorem as a statement in a proof assistant or a function in a computer algebra system can be very expensive. This comes at the price of being context-sensitive: expressions cannot be easily moved across environments, which makes *Finding*, *Reusing*, and *Interoperability* difficult.

Moreover, because each tool usually defines its own formal language and because these are usually mutually incompatible, interoperability and reuse across these individual tools are practically non-existent. To overcome this problem, multiple representation formats have been developed for symbolic data, usually growing out of small research projects and reaching different degrees of standardization, tool support, and user following. These are usually optimized for specific applications, and little cross-format sharing is possible. In response to this problematic situation, standard formats have been designed such as MathML [**CarlisleEd:MathML3:on**] and OMDoc/MMT [**uniformal:on**].

Relational data employs representation theorems that allow encoding mathematical objects as ground data built from numbers, strings, tuples, lists, etc. Thus, relational data combines optimized storage and processing with capturing the whole semantics of the objects. It is also easy to produce and curate as general purpose database technologies and interchange formats such as CSV or JSON are readily available.

Relational datasets can be subdivided based on the structure of the entries, which often enable different optimized database solutions. The most important ones are record data, where datasets are sets of records conforming to the same schema and which are stored in relational databases, and **array** data, which consists of very large, multidimensional arrays stored in optimized array databases.

However, these representation theorems do not always exist because sets and functions, which are the foundation of most mathematics, are inherently hard to represent concretely. Moreover, the representation theorems may be very difficult to establish and understand, and there may be multiple different representations for the same object. Therefore, applicability is limited and must be established on a case by case basis.

Therefore, *Interoperability* is difficult because users need to know the exact representation theorem and the exact way how it is applied to understand the encoding. Therefore, even if the representation function is documented, *Finding*, *Reuse*, and *Interoperability* are theoretically difficult, practically expensive, and error-prone. For example, consider the following very recent incident from (Jan. 2019): There are two encoding formats for directed graphs, both called `digraph6`: Brendan McKay's [**McKayFormats:on**] and the one used by the GAP package Digraphs [**GAPDigraphFormat:on**], whose authors were unaware of McKay's format and essentially reinvented a similar one [**digraph6issue:on**]. The resulting problem has since been resolved but not without causing some misunderstandings first.

Linked data introduces identifiers for objects and then treats them as blackboxes, only representing the identifier and not the original object. The internal structure and the semantics of the object remain unspecified except for maintaining a set of named relations and attributions for these identifiers. This abstraction allows for universal applicability at the price of not representing the complete mathematical object.

The named relations allow forming large networks of objects, and the attributions of concrete values provide limited information about each one. Linked data can be subdivided into **knowledge graphs** based on mathematical ontologies and **metadata**, e.g., as used in publication indexing services.

As linked data forms the backbone of the Semantic Web, linked data formats are very well-standardized: data formats come as RDF, the relations and attributes are expressed as ontologies in OWL2, and RDF-based databases (also called triplestores) can be queried via SPARQL. For example, services like DBPedia and Yago crawl various aspects of Wikipedia to extract linked data collections and provide SPARQL endpoints. The WikiData database [**wikidata:on**] collects such linked data and uses them to answer queries about the objects.

Thus, contrary to, linked data has very good FAIR-readiness, in particular allowing for URI-based *Access*, efficient *Finding* via query languages, and URI-mediated *Reuse* and *Interoperability*. However, this FAIR-readiness comes at the price of not capturing the complete semantics of the objects so that *Access* and *Finding* are limited and *Interoperability* and *Reuse* are subject to misinterpretation.

Narrative data consists of mathematical documents and text fragments. We speak of **mathematical vernacular** for the peculiar mixture of mathematical formulae, natural language with special idioms, and diagrams. There are four levels of formality of narrative data:

(1) **digitized:** scanned into images from documents,
(2) **presentational:** represented in a form that allows flexible presentation on electronic media, such as web browsers; born digital or OCRed from digitized ones,
(3) **semantic**: in a form that makes explicit the functional structure and the relations between formulae, the objects they denote and the mathematical context.

All levels of formality are relevant for mathematical communication, but machine support for reasoning and knowledge management can only be given at the semantic level. We could extend this classification by a fourth level of narrative data for formalized documents; but these abstract from the narrative form and are therefore counted as symbolic data.

Note that we can always go from higher levels to lower ones, by styling: presenting semantic features by narrative patterns. Therefore we also count such patterns as narrative data – e.g. **notation definitions** such as $\binom{n}{k}$ or C^n_k for the binomial coefficients or verbalizations in different languages.

3.3. Deep FAIRness

Relational and linked data can be easily processed and shared using standardized formats such as CSV or RDF. But in doing so, the semantics of the original mathematical objects is not part of the shared resource: in relational data, understanding the semantics requires knowing the details of the representation theorem and the encoding; in linked data, almost the entire semantics is abstracted away anyway, which also makes it hard to precisely document the semantics of the links. For datasets with very simple semantics, this can be remedied by attaching informal labels (e.g., column heads for relational data), metadata, or free-text documentation. But this is not

Service	Shallow	Deep
Identification	DOI for a dataset	DOIs for each entry
Provenance	who created the dataset?	how was each entry computed?
Validation	is this valid XML?	does this XML represent a set of polynomials?
Access	download a dataset	download a specific fragment
Finding	find a dataset	find entries with certain properties
Reuse	impractical without accessible semantics	
Interoperability	impossible without accessible semantics	

FIGURE 9. Examples of shallow and deep FAIR services

Data	Findable	Accessible	Interoperable	Reusable
Symbolic	Hard	Easy	Hard	Hard
Relational	Impossible without access to the encoding function			
Linked	Easy but only applicable to the small fragment of the semantics that is exposed			
Narrative	Hard	License-encumbered	Human-only	

FIGURE 10. Deep FAIR readiness of mathematical data

sufficient for datasets in mathematics and related scientific disciplines where the semantics is itself very complex.

For example, an object's semantic type (e.g., "polynomial with integer coefficients") is typically very different from the type as which it is encoded and shared (e.g., "list of integers"). The latter allows reconstructing the original, but only if its type and encoding function (e.g., "the entries in the list are the coefficients in order of decreasing degree") are known. Already for polynomials, the subtleties make this a problem in practice, e.g., consider different coefficient orders, sparse vs. dense encodings, or multivariate polynomials. Even worse, it is already a problem for seemingly trivial cases like integers: for example, the various datasets in the LMFDB use at least 3 different encodings for integers (because the trivial encoding of using the CPU's built-in integers does not work because the involved numbers are too big). But mathematicians routinely use much more complex objects like graphs, surfaces, or algebraic structures.

We speak of **accessible semantics** if data has metadata annotations that allow recovering the exact semantics of the individual entries of a data set. Notably, in mathematics, this semantics metadata is very complex, usually symbolic data itself that cannot be easily annotated ad hoc. But without knowing the semantics, mathematical datasets only allow FAIR services that operate on the dataset as a whole, which we call **shallow** FAIR services. But it is much more important to users to have **deep** services, i.e., services that process individual entries of the dataset.

Figure 9 gives some examples of the contrast between shallow and deep services. Note that deep services do not always require accessible semantics for every entry, e.g., deep accessibility can be realized without. But many deep services are only possible if the service can access and understand the semantics of each entry of the dataset, e.g., deep search requires checking for each entry whether it matches the search criteria.

In mathematics, shallow FAIR services are relatively easy to build but have significantly smaller practical relevance than deep FAIR services. Deep services, on the other hand, are so difficult to build that they are essentially non-existent except when built ad hoc for individual datasets. Figure 10 gives an overview of the difficulty for the different kinds of data.

Note that deep FAIR services are particularly desirable in mathematics, their advantages are by no means limited to mathematics. For example, in 2016 [**ZieEreElO:GeneErrors16**], researchers found widespread errors in papers in genomics journals with supplementary Microsoft Excel gene lists. About 20% of them contain erroneous gene name because the software misinterpreted string-encoded genes as months. In engineering, encoding mistakes can quickly become safety-critical, i.e., if a dataset of numbers is shared without their physical units, precision, and measurement type. With accessible semantics, datasets can be validated automatically against their semantic type to avoid errors such as falsely interpreting a measurement in inch as a measurement in meters, a gene name as a month, or a column-vector matrix as a row-vector matrix.

In order to support the development Deep FAIR services for mathematics, we extend the original FAIR requirements from [**WilDumAal:FAIR16**], which focused on shallow FAIR, to deep FAIR:

DF The internal structure of each object is represented and indexed in a way that allows searching for individual entries.

DA Each dataset includes a representation of the semantics of the represented objects.

DI The representation of each object uses a formal, accessible, shared, and broadly applicable language for knowledge representation, uses FAIRly shared vocabularies, and where applicable includes qualified references to other representations.

DR The representation of each object is richly described with a plurality of accurate and relevant attributes, is released with a clear and accessible data usage license, is associated with detailed provenance information, and meets domain-relevant community standards.

3.4. data.mathhub.info, a System for Deep FAIR Datasets in Mathematics

We present a unified infrastructure to support Deep FAIR for relational mathematical data. It builds on our MathHub system, a portal for narrative and symbolic mathematical data. data.mathhub.info is a part of the MathHub portal and provides storage and hosting with integrated support for Deep FAIR. In the future, this will also allow for the development of mathematical query languages (i.e., queries that abstract from the encoding) and mathematical validation (e.g., type-checking relative to the mathematical types, not the database types).

Census of relational data in mathematics In some areas of mathematics, research products can consist of listings or tabulations of complex mathematical objects and their properties. These datasets can later be used by researchers to form or refute conjectures. To facilitate the collection of information about relational data in mathematics, we set up a database with a website frontend [**bercic:cmo:table**]. The “relational data” part of Figure 2 is a subset of this database. While it grew out of the necessity to keep track of the information, it has at least two further goals. First, it aims to make it easy for anyone to see what information has been collected so far. Second, it aims to eventually make it easy to contribute information.

The information about the datasets can be displayed in different views (with switching implemented through tabs): general information, information about size, information pertaining to the FAIR principles, as well as some other properties.

Currently, the census contains about 70 datasets from several areas of mathematics. This includes links to dataset websites and author information for (nearly) all of the datasets, as well as literature references, area of mathematics and size-related information for many. Even this small sample shows large variations in terms of structure, content organisation, provenance, infrastructure and shareability, and size.

Perhaps the most important immediate use for this census is as a “market study” for data.mathhub.info. It serves as a source of use cases for the infrastructure, as well as beginnings of a community of researchers that work with mathematical data. Even in this initial stage, the census gives the developers of data.mathhub.info some idea of the requirements for the system in terms of the ranges of dataset size, complexity, etc.

We will continue to gather information about the relational datasets in mathematics in the living census website. Finally, we plan to use the new information as a basis for a more structured census.

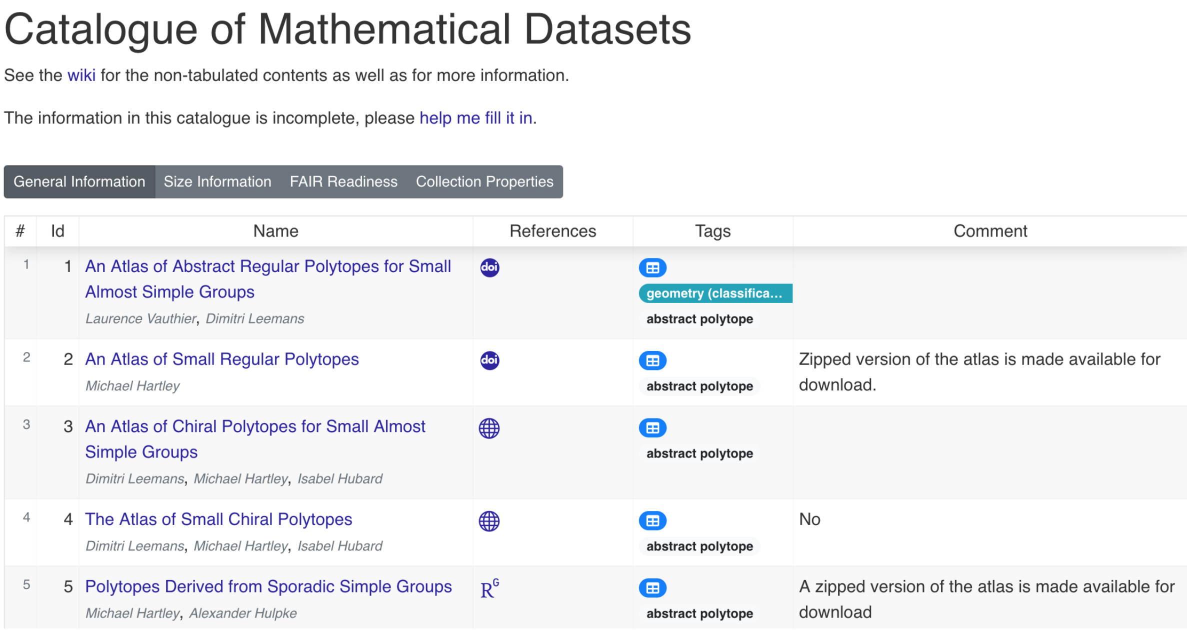


FIGURE 11. Census website

Mathematical data description language (MDDL) We developed a mathematical data description language MDDL in [**BerKohRab:tumdi19**] (Math Data Description Language) that uses symbolic data to specify the semantics of relational data. MDDL schemas combine the low-level schemas of relational database with high-level specifications (which critically use symbolic mathematical data) of the mathematical types of the data in the tables.

```
theory MatrixS : ?MDDL =
    include MitM:?IntegerMatrix
    mat: matrix ℤ 2 2 |
        meta ?Codecs?codec MatrixAsArray IntIdent |
    trace : ℤ |
        meta ?Codecs?codec IntIdent |
    orthogonal: bool |
        meta ?Codecs?codec BoolIdent | ||
```

FIGURE 12. Schema theory for Joe's dataset

To fortify our intuition let us assume that Joe has collected a set of integer matrices together with their trace and the Boolean property whether they are orthogonal. Figure 12 shows a MDDL theory that describes his database schema. For example, the mathematical type of the field mat is integer 2×2 matrices; the codec annotation specifies how this mathematical type is be encoded as a low-level database type (in this case: arrays of integers). Concretely, the codec is MatrixAsArray codec operator applied to the identity codec for integers. These codec annotations capture the representation theorem that allows representing the mathematical objects as ground data that can be stored in databases.

The information is sufficient to generate the datasets-specific components, including a web interface. The generation of APIs for computational software such as computer algebra systems is also possible and currently under development. We describe this in more detail in the following section.

Description of the data.mathhub.info prototype The prototype uses MDDL dataset descriptions to produce the necessary infrastructure for each dataset. The concept of codecs is crucial in the sense that they transparently connect the mathematical level of specification with the database level – a critical prerequisite for the deep FAIR properties postulated above. Moreover, in Figure 12, the mathematical background knowledge is imported from a theory IntegerMatrix in the Math In The Middle ontology (MitM) [**MitM:on**], which supplies the full mathematical specification and thus the basis for *Interoperability* and *Reusability*; see [**BerKohRab:tumdi19**; **WieKohRab:vtuimkb17**; **KohMuePfe:kbimss17**] for details. The overhead of having to specify the semantics of the mathematical data is offset by the fact that we can reuse central resources like the MitM ontology and codec collection. Thus, MitM and MDDL form the nucleus of a common vocabulary for typical mathematical relational datasets.

DMH_infra.pdf

FIGURE 13. data.mathhub.info Infrastructure

Currently, the infrastructure produced includes everything necessary to display a simple website with basic search functionality. The interface is shown in Figure 14, it's structure in 13. The infrastructure consists of four main parts - a database (currently either Postgres of Sqlite), a backend (written in Django), a frontend (running inside the users' browser and written in React), and a data importer.

To import data into system we start with an MDDL description of the dataset. This is used to generate a so-called schema. The schema takes the form of a JSON file and aggregates all information required for the system as a whole to function. This schema contains all dataset properties along with the codecs used for them, all human-readable dataset descriptions and other meta-data. In a first step, the importer records all this information inside the database.

Next, the encoded data is provided to the importer in the form of one or several JSON files. The importer can then use this file, and the schema, to interpret each value correctly and store it inside the database, concluding the import. During this process the importer also records appropriate data provenance.

The frontend of the system is written in the React web framework. It runs inside the users's browser and communicates with the backend via REST. Currently, it enables two main tasks: displaying all data inside a specific dataset, and searching for data subject to specific criteria.

The frontend first requests the list of properties and used codecs from the backend. Based on this information, it then uses the appropriate UI components to build a search interface or present the specific values. This architecture implies that apart from adding support for new codecs, neither the frontend nor the backend code needs to be changed when a new dataset becomes available.

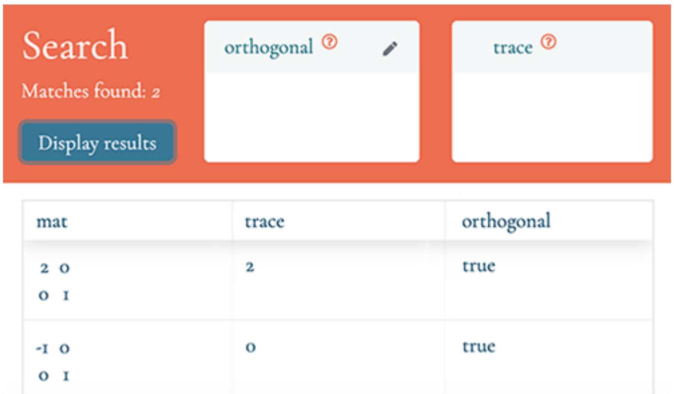


FIGURE 14. Website for Joe's dataset

The data.mathhub.info *Data Model.* The following concepts appear in the data model.[2]

- **Dataset** corresponds to a single research product. It combines information about the **Items** it contains, the properties (mathematical invariants) each item has, which codecs are used to encode the actual values, and finally, the provenance. The provenance can be composed of several provenances for sub-datasets. The **Dataset**s can overlap non-trivially: the smaller graphs in the Census of Cubic Vertex-Transitive Graphs[11] also appear in the list of Transitive Graphs)[12], which also contains graphs of other degrees.
- **Item** is a single mathematical object, represented in the system simply by a unique identifier. Examples of these include groups, graphs, lattices, and other complex mathematical objects. An **Item** can belong to more than one dataset: the Petersen graph naturally appears in both previously mentioned censuses.
- **Provenance** is information on how each datum was produced. Each **Item** needs to be linked to a provenance instance and all data produced with an instance of provenance should belong to the same dataset.
- **Property** is a mathematical invariant of a mathematical structure. An example of a such a **Property** is orthogonality of a matrix. A property can be encoded in several ways (recall the case of integers in LMFDB, where three different encodings are used). Encodings of objects (such as the `graph6` representation for graphs) are a special case of properties.
- **Codec** A codec is a pair of partial mappings between mathematical values of properties and the values represented in the database. We will describe codecs in more detail in a following section.

[2] EDNOTE: @TW: sanity-check for the data model

[11] http://staff.matapp.unimib.it/ spiga/census.html

[12] http://staffhome.ecm.uwa.edu.au/ 00013890/remote/trans/index.html

This data model already works well over a large set of examples we have examined so far. To support more datasets, we will add further concepts in the future. Some further concepts will include “Aggregated Datasets” which will be composed of several datasets and will introduce support for databases such as the OEIS and House of Graphs. These curated datasets are composed of a large number of smaller contributions.

Codecs are the glue that bind the data to the semantics and the MitM ontology. They also separate the mathematical meaning from the implementation. This separation of concerns enables optimising the mathematical layer (including the interface) and the database implementation separately. These two layers have fundamentally different goals: mathematicians (the users) do not care about the representation of data. On the other hand, the implementation needs different representations of the data for different purposes. For data.mathhub.info we expanded the notion of codecs as used previously in OpenDreamKit [**WieKohRab:vtuimkb17**] to include information for the interface.

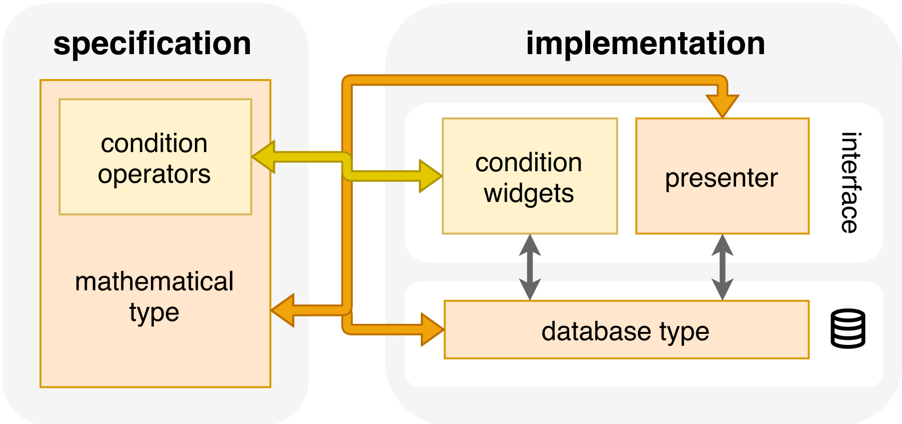


FIGURE 15. Diagram of information held by a data.mathhub.info codec

For example, let us consider one of the basic codecs, `StandardInt`. This codec encodes elements of $\mathbb{Z}$ (mathematical type) as standard [3]-bit integers (the database type). The current list of condition operators given a constant c contains the following unary operators: EdN:3

$$
\begin{array}{lll}
x \to x < c & x \to x > c & x \to x = c \\
x \to x \leq c & x \to x \geq c & x \to x \neq c
\end{array}
$$

The condition widget in the interface allows the user to enter the operator as a string, for example `"<3"`. This then gets mapped to a corresponding query at the database level. Finally, the presenter is direct and simply displays the integer value.

The Workshop on Mathematical Data in Cernay, France (August 17-24th), was dedicated to improving the status of relational data in mathematics. The workshop brought together interested users and authors of mathematical datasets, data framework developers, and experts interested in integrating mathematical databases with computer algebra systems. The workshop enabled progress on several fronts, including mayor steps in the development of data.mathhub.info. These included improving the architecture and model, as well as importing several real-life datasets into the system. We sketched out a submission and editorial process for the platform. We discussed provenance of data in mathematics and drafted a standard and formalisation of math data provenance.

At the workshop, Dr. Andrea Kohlhase tested the user experience of the existing web interface through user interviews. The list of issues identified through the interviews is available at the

[3] EDNOTE: how many?

workshop repository[13]. Dr. Andrea Kohlhase also produced a clickable prototype of an updated interface and used that in a few interviews combined with an eye-tracker test.

[13] https://github.com/OpenDreamKit/MathDataWorkshop/issues/3

4. Computational Documents

A virtual research environment (VRE) has, as a foundation, a unified DKS[4] base and a joint user interface. The OpenDreamKit approach is to create a mathematical VRE by integrating various pre-existing mathematical software systems. Furthermore, "Computational notebooks" (as reported on in [**ODK-D4.2**]) serve as an interface. EdN:4

Notebooks allow users to navigate the combined information space of all the underlying tools, systems and resources integrated into the VRE as one. However, they aggravate the already serious problem of finding anything. Thus, we decided to adapt the MathWebSearch formula search engine to this joint user interface.

4.1. The MathWebSearch Formula Search Engine

MathWebSearch is a web application that provides low-latency answers to full-text queries which consist of keywords and formulae. We give only a brief overview of the pre-existing MathWebSearch architecture here and refer the reader to [**ProKoh:mwssofse12**] and [**ODK-D6.1**]. We furthermore describe our re-worked design in Section 4.2 below. In a nutshell, MathWebSearch consists of a search web application (the MathWebSearch daemon) that indexes formula harvests (essentially lists of content MathML formulae and URI references) and answers queries (Content MathML schemata with query variables). Multiple domain-specific front-ends can talk to the MathWebSearch daemon using an XML-based protocol, they transform the user's information into Content MathML queries.

The MathWebSearch system has been used to supply search instances on various corpora of mathematical documents, we describe two here, others can be seen on `http://search.mathweb.org`.

4.1.1. *arXiv search.*

Begun on August 14, 1991, created by Paul Ginsparg, the "Cornell e-Print arXiv" (`http://arXiv.org`) is a repository of scientific papers and electronic preprints in fields of mathematics, computer science, physics, astronomy, biology and statistics or finance written in TEX/LATEX for an optimized transfer over the internet and an easily rendered client-side. In present the project is hosted by Cornell University and includes over a million articles and increases with around 8000 per month.

The KWARC group have converted by arXiv corpus into HTML5 [**StaKoh:tlcspx10**] and harvested it for MathWebSearch. The instance at `http://arxivsearch.mathweb.org` indexes over 105 000 math-heavy papers. This subcorpus has also been used for the NTCIR Math Information Retrieval Challenges [**AizKohOun:nmpto13**; **AizKohOunSch:nmto14**; **AizKohOunSch:nmto16**].

4.1.2. *zbMATH search.*

Zentralblatt Math (zbMath) is a mathematical information service that curates a database of reviews and classifications (MSC, see [**MSC2010**]) for all articles in mathematics since the middle of the 19th century. The database currently contains 3.8 million reviews and grows at a rate of ca 120 000 reviews per year. The zbMATH portal at `https://zbmath.org/` offers a faceted search engine for reviews based on bibliographic metadata, MSC classification, and *formulae*. The latter is driven by MathWebSearch.

4.2. Enabling Formula Search Deployments

The previous MathWebSearch deployments required a significant amount of application-specific code. This code was typically written by students, and as a result it was not of high quality. In particular, most code was too specialized to be re-usable.

This made it impossible to directly use MathWebSearch inside OpenDreamKit. To allow using MathWebSearch flexibly, we needed to inject the system with a lot of software engineering. To achieve this, we developed new infrastructure on top of the core MathWebSearch daemon. This

[4] EdNote: TW@MK: is this the tetrapod now

works takes the form of several components, each of which can deployed independently using Docker (see also WP3). The structure of our infrastructure can be seen in Figure 16, with new components in green.

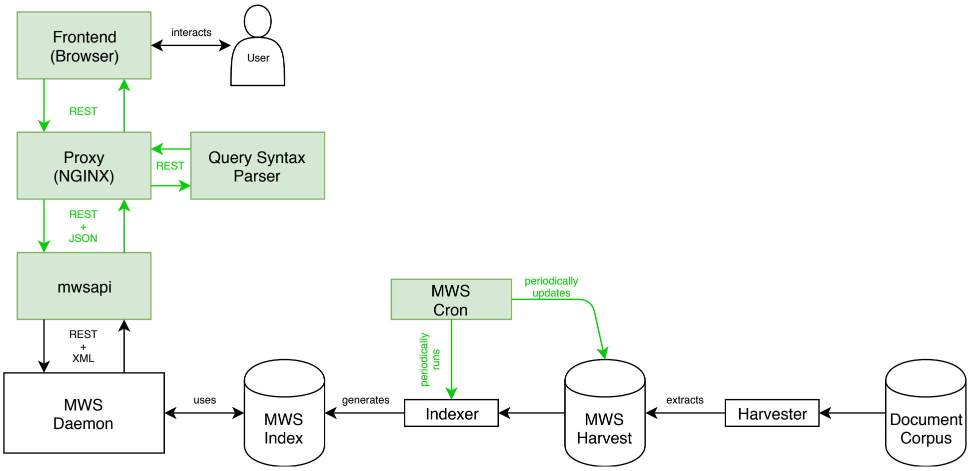


FIGURE 16. Structure of our newly developed MathWebSearch Deployment Infrastructure. Newly developed and updated components colored in green.

We describe the components of our infrastructure.

The MathWebSearch Daemon The central component is the *MathWebSearch daemon*, which can be found in the bottom left. As previously, it uses an *Index* and exposes an XML-based API for queries. The only change we made to the core daemon is that it now exists inside a Docker Container at [14].

The Harvesters and The Indexer As before, in order to generate an index from a set of documents we need two components. The indexer, as an application-independent component, has also been dockerized [15]. First we extract a set of ContentMathML formulae using a corpus-specific *Harvester* (bottom right). The generated *Harvest* is then sent to the *Indexer* (bottom center), which generates or updates the *Index*.

Additionally, we introduced a new scheduler component called *MWS cron*. This periodically sends updated harvests to the *indexer*, which then in turn updates the index. This process ensures that the Index remains up-to-date, and does not have to be re-generated manually. The source code of this component can be found at [16].

Frontend, mwsapi and the query syntax parser In order to process end user queries, we introduced several additional components.

The most important component of these is a new React-powered *frontend*[17], which runs inside the users' browser. It allows users to enter a formula search query, and view the results. The frontend contains corpus-specific branding and text, and is otherwise not corpus-specific. In particular the querying code which interacts with the backend components, does not require specialization.

While the MathWebSearch daemon only understands formulae in Content MathML, users often enter them using different representations, such as LaTeX. For this purpose, the frontend allows entering queries in human-writable, corpus-specific syntax. In order to transform the user query into a system-understood query, we make use of a new component called the *Query Syntax*

[14] `https://hub.docker.com/r/mathwebsearch/mathwebsearch/`
[15] `https://github.com/MathWebSearch/mws-indexer`
[16] `https://github.com/MathWebSearch/mws-cron`
[17] `https://github.com/MathWebSearch/frontend`

Parser. For LaTeX syntax this is achieved using LaTeXML along with a custom MathWebSearch extension, but this component is fully interchangeable if the user desires other syntaxes.

The frontend does not directly send MathML queries to the *Daemon*. Instead, it sends them to a thin API layer on top called *mwsapi*. This layer forwards the queries to the daemon and, upon receiving a response, performs some post-processing. This process includes transforming substitutions returned by MathWebSearch into a format that can be directly presented to the user by the frontend. The layer is written in go and doubles as api bindings to use MathWebSearch programatically in other applications. The source code and documentation are available on Github at [18].

As the frontend, the mwsapi server and the Query Syntax Presenter are all exposed to the end-user under the same url, a proxy delegating requests accordingly was also necessary. This is using an nginx server.

4.3. Building a Notebook Search

Sage can produce LaTeX formulae as output[5]. Using the newly developed infrastructure above, EdN:5 it was straightforward to apply MathWebSearch and enable users to search for these formulae.

Applying our infrastructure involved two steps, which we briefly describe below.

Harvesting Formulae To enable MathWebSearch to search the formulae, we needed to build a harvester that extracted formulae from a notebook and as ContentMathML formulae. To achieve this, we converted each LaTeX formula via LaTeXML and aggregated them inside an MathWebSearch Harvest file. The script to achieve this can be found at [19] and involved around 100 lines of straightforward Python script written in a few hours.

To test our harvester, we decided to make use of Sage Jupyter Notebooks found on GitHub. We implemented a second Python script which used the GitHub API to download all matching files and then apply our harvester to them. This process resulted in an MathWebSearch Harvest of 0 documents, containing 0 formulae[6]. EdN:6

Query Syntax Parser To enable users to search the formulae in an appropriate syntax, a query syntax parser was required. However, as Sage produces LaTeX formulae, we can re-use the LaTeXML query syntax parser for this purpose.

Deploying the system

The third and final step involved deploying the system. This required minor customization in the branding of the frontend. The user interface is deployed at `https://jupytersearch.mathweb.org` and can be seen in Figure 17.

In addition the notebook search, we have also exercised a similar progress for the n-category Cafe (nLab, `https://ncatlab.org/`, a community-run semantic wiki on category theory and applications with more than 13K pages). Similar to notebooks, NLab also contains formulae in LaTeX syntax, enabling us to largely re-use all components we had created previously. The NLab Harvester can be found at [20], and the corresponding frontend at `https://nlabsearch.mathweb.org`.

[18] `https://github.com/MathWebSearch/mwsapi`

[5] EdNote: TW@MK: Not sure how to describe this in more detail; should we ask Nicolas for this?

[19] `https://github.com/OpenDreamKit/jupyter-notebook-harvester/blob/master/harvest`

[6] EdNote: TW: Update numbers

[20] `https://github.com/MathWebSearch/nlab_harvester`

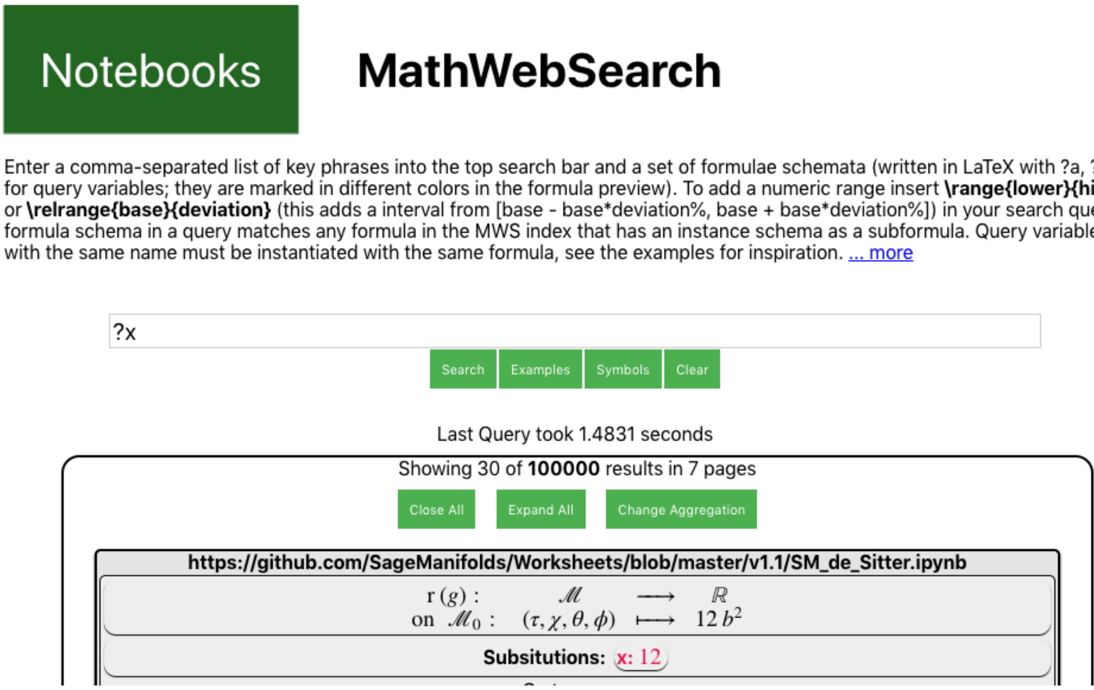


FIGURE 17. The deployed Jupyter Notebook Search Frontend

5. CONCLUSION

The main achievements made in Work Package 6 over the last year of the OpenDreamKit project were:

(1) the re-conceptualization of integrating the different aspects of doing mathematics, which led to a better understanding of the nature and intended semantics of VRE components (see Section 1.1),
(2) the integration of a major formal knowledge base into the MitM Ontology, which provides the pivotal point for system integration and service discovery (see Section 2),
(3) the development of a semantic model for mathematical datasets (see Section 3), which has been used in WP6 in two ways:
   - The Warwick group inventoried all the LFMDB datasets, and to (manually) recover their specifications (schema information) at the mathematical and data base level. In essence this retrofits the existing LFMDB project with the a more semantic level and has led to a vastly improved and more semantic API for LMFDB (see `http://www.lmfdb.org/api2/`) that has recently come online. Moreover, a SAGE interface based on this new API is currently under development.
   - The Erlangen group built a from-scratch implementation of a hub for mathematical data (see Section 3.4).
(4) special and adapted search facilities for all kinds of mathematical data and VRE components (see Section 4),
(5) a standalone implementation of persistent memoization in Python and GAP (see D6.9).

Several of these were described in detail in this report. In order to describe the general picture, we briefly go over the various parts in the rest of this section.

Knowledge We have introduced an upper ontology for formal mathematical libraries (ULO), which we propose as a community standard, and we exemplified its usefulness at a large scale. We posit ULO as an interface layer that enables a separation of concerns between library maintainers and users/application developers. Regarding the former, we have shown how ULO data can be extracted from formal knowledge libraries such as Isabelle. We encourage other library maintainers to build similar extractors. Regarding the latter, we have shown how powerful, scalable applications like querying can be built with relative ease on top of ULO datasets. We encourage other users and library-near developers to build similar ULO applications, or using future datasets provided for other libraries.

Finally, we expect our own and other researchers' applications to generate feedback on the specific design of ULO, most likely identifying various omissions and ambiguities. We will collect these and make them available for a future release of ULO 1.0, which should culminate in a standardization process.

Data We have analyzed the state of research data in mathematics with a focus on the instantiation of the general FAIR principles to mathematical data. Realizing FAIR mathematical data is much more difficult than for other disciplines because mathematical data is inherently complex, so much so that datasets can only be understood (both by humans or machines) if their semantics is not only evident but itself suitable for automated processing. Thus, the accessibility of the mathematical meaning of the data in all its depth becomes a prerequisite to any strong infrastructure for FAIR mathematical data.

Based on these observations, we developed the concept of Deep FAIR research data in mathematics. As a first step towards developing a Deep FAIR–enabling standard for mathematical datasets, we focused on relational datasets. We presented the prototypical data.mathhub.info system, which lets mathematicians integrate a dataset by specifying its semantics using a central knowledge and codec collection. We expect that data.mathhub.info also helps alleviate the problem of *disappearing datasets*: Many datasets are created in the scope of small, underfunded or unfunded research projects, often by junior researchers or PhD students, and are often abandoned when developer change research areas or pursue a non-academic career.

Software: computational mathematical documents For the S aspect of what was called D/K/S-structures in the OpenDreamKit proposal or the **narration** and **computation** aspects of the finer tetrapod model from Figure 1, we have developed a formula harvester for Jupyter notebooks and a formula search engine that builds on them.

To make this possible, we had to invest a heavy dose of software engineering into the MathWebSearch system: Even though the system has successfully been used as a formula search engine in the zbMATH publication information system (see `https://zbmath.org/formulae/`), the deployment of the system required a lot of domain-specific development and workflow integration. To this end we have developed Go bindings for the MathWebSearch daemon, documented the interfaces, and provided a web application wrapper. With this, specific applications only need a domain-specific harvester and minimal customization of our generic front-end. We have exercised that for the Jupyter Search engine (as envisioned in task **T6.10**) and analogously for a formula search engine for the n-category Cafe (nLab, see `https://nlabsearch.mathweb.org/`).

Persistent Memoization As an integration layer between computation and data, we have developed a persistent memoization infrastructure. Even though it is called "persistent" memoization, the temporal scope of the memoized data is potentially less than the eternity-scope of datasets in LMFDB and data.mathhub.info. Indeed, the characteristic innovation in D6.9 is that mathematical objects and data can be shared across multiple computations, in multiple systems of in different runs of the same system. It allows omitting the semantic level, in which case systems are required to ensure data is read in with the same meaning that it had when it was written out. But it is flexible enough to use data.mathhub.info, or a variant of it, as the physical storage of the data. Thus, the borders between persistent memoization and mathematical datasets become fluent: indeed, datasets often start as private computation caches and gradually become complete, curated, published datasets. We will study the spectrum and conversions from memoization caches to data.mathhub.info datasets in the future.

Disclaimer: this report, together with its annexes and the reports for the earlier deliverables, is self contained for auditing and reviewing purposes. Hyperlinks to external resources are meant as a convenience for casual readers wishing to follow our progress; such links have been checked for correctness at the time of submission of the deliverable, but there is no guarantee implied that they will remain valid.